ISBN 978-83-921270-4-8

Warszawska Wyższa
Szkoła Informatyki
ul. Lewartowskiego 17
00-169 Warszawa

www.wwsi.edu.pl

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

informatyka+ tom 1 Podstawy algorytmiki. Zastosowania informatyki

ZBIÓR WYKŁADÓW WSZECHNICY POPOŁUDNIOWEJ

tom 1

Podstawy algorytmiki. Zastosowania informatyki

informatyka+

Człowiek – najlepsza inwestycja

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Podstawy algorytmiki. Zastosowania informatyki

Publikacja współfinansowana ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego

Człowiek – najlepsza inwestycja

Tom 1.
Podstawy algorytmiki. Zastosowania informatyki

Redaktor merytoryczny: prof. dr hab. Maciej M. Sysło
Redaktor: Magdalena Kopacz

Publikacja opracowana w ramach projektu edukacyjnego Informatyka+
– ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych
w zakresie technologii informacyjno-komunikacyjnych (ICT)
www.informatykaplus.edu.pl
kontakt@informatykaplus.edu.pl

Wydawca:
Warszawska Wyższa Szkoła Informatyki
ul. Lewartowskiego 17, 00-169 Warszawa
www.wwsi.edu.pl
rektorat@wwsi.edu.pl

Wydanie pierwsze

Copyright©Warszawska Wyższa Szkoła Informatyki, Warszawa 2011
Publikacja nie jest przeznaczona do sprzedaży.

ISBN 978-83-921270-4-8

Projekt graficzny: FRYCZ I WICHA

Wszystkie tabele, rysunki i zdjęcia, jeśli nie zaznaczono inaczej, pochodzą z archiwów autorów lub zostały przez nich opracowane.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska
Wyższa Szkoła
Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

ZBIÓR WYKŁADÓW WSZECHNICY POPOŁUDNIOWEJ

tom 1

Podstawy algorytmiki. Zastosowania informatyki

informatyka+

PODSTAWY ALGORYTMIKI. ZASTOSOWANIA INFORMATYKI
ZBIÓR WYKŁADÓW WSZECHNICY POPOŁUDNIOWEJ

WSTĘP

Zgodnie z założeniami, projekt Informatyka+ – ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych w zakresie technologii informacyjno-komunikacyjnych (ICT) ma na celu podniesienie poziomu kluczowych kompetencji uczniów szkół ponadgimnazjalnych w zakresie informatyki i jej zastosowań, niezbędnych do dalszego kształcenia się na kierunkach informatycznych i technicznych lub podjęcia zatrudnienia, oraz stworzenie uczniom zdolnym innowacyjnych możliwości rozwijania zainteresowań naukowych w tym zakresie. Program ten jest alternatywną formą kształcenia pozalekcyjnego.

Realizacja projektu zbiegła się w czasie ze światowym kryzysem kształcenia informatycznego. Od upadku dot.comów na początku tego wieku aż o 50% spadło zainteresowanie studiami informatycznymi w Stanach Zjednoczonych i podobne tendencje zaobserwowano w Wielkiej Brytanii oraz w innych krajach. Powszechny i łatwy dostęp do najnowszej technologii komputerowej i prostota w opanowaniu jej podstawowych funkcji doprowadzają młodych użytkowników tej technologii do przekonania, że posiedli jej najważniejsze tajniki i szkoda czasu na głębsze studia w tym kierunku. Rynek pracy jednak jest w stanie wchłonąć każdą liczbę wysoko i średnio wykwalifikowanych informatyków i specjalistów z dziedzin pokrewnych.

Projekt Informatyka+ jest formą działań określanych mianem *outreach*, które są adresowane przez uczelnie do uczniów i mają na celu głębsze zaprezentowanie, czym jest informatyka, przybliżenie jej zastosowań oraz wskazanie możliwości dalszego kształcenia się w kierunkach związanych z profesjonalnym wykorzystaniem technologii komputerowej i technologii informacyjno-komunikacyjnej. Inicjatywę tę należy uznać za niezmiernie aktualną i potrzebną, wpisującą się zarówno w myślenie o przyszłości dziedziny informatyka i o przyszłych karierach młodych Polaków w zawodach informatycznych, jak i rozwoju nowoczesnego państwa. Szczegółowe informacje o projekcie i jego efektach są zamieszczane na stronie http://www.informatykaplus.edu.pl/.

Niniejszy zbiór wykładów prowadzonych w ramach Wszechnicy Popołudniowej, stanowiącej jedną z form realizacji projektu, oddajemy przede wszystkim do rąk uczniów. Tom 1 zawiera wykłady z zakresu: Tendencje w rozwoju informatyki i jej zastosowań oraz Algorytmika i programowanie.

Pierwsza grupa tematów reprezentuje szerokie spektrum zastosowań informatyki w różnych dziedzinach, takich jak kryptografia, ekonomia i medycyna, oraz wykorzystania metod informatycznych przy rozwiązywaniu rzeczywistych problemów, jak wnioskowanie i wyszukiwanie informacji w Internecie.

Druga grupa zagadnień stanowi łagodne wprowadzenie do algorytmiki, która jest bazą dla komputerowego rozwiązywania problemów. Przedstawiono podstawowe problemy i możliwie najefektywniejsze algorytmy ich rozwiązywania. Algorytmom towarzyszą odpowiednio dobrane struktury danych, a głównym kryterium doboru problemów i algorytmów jest złożoność obliczeniowa ich rozwiązywania. Omówiono problemy, które można rozstrzygnąć bardzo efektywnie, a także te, dla których dotychczas nie wymyślono praktycznych i użytecznych metod rozwiązywania.

Do lektury zamieszczonych tekstów wystarczy znajomość matematyki i informatyki na poziomie szkoły ponadgimnazjalnej. Nowe pojęcia są wprowadzane na przykładach i w sposób intuicyjny, umożliwiający nadążanie za tokiem wykładu.

Prof. dr hab. Maciej M. Sysło
Merytoryczny Koordynator
Projektu Informatyka+

Warszawa, jesienią 2011 roku

Tendencje
w rozwoju informatyki

Informatyka – klucz do zrozumienia, kariery, dobrobytu

Czy komputery będą robić biznes

Algorytmika Internetu

Jak wnioskują maszyny

Język językowi nie równy

Programowanie współbieżne w informatyce i nie tylko

Jak informatyka pomaga zajrzeć do wnętrza ludzkiego ciała

Naśladowanie żywego mózgu w komputerze

Od złamania Enigmy do współczesnej kryptologii

Od abaków do maszyny ENIAC i Internetu

Informatyka – klucz do zrozumienia, kariery, dobrobytu

Maciej M. Sysło

Uniwersytet Wrocławski, UMK w Toruniu

syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl

http://mmsyslo.pl/

Streszczenie

Wykład składa się z dwóch części – w pierwszej krótko omówiona jest kwestia edukacji informatycznej, jej stan, wzloty i upadki, cele oraz dalsze kierunki rozwoju. Opis edukacji informatycznej i informatyki jako dziedziny stanowi tło dla rozważań na temat możliwości kariery na tym polu, co ma zachęcić słuchaczy do poważnego zainteresowania się rozwijaniem swoich informatycznych umiejętności, włącznie z podjęciem studiów na kierunku informatyka lub pokrewnym. Przedstawione są najpierw kariery w informatyce, tylko te z klasą, jak John Napier, autorzy RSA, Samuel Morse i David Huffman, Claude Shannon oraz anonimowi wynalazcy patentów związanych z maszynami do pisania. Krótko komentujemy również znaczenie dla informatyki i jej zastosowań osiągnięć współczesnych, którym poza klasą towarzyszy również „kasa". Ostatnia część wykładu jest poświęcona omówieniu wybranych wyzwań, czyli problemów, które czekają na adeptów informatyki. Wśród nich jest problem komiwojażera, cała gama problemów związanych z liczbami pierwszymi oraz jeden z problemów milenijnych (czy P = NP).

Spis treści

1 WPROWADZENIE

Tytuł tego wykładu nawiązuje do tytułu książki Andrzeja Targowskiego *Informatyka – klucz do dobrobytu*, która ukazała się w 1971 roku. Istniało wtedy silne przekonanie wśród osób zajmujących się informatyką, że komputery i cała dziedzina z nimi związana może przyczynić się do znaczącego poprawienia warunków życia nie tylko osób zajmujących się informatyką, ale całego społeczeństwa. Tak się jednak nie stało, chociaż wykorzystanie komputerów przyczynia się do rozwoju dziedzin, w których są używane, również w życiu zwykłych obywateli. Jednym z celów tego wykładu jest przekonanie słuchaczy, że droga do dobrobytu, na której pojawiają się komputery i informatyka, wiedzie w pierwszym rzędzie przez zrozumienie ich istoty, działania, możliwości, kierunków rozwoju, a także ograniczeń. Wbrew powszechnemu mniemaniu istnieją obliczenia, których nie jest w stanie wykonać żaden komputer, co więcej – nawet wszystkie istniejące na świecie komputery razem wzięte tego nie potrafią. Cała nadzieja w nowych algorytmach i rozwiązaniach.

Wykład jest adresowany do uczniów, których chcemy zainteresować informatyką tak, aby ta dziedzina stała się ich ulubioną i w konsekwencji, by podjęli studia na kierunkach informatycznych lub o zbliżonych profilach i wiązali z nimi przyszłość.

Informatyka to obecnie ugruntowana dziedzina wiedzy, której zastosowania można znaleźć niemal w każdej innej dziedzinie. Ilustrują to inne wykłady w tym projekcie, dotyczące na przykład: ekonomii, kryptografii, gier, medycyny, mózgu. Wokół informatyki narosło niestety wiele nieporozumień, na ogół związanych z tym, że obecnie łatwo można posiąść podstawowe umiejętności posługiwania się komputerem i jego oprogramowaniem ani nie będąc informatykiem, ani nie kształcąc się w tym kierunku.

Jednak niemal każdy człowiek posługując się komputerem powinien w jakimś zakresie znać głębiej jego działanie, a zwłaszcza sposoby jego wykorzystania w różnych sytuacjach i do rozwiązywania różnych problemów. Piszemy o tym w p. 2.3.

W dalszej części, dla kilku wybranych problemów ilustrujemy ich własności i rozwiązania oraz komentujemy rolę tych problemów w rozwoju metod komputerowych. Niektóre z przedstawionych problemów nadal stwarzają duże wyzwanie dla informatyków, zarówno pracujących nad konstrukcjami nowych komputerów, jak i nad coraz doskonalszym wykorzystaniem tych istniejących.

Z wieloma z opisywanych faktów, problemów ich rozwiązań są związane znane w świecie informatycznym nazwiska. Za wieloma stoją osoby, zajmujące wysokie i najważniejsze pozycje wśród najbogatszych ludzi świata. Chcemy Was przekonać, że jedni i drudzy reprezentują najwyższą klasę informatyków, a Ci najbogatsi, dodatkowo... mają z tego olbrzymie pieniądze. Źródeł sukcesów jednych i drugich można się doszukiwać głównie w ich osiągnięciach na polu informatyki.

2 KSZTAŁCENIE INFORMATYCZNE

W tym rozdziale przedstawiamy lepsze i gorsze czasy zajęć z informatyki oraz cele powszechnego kształcenia informatycznego.

2.1 PIERWSZE ZAJĘCIA Z INFORMATYKI

Pierwsze regularne zajęcia z informatyki w polskiej szkole miały miejsce w połowie lat 60. XX wieku – był to przedmiot „Programowanie i obsługa maszyn cyfrowych" prowadzony w III LO we Wrocławiu. Uczniowie pisali programy w zeszytach, a później uruchamiali je na komputerze Elliott 803 w Katedrze Metod Numerycz-

nych Uniwersytetu Wrocławskiego. Programy służyły do wykonywania obliczeń matematycznych. Przed erą komputerów osobistych niewiele więcej można było robić z pomocą komputerów.

Dopiero z pojawieniem się IBM PC na początku lat 80. minionego wieku stało się możliwe rzeczywiste upowszechnianie nauczania informatyki. Pierwszy program nauczania przedmiotu **elementy informatyki** dla liceów powstał w 1985 roku, a w 1990 roku – dla szkół podstawowych. Na początku lat 90. ukazał się pierwszy podręcznik do elementów informatyki. Był on bardzo uniwersalny, gdyż w niewielkim stopniu zależał od konkretnego oprogramowania – miał aż 9 wydań i do dzisiaj jest czasem wykorzystywany na niektórych zajęciach.

Wydzielone zajęcia informatyczne w polskich szkołach były bardzo poważnie traktowane w kolejnych reformach systemu oświaty i nigdy pod żadnym naciskiem przedmiot informatyka nie został usunięty ze szkół, chociaż taki przykład płynął ze Stanów Zjednoczonych, gdzie od lat 90. komputery w szkołach były wykorzystywane głównie do kształcenia umiejętności z zakresu technologii informacyjno-komunikacyjnej. Obecnie w Stanach Zjednoczonych wraca się do kształcenia w zakresie informatyki, co ma powstrzymać spadek zainteresowania uczniów karierami informatycznymi, wynoszący w ostatnich latach aż 50% (podobnie jest w Wielkiej Brytanii).

2.2 REGRES EDUKACJI INFORMATYCZNEJ

Zmniejszone zainteresowanie uczniów kształceniem informatycznym w szkołach jest obserwowane nie tylko w Stanach Zjednoczonych. Powodów tego jest wiele. Z jednej strony wiele osób, w tym nauczyciele i rodzice, nie uważa informatyki za niezależną dziedzinę nauki, a zatem także za szkolny przedmiot. Powszechnie myli i utożsamia się informatykę z technologią informacyjno-komunikacyjną i sprowadza edukację informatyczną do udostępniania uczniom i nauczycielom komputerów i Internetu w szkole i w domu. Nie odróżnia się stosowania komputerów i sieci Internet od studiowania podstaw informatyki. W Polsce zmniejszanie się liczby kandydatów na studia informatyczne jest spowodowane również przez niż, na szczęście okresowy, oraz, w poszczególnych uczelniach, powstawaniem kolejnych szkół prywatnych.

Jest też wiele powodów obniżonego zainteresowania samych uczniów informatyką jako dziedziną kształcenia i przyszłą karierą zawodową. Na początku informatyka była kojarzona z programowaniem komputerów, co wywoływało silny sprzeciw, gdyż uważano, że niewielu uczniów zostanie kiedyś programistami. W końcu lat 80. i początku lat 90. tylko nieliczni uczniowie używali komputerów w szkole lub w domu przed wstąpieniem na uczelnię. Na przełomie XX i XXI wieku główny nacisk w szkołach zmienił się diametralnie – kształcono z zakresu korzystania z aplikacji biurowych i Internetu. Obecnie wielu przyszłych studentów zdobywa pierwsze doświadczenia informatyczne przed rozpoczęciem studiów najczęściej poza szkołą. Co więcej, dostępne oprogramowanie umożliwia tworzenie nawet bardzo złożonych aplikacji komputerowych bez wcześniejszego zaznajomienia się z: logiką, metodami programowania, matematyką dyskretną, które należą do kanonu kształcenia informatycznego. W rezultacie, absolwenci szkół średnich nieźle radzą sobie z wykorzystaniem komputerów do zabawy, poszukiwań w sieci i do komunikowania się, ale znikoma jest ich wiedza na temat informatyki jako dyscypliny oraz o tym, jak funkcjonuje komputer i sieć komputerowa. Dorastając mają oni na tyle dość styczności z technologią informatyczną, że nie interesuje ich rozwijanie swoich umiejętności w tym zakresie na poziomie uczelni, a w konsekwencji – kreowanie nowej kultury i nowej technologii. Potrzebny jest więc sposób, jak zmotywować uczniów, aby zainteresowali się tym, co dzieje się poza ekranem komputera, jak zbudowany jest komputer i sieć oraz jak działa oprogramowanie, a w dalszej perspektywie tworzyli własne rozwiązania informatyczne.

2.3 POTRZEBA ZMIAN

Chcąc zmienić opisaną sytuację, zajęcia informatyczne w szkołach powinny przygotowywać uczniów do dalszego kształcenia się w kierunkach związanych z informatyką, zamiast utwierdzać ich w przekonaniu, że ich

wiedza i umiejętności w tym zakresie są wystarczające. Czasem uczniowie są niezadowoleni i zniechęceni sposobem prowadzenia w szkole zajęć informatycznych.

Badania rynku zatrudnienia i potrzeb społecznych potwierdzają jednak, że nadal będą potrzebni eksperci i specjaliści z różnych obszarów informatyki i jej zastosowań. Dlatego duże znaczenie należy przywiązywać do przygotowania uczniów ze szkół, by w przyszłości mogli świadomie wybrać studia informatyczne i karierę zawodową związaną z informatyką.

Warto uwzględnić także, że poszerza się gama zawodów określanych mianem **IT Profession**, czyli zawodów związanych z profesjonalnym wykorzystywaniem zastosowań informatyki i technologii informacyjno-komunikacyjnej. Pracownicy tych profesji albo są informatykami z wykształcenia, albo najczęściej nie kończyli studiów informatycznych, jednak muszą profesjonalnie posługiwać się narzędziami informatycznymi. Do IT Profession zalicza się np. specjalistów z zakresu bioinformatyki, informatyki medycznej, telekomunikacji, genetyki itp. – wszyscy oni muszą umieć „programować" swoje narzędzia informatyczne. Informatyk ich w tym nie wyręczy, gdyż nie potrafi. W Stanach Zjednoczonych do IT Profession zalicza się obecnie ponad 40 zawodów, w których profesjonalnie są wykorzystywane zastosowania z dziedziny informatyki, i ta lista stale się powiększa.

2.4 CELE ZAJĘĆ INFORMATYCZNYCH

Kształcenie na wydzielonych przedmiotach informatycznych w szkole było początkowo (lata 80.-90. XX wieku) skupione na prostej **alfabetyzacji komputerowej**, czyli podstawach posługiwania się komputerem i siecią. Na przełomie XX i XXI wieku alfabetyzacja została poszerzona do **biegłości komputerowej**, przygotowującej również na zmiany w technologii, by np. nie uczyć się o kolejnych wersjach pakietu Office. Dużym wyzwaniem jest oparcie kształcenia informatycznego wszystkich uczniów na idei tzw. **myślenia komputacyjnego,** czyli na bazie metod rozwiązywania problemów z różnych dziedzin z pomocą komputerów. Podobnie jak maszyny drukarskie przyczyniły się do upowszechnienia kompetencji w zakresie 3R (*reading, writing, arithmetic*), tak dzisiaj komputery i informatyka przyczyniają się do upowszechniania myślenia komputacyjnego, związanego z posługiwaniem się komputerem.

Myślenie komputacyjne, towarzyszące procesom rozwiązywania problemów za pomocą komputerów, można scharakteryzować następującymi cechami:

- problem jest formułowany w postaci umożliwiającej posłużenie się w jego rozwiązaniu komputerem lub innymi urządzeniami;
- problem polega na logicznej organizacji danych i ich analizie, czemu mogą służyć m.in. modele danych i symulacje modeli;
- rozwiązanie problemu można otrzymać w wyniku zastosowania podejścia algorytmicznego, ma więc postać ciągu kroków;
- projektowanie, analiza i komputerowa implementacja (realizacja) możliwych rozwiązań prowadzi do otrzymania najbardziej efektywnego rozwiązania i wykorzystania możliwości i zasobów komputera;
- nabyte doświadczenie przy rozwiązywaniu jednego problemu może zostać wykorzystane przy rozwiązywaniu innych sytuacji problemowych.

Przestrzeganie tych etapów posługiwania się komputerem w różnych sytuacjach problemowych ma zapewnić, by rozwiązania problemów czy realizacje projektów były:

- w dobrym stylu i czytelne dla wszystkich tych, którzy interesują się dziedziną, do której należy rozwiązywany problem lub wykonywany projekt;
- poprawne, czyli zgodne z przyjętymi w trakcie rozwiązywania założeniami i wymaganiami;
- efektywne, czyli bez potrzeby nie nadużywają zasobów komputera, czasu działania, pamięci, oprogramowania, zasobów informacyjnych.

2.5 POPRAWA SYTUACJI

Świadomość, że maleje zainteresowanie uczniów studiowaniem na kierunkach technicznych, ścisłych i przyrodniczych, w tym również na informatyce, powoduje podejmowanie różnych działań zaradczych. W Stanach Zjednoczonych powstała specjalna inicjatywa federalna pod nazwą **STEM** (*Science, Technology, Engineering, Mathematics*), w ramach której prowadzone są liczne działania, które mają na celu przynajmniej powrót do sytuacji sprzed 10 lat.

W Polsce inicjatywy tego typu można podzielić na dwie grupy, w obu przypadkach są wspierane przez fundusze UE. Z jednej strony, Ministerstwo Nauki i Szkolnictwa Wyższego wspiera kierunki deficytowe, które z kolei oferują studentom dość wysokie stypendia (ok. 1000 zł), z drugiej strony – Ministerstwo Edukacji Narodowej i organy samorządowe prowadzą projekty, których celem – pod hasłem: Człowiek – najlepsza inwestycja – jest wspieranie rozwoju wiedzy i umiejętności w dziedzinach deficytowych.

Jednym z takich projektów jest **Informatyka +** (http://www.informatykaplus.edu.pl/), w ramach którego jest wygłaszany ten wykład. Weźmie w nim udział w ciągu trzech lat ponad 15 tys. uczniów ze szkół ponadgimnazjalnych z pięciu województw. Celem tego projektu jest podwyższenie kompetencji uczniów szkół ponadgimnazjalnych z zakresu informatyki i jej zastosowań, niezbędnych do dalszego kształcenia się na kierunkach informatycznych i technicznych lub podjęcia zatrudnienia, oraz stworzenie uczniom zdolnym innowacyjnych możliwości rozwijania zainteresowań w tym zakresie. Program ten jest formą kształcenia pozalekcyjnego, które ma służyć zarówno zwiększeniu zainteresowania uczniów pogłębionym edukowaniem w zakresie współczesnych technologii informacyjno-komunikacyjnych, jak i podniesieniu ich osiągnięć w tym obszarze.

Program Informatyka + jest też przykładem działań określanych mianem ***outreach***, polegających na tym, że uczelnia wyższa wraz ze swoimi pracownikami naukowo-dydaktycznymi stara się przedstawić uczniom ze szkół swoje działania i zachęcić do podjęcia kształcenia w kierunkach reprezentowanych przez uczelnię. Ta prezentacja uczelni i kierunków kształcenia przyjmuje formę zajęć prowadzonych przez pracowników uczelni.

3 CO TO JEST INFORMATYKA

Chociaż źródeł informatyki można się doszukać w różnych dziedzinach nauki i techniki, informatyka jako dziedzina zaczęła rodzić się wraz z pojawianiem się komputerów i dzisiaj jest kojarzona z tymi urządzeniami, chociaż w ostatnich latach przechodzą one głęboką ewolucję. Można powiedzieć, że **informatyka** zajmuje się projektowaniem, realizacją, ocenianiem, zastosowaniami i konserwacją systemów przetwarzania informacji z uwzględnieniem przy tym aspektów sprzętowych, programowych, organizacyjnych i ludzkich wraz z implikacjami przemysłowymi, handlowymi, publicznymi i politycznymi. Wspomniane systemy przetwarzania informacji na ogół bazują na rozwiązaniach komputerowych, a w ogólności – mikroprocesorowych (jak telefony komórkowe). Z kolei informacje mogą mieć najprzeróżniejszą postać. Na początku były to tylko liczby, ale z czasem stało się możliwe przetwarzanie tekstów, a później również grafiki, dźwięków i filmów.

Sam termin informatyka pojawił się w języku polskim jako odpowiednik terminu angielskiego *computer science*. Podobnie brzmią terminy w języku francuskim *informatique* i niemieckim *Infomatik*.

Nieustannie rozszerzające się zastosowania informatyki w społeczeństwie oraz zwiększenie roli komputerów w komunikacji i wymianie informacji miało wpływ na pojawienie się nowej dziedziny, technologii informacyjno-komunikacyjnej (ang. ICT – *Information and Communication Technology*), która znacznie wykracza swoim zakresem poza tradycyjnie rozumianą informatykę. Przyjmuje się, że **technologia informacyjno-komunikacyjna (TIK)** to zespół środków (czyli urządzeń, takich jak komputery i ich urządzenia zewnętrzne oraz sieci komputerowe) i narzędzi (czyli oprogramowanie), jak również inne technologie (jak telekomunikacja), które służą wszechstronnemu posługiwaniu się informacją. TIK obejmuje więc swoim zakresem m.in.: informację, komputery, informatykę i komunikację. Technologia informacyjno-komunikacyjna jest więc połączeniem zastosowań informatyki z wieloma innymi technologiami pokrewnymi.

Warto bliżej przyjrzeć się coraz bardziej popularnemu pojęciu technologii informacyjno-komunikacyjnej, które wyłaniało się wraz z rozwojem komputerów, sieci komputerowych i oprogramowania. W języku polskim, ten termin jest wiernym odpowiednikiem określenia w języku angielskim i niesie w sobie to samo znaczenie. Wątpliwości może budzić połączenie słowa technologia (określenie związane z procesem) zwłaszcza ze słowem informacja (w tradycyjnym sensie jest to obiekt o ustalonej formie zapisu). To połączenie ma jednak głębokie uzasadnienie we współczesnej postaci informacji i w sposobach korzystania z niej. **Informacji towarzyszą bowiem nieustannie procesy i działania.** Zarówno sam obiekt – informacja, szczególnie umieszczona w sieci Internet w każdej chwili ulega zmianie (poszerzeniu, aktualizacji, dopisaniu powiązań, nowym interpretacjom itd.), jak i korzystanie z niej jest procesem. Nie tylko sięgamy po nią, jak po fragment zapisany w książce (np. w encyklopedii) stojącej na półce, ale – pisząc odpowiednią frazę i wydając polecenie dla komputerowego systemu wyszukiwania informacji, znajdujących się na różnych nośnikach (w tym m.in. w sieci) – uruchamiamy proces jej uformowania w wybranym zakresie i postaci.

Informatyka jest obecnie dziedziną naukową równoprawną z innymi dziedzinami, którą można studiować i w której można prowadzić badania naukowe. Studia informatyczne można podejmować na uczelniach o różnych profilach kształcenia, np. uniwersyteckim, technicznym, ekonomicznym.
W ostatnich latach coraz większą popularnością zwłaszcza w Stanach Zjednoczonych cieszy się termin ***computing***[1], który nie ma ugruntowanego odpowiednika w języku polskim. Przekłada się ten termin na **komputyka**. Informatyka, rozumiana tradycyjnie jako odpowiednik *computer science*, jest jednym z pięciu kierunków studiowania w ramach komputyki według standardów amerykańskich (IEEE, ACM), są to:

- *Computer Engineering* – budowa i konstrukcja sprzętu komputerowego;
- *Information Systems* – tworzenie systemów informacyjnych;
- *Information Technology* – technologia informacyjna, zastosowania informatyki w różnych dziedzinach;
- *Software Engineering* – produkcja oprogramowania;
- *Computer Science* – studia podstawowe, uniwersyteckie studia informatyczne.

Warto jeszcze przytoczyć inne sformułowania związane z określeniem, co to jest informatyka. Na ogół dotykają one wybranych aspektów tej dziedziny.

Uważa się nie bez powodów, że:

informatyka jest dziedziną wiedzy i działalności zajmującą się algorytmami.

Przy okazji warto wspomnieć, że za pierwszy algorytm uważa się algorytm Euklidesa podany 300 lat p.n.e, gdy jeszcze nie istniało pojęcie algorytm, a o komputerach czy maszynach liczących nikt jeszcze nie myślał. Może się wydawać, że spojrzenie na informatykę przez pryzmat algorytmów jest bardzo ograniczone. Zapew-

[1] *Computing* określa się jako, (...) *any goal-oriented activity requiring, benefiting from, or creating computers. Thus, computing includes designing and building hardware and software systems for a wide range of purposes; processing, structuring, and managing various kinds of information; doing scientific studies using computers; making computer systems behave intelligently; creating and using communications and entertainment media; finding and gathering information relevant to any particular purpose, and so on. The list is virtually endless, and the possibilities are vast.* Computing Curricula 2005, ACM, IEEE, 2006. Proponowanym przykładem tego terminu posługują się w swoich wystąpieniach i publikacjach ks. Józef Kloch i Andrzej Walat.

ne tak, chociaż komputer jest urządzeniem, które tylko wykonuje programy, a każdy program jest zapisem jakiegoś algorytmu. A więc w tej definicji jest zawarte zarówno programowanie (jako zapisywanie algorytmów), jak i komputer (jako urządzenie do ich wykonywania). W tym określeniu można dopatrzyć się również sieci, która jest medium działającym na bazie odpowiednich algorytmów komunikacyjnych.

Ostatecznie nie byłoby informatyki, gdyby nie było komputerów. Słowo komputer pochodzi od angielskiego słowa *computer*, które w pierwszym rzędzie oznacza osobę, która wykonuje obliczenia, a dopiero na drugim miejscu określa urządzenie służące do obliczeń[2]. Zanim ten termin zadomowił się w języku polskim, komputery nazywano (matematycznymi) maszynami liczącymi. Pierwsze pojawienie się słowa *computer* w odniesieniu do urządzeń liczących to koniec XIX wieku, gdy firma o nazwie Rapid Computer z Chicago zaczęła wytwarzać proste urządzenia do dodawania o nazwie *comptometer*.

Nie należy jednak sprowadzać informatyki do dziedziny zajmującej się komputerami. Bardzo zgrabnie to ujął holenderski informatyk Edsger Dijkstra (znane są algorytmy Dijkstry w teorii grafów):

Informatyka jest w takim sensie nauką o komputerach,
jak biologia jest nauką o mikroskopach,
a astronomia – nauką o teleskopach.

Dobrze jest więc pamiętać, że w zajmowaniu się informatyką i korzystaniu z jej osiągnięć, komputer jest głównie narzędziem. Komputer może być jednak bardzo pomocnym narzędziem w pracy intelektualnej, wręcz partnerem w „dyskusji" z nim za pośrednictwem algorytmów. Tutaj z kolei pasują słowa innego znanego informatyka, Amerykanina Donalda Knutha:

Mówi się często, że człowiek dotąd nie zrozumie czegoś,
zanim nie nauczy tego – kogoś innego.
W rzeczywistości,
człowiek nie zrozumie czegoś naprawdę,
zanim nie zdoła nauczyć tego – komputera.

Na zakończenie tych ogólnych rozważań o komputerach i informatyce warto jeszcze skomentować dość częste przekonanie o wszechmocy komputerów. Zapewne komputery są w stanie szybko wykonywać polecenia, które im wydajemy. Zilustrujemy jednak w dalszej części, że istnieje wiele problemów, przy rozwiązywaniu których komputery są bezradne. Jest jednak szansa, by usprawnić ich działanie – cała nadzieja w szybkich algorytmach, jak to zgrabnie ujął Ralf Gomory, były szef ośrodka badawczego IBM:

Najlepszym sposobem przyspieszania komputerów
jest obarczanie ich mniejszą liczbą działań.

4 KARIERY W INFORMATYCE

Zachętą do zainteresowania się jakąś dziedziną oraz obrania jej jako obszaru kariery zawodowej mogą być sukcesy zawodowe wybranych przedstawicieli tej dziedziny. Dzisiaj wymienia się wiele osób, które w informatyce zrobiły karierę, a miernikiem ich kariery są miejsca na liście najbogatszych osób świata. Wśród nich, w kolejności na tej liście od najwyżej notowanej pozycji znajdują się następujące osoby związane z różnymi obszarami nowych technologii [dane z września 2011]: Bill Gates (Microsoft), Larry Ellison (Oracle), Jeff Bezos (Amazon), Mark Zuckerberger (Facebook), Sergey Brin (Google), Larry Page (Google), Michael Dell (Dell), Steve Ballmer (Microsoft), Paul Allen (Microsoft), Steve Jobs (Apple). To są kariery z ostatnich lat i można powiedzieć, że to „kariery z kasą". Skomentujemy je jeszcze dalej – okaże się, że ta „kasa" jest pochodną pewnego pomysłu oraz konsekwencji w jego rozwijaniu i wdrażaniu. Okaże się, że droga do dobrobytu w przypadku tych osób wiedzie przez zrozumienie oraz twórcze i innowacyjne działanie w obranym kierunku.

[2] **computer** – **1.** a person who computes **2.** a device used for computing, Webster's New World Dictionary, Simon and Schuster, 1969.

Zanim wrócimy do wymienionych wyżej postaci, chcielibyśmy zwrócić uwagę na kariery w informatyce, które klasyfikujemy jako „kariery z klasą", a są związane z odkryciami epokowymi dla zastosowań komputerów.

4.1 KARIERY Z KLASĄ

Przedstawiamy tutaj wybrane osiągnięcia, które miały przełomowe znaczenie w rozwoju zastosowań informatyki. Chociaż osoby związane z tymi odkryciami na trwałe zapisały się w historii rozwoju myśli, a ich odkrycia należą do kanonu wiedzy informatycznej, ani w swoich czasach, ani tym bardziej teraz nie znajdziemy ich na liście najbogatszych osób czerpiących zyski ze swoich osiągnięć.

4.1.1 LOGARYTM

Zaczniemy od logarytmu, który jest wszechobecny w informatyce.

Komentarz, uwagi historyczne

W przeszłości uzasadnieniem dla posługiwania się logarytmem były jego własności, które legły u podstaw wprowadzenia go do matematyki, a dokładniej – do obliczeń. Logarytm ułatwia wykonywanie złożonych obliczeń, na przykład dzięki zastąpieniu działań multiplikatywnych, takich jak mnożenie i dzielenie, przez dodawanie i odejmowanie. Nie tak dawno jeszcze w szkołach posługiwano się tablicami logarytmicznymi, a w uczelniach i w pracy przyszli i zawodowi inżynierowie korzystali z suwaków logarytmicznych.

Odkrywcą logarytmu był matematyk szkocki John Napier (1550-1617), a suwak logarytmiczny wynalazł William Oughtred (1575-1660) w 1632 roku. Na rysunku 1 są pokazane różne typy suwaków: prosty, cylindryczny (model Otis King), na walcu (model Fullera) i okrągły. Różnią się one długością skali, od czego zależy dokładność obliczeń – prosty ma skalę o długości 30 cm, cylindryczny – 1,5 metra, a na walcu – ponad 12 metrów.

Rok 1972 to początek agonii suwaków – zaczęły je wypierać kalkulatory elektroniczne stworzone za pomocą... suwaków. Ponad 40 milionów wcześniej wyprodukowanych suwaków stało się nagle bezużyteczne i obecnie stanowią głównie eksponaty kolekcjonerskie, jak te na ilustracjach (należą one do kolekcji autora).

Dzisiaj jednak nie można wyobrazić sobie zajmowania się informatyką, nawet na najniższym poziomie w szkole, bez przynajmniej „otarcia" się o logarytmy. Logarytm pojawia się, gdy chcemy uzyskać odpowiedź na następujące pytania:

- ile należy przejrzeć kartek w słowniku, aby znaleźć poszukiwane hasło?
- ile miejsca w komputerze, a dokładniej – ile bitów, zajmuje w komputerze liczba naturalna?
- jak szybko można wykonywać potęgowanie dla dużych wartości wykładników potęg?
- ile trwa obliczanie największego wspólnego dzielnika dwóch liczb za pomocą algorytmu Euklidesa?
- a ogólniej – ile kroków wykonuje algorytm typu dziel i zwyciężaj, zastosowany do danych o *n* elementach?

O znaczeniu i „potędze" logarytmów i funkcji logarytmicznej w informatyce, a ogólniej – w obliczeniach decyduje szybkość wzrostu jej wartości, **nieporównywalnie mała** względem szybkości wzrostu jej argumentu, co ilustrujemy w tabeli 1. Zauważmy, że dla liczb, które mają około stu cyfr, wartość logarytmu wynosi **tylko** ok. 333.

Rysunek 1.
Suwaki logarytmiczne: prosty, cylindryczny, na walcu i okrągły

Tabela 1.
Wartości funkcji logarytm dla przykładowych argumentów

n	$\log_2 n$
128	7
1 024	10
65 536	16
1 048 576	20
10^{10}	ok. 34
10^{20}	ok. 67
10^{30}	ok. 100
10^{50}	ok. 167
10^{100}	ok. 333
10^{500}	ok. 1670

Szybkie potęgowanie

Podnoszenie do potęgi jest bardzo prostym, szkolnym zadaniem. Na przykład, aby obliczyć 3^4, wykonujemy trzy mnożenia 3*3*3*3. A zatem w ogólności, aby obliczyć wartość potęgi x^n tym sposobem należy wykonać $n - 1$ mnożeń, o jedno mniej niż wynosi wykładnik potęgi – ten algorytm będziemy nazywali **algorytmem szkolnym**. Czy ten algorytm jest na tyle szybki, by obliczyć na przykład wartość następującej potęgi:

$$x^{12345678912345678912345678912345}$$

która może pojawić przy szyfrowaniu metodą RSA (patrz p. 4.1.2) informacji przesyłanych w Internecie?

Spróbujmy obliczyć, ile czasu będzie trwało obliczanie powyższej potęgi stosując szkolny algorytm, czyli ile czasu zabierze wykonanie 123456789123456789123456789123 44 mnożeń. Przypuśćmy, że dysponujemy superkomputerem, czyli obecnie najszybszym komputerem. Taki komputer w 2011 roku działał z szybkością ok. 1 PFlops, czyli wykonywał 10^{15} operacji na sekundę. A zatem, obliczenie powyższej potęgi będzie trwało:

12345678912345678,912345678912344 sekund;
12345678912345678,912345678912344/60 = 205761315205761,31520576131520567 minut;
205761315205761,31520576131520567/60 = 3429355253429,3552534293552534278 godzin;
3429355253429,3552534293552534278/24 = 142889802226,22313555955646889282 dób;
142889802226,22313555955646889282/365 = 391478910,2088305083823464901 1733 lat,

czyli około $4*10^8$ lat. Jeśli taki algorytm byłby stosowany do szyfrowania naszej poczty w Internecie, to nigdy nie otrzymalibyśmy żadnego listu. Tutaj trzeba dodać, że w praktycznych sytuacjach muszą być obliczane potęgi o wykładnikach, które mają kilkaset cyfr.

Istnieje wiele algorytmów, które służą do szybkiego obliczania wartości potęgi. Większość z nich korzysta, bezpośrednio lub pośrednio, z binarnej reprezentacji wykładnika. Podstawowe algorytmy szybkiego potęgowania przedstawiono w książce [5]. Tutaj krótko opiszemy algorytm wykorzystujący rekurencję.

Zauważmy, że jeśli n jest liczbą parzystą, to zamiast obliczać wartość potęgi x^n, wystarczy obliczyć $y = x^{n/2}$, a następnie podnieść y do kwadratu. Jeśli n jest liczbą nieparzystą, to $n - 1$ jest liczbą parzystą. A zatem mamy następującą zależność:

$$x^n = \begin{cases} 1 & \text{jeśli } n = 0, \\ (x^{n/2})^2 & \text{jeśli } n \text{ jest liczbą parzystą}, \\ (x^{n-1})x & \text{jeśli } n \text{ jest liczbą nieparzystą}, \end{cases}$$

która ma charakter **rekurencyjny** – po prawej stronie równości są odwołania do potęgowania, czyli do tej samej operacji, której wartości liczymy, ale dla mniejszych wykładników. Pierwszy wiersz w powyższej równości to tzw. **warunek początkowy** – służy do zakończenia odwołań rekurencyjnych dla coraz mniejszych wykładników, aby cały proces obliczeń zakończył się. Ta zależność ma prostą realizację w języku programowania:

```
function potega(n:integer):real;
begin
  if n = 0 then potega:=1
  else if n mod 2 = 0 then potega:=potega(n div 2)^2
       else potega:=potega(n-1)*x
```

Aby określić, ile mnożeń jest wykonywanych w tym algorytmie należy zauważyć, że kolejne wywołania rekurencyjne odpowiadają kolejnym od końca bitom w binarnym rozwinięciu wykładnika i podnosimy do kwadratu tyle razy, ile jest pozycji w reprezentacji binarnej oraz dodatkowo mnożymy przez x, gdy w rozwinięciu pojawia się bit 1. Liczba bitów w binarnej reprezentacji liczby n wynosi około $\log_2 n$ i bit równy 1 może pojawić się na każdej pozycji, a zatem w sumie jest wykonywanych nie więcej niż $2 \log_2 n$ mnożeń. Dla naszej potęgi mamy więc:

$2 \log_2 n = 2 \log_2 123456789123456789123456789123 45 = 2*103,28 = 206,56,$

a zatem obliczenie przykładowej wartości potęgi, której wykładnik ma 32 cyfry wymaga wykonania nie więcej niż 206 mnożeń, co nawet na zwykłym komputerze osobistym potrwa niezauważalny ułamek sekundy. Z tabeli 1 wynika, że obliczenie wartości potęgi dla wykładnika o stu cyfrach wymaga wykonania nie więcej niż 670 mnożeń, a dla wykładnika o 500 cyfrach – nie więcej niż 3400 mnożeń, co także potrwa ułamek sekundy na komputerze osobistym. To zadanie szybkiego potęgowania jest znakomitą ilustracją wcześniej cytowanych słów Ralfa Gomory'ego, że najlepszym sposobem przyspieszania obliczeń komputerowych jest obarczanie komputera mniejszą liczba działań, czyli prawdziwe przyspieszanie obliczeń osiągamy dzięki efektywnym algorytmom, a nie szybszym komputerom.

Przedstawiona w rekurencyjnym algorytmie potęgowania technika algorytmiczna nosi nazwę **dziel i zwyciężaj**. Większość algorytmów opartych na tej technice jest bardzo efektywnych i ich złożoność na ogół wyraża się za pomocą funkcji logarytmicznej. Okazuje się, że algorytm Euklidesa, odkryty 1500 lat przed wprowadzeniem logarytmów, także bazuje na tej technice. Euklides był więc bardzo bliski wynalezienia logarytmu.

4.1.2 SZYFROWANIE

Człowiek szyfrował, czyli utajniał treści przesyłanych wiadomości, od kiedy zaczął je przekazywać innym osobom. Największym polem dla szyfrowania były zawsze wiadomości mające związek z obronnością i bezpieczeństwem, a także z prowadzonymi działaniami bojowymi.

Terminem **kryptografia** określa się utajnianie znaczenia wiadomości. **Szyfr**, to ustalony sposób utajniania (czyli **szyfrowania**) znaczenia wiadomości. Z kolei, łamaniem szyfrów, czyli odczytywaniem utajnionych wiadomości, zajmuje się **kryptoanaliza**. Kryptografia i kryptoanaliza to dwa główne działy **kryptologii**, nauki o utajnionej komunikacji.

Wielokrotnie w historii ludzkości szyfrowanie miało istotny wpływ na bieg wydarzeń. Najbardziej spektakularnym przykładem jest chyba historia rozpracowania niemieckiej maszyny szyfrującej Enigma, dzięki czemu – jak utrzymują historycy – II wojna światowa trwała 2-3 lata krócej. Dużą w tym rolę odegrali polscy matematycy: Marian Rejewski, Jerzy Różycki i Henryk Zygalski.

Obecnie, wraz z ekspansją komputerów i Internetu, coraz powszechniej dane i informacje są przechowywane i przekazywane w postaci elektronicznej. By nie miały do nich dostępu nieodpowiednie osoby, szyfrowane są zarówno dane przechowywane w komputerach, jak i tym bardziej dane i informacje przekazywane drogą elektroniczną, np. rozmowy telefoniczne czy operacje bankowe wykonywane z automatów bankowych. Szyfrowanie danych i wiadomości jest więc niezbędnym elementem dobrze zabezpieczonych systemów komputerowych.

Pojawianie się coraz silniejszych komputerów powoduje realne zagrożenie dla przesyłania utajnionych wiadomości. Kryptoanalityk może skorzystać z mocy komputera, by prowadzić analizę kryptogramów metodą prób i błędów. Co więcej, z ekspansją komunikacji najpierw telefonicznej, a obecnie – internetowej wzrosła do olbrzymich rozmiarów liczba przesyłanych wiadomości. Państwa, instytucje, a także pojedynczy obywatele chcieliby mieć gwarancję, że system wymiany wiadomości może zapewnić im bezpieczeństwo i prywatność komunikacji.

Szyfrowanie wiadomości polega na zastosowaniu odpowiedniego algorytmu szyfrowania. Algorytmy szyfrowania są powszechnie znane, a o ukryciu wiadomości decyduje **klucz szyfrowania**, niezbędny do uruchomienia algorytmu szyfrowania i deszyfrowania. Wymienianie się kluczami szyfrowania między nadawcą i odbiorcą wiadomości zawsze stanowiło najsłabszy punkt procedury szyfrowania, klucz może zostać przechwycony. Na przykład Marian Rejewski osiągnął pierwsze sukcesy przy deszyfracji Enigmy, korzystając z faktu, że klucz do zaszyfrowanej wiadomości był powtarzany na początku wiadomości.

W połowie lat siedemdziesiątych pojawiła się sugestia, że wymiana klucza między komunikującymi się stronami być może nie jest konieczna. Tak zrodził się pomysł **szyfru z kluczem publicznym**. Działanie odbiorcy i nadawcy utajnionych wiadomości w przypadku szyfrowania z kluczem publicznym jest następujące:

1. Odbiorca wiadomości tworzy parę kluczy: publiczny i prywatny i ujawnia klucz publiczny, np. zamieszcza go na swojej stronie internetowej.
2. Ktokolwiek chce wysłać zaszyfrowaną wiadomość do odbiorcy szyfruje ją jego kluczem publicznym, zaś tak utworzony kryptogram może odczytać jedynie odbiorca posługując się swoim kluczem prywatnym.

Pierwszy szyfr z kluczem publicznym opracowali w 1977 roku Ronald **R**ivest, Adi **S**hamir i Leonard **A**dleman z MIT i od inicjałów ich nazwisk pochodzi jego nazwa – **szyfr RSA**. Ten szyfr został wykorzystany w komputerowej realizacji szyfrowania z kluczem publicznym, zwanej **szyfrem PGP** (ang. *Pretty Good Privacy*), który jest powszechnie stosowany w Internecie. Na przykład uczestnicy zawodów Olimpiady Informatycznej wysyłają rozwiązania zadań szyfrowane kluczem publicznym dostępnym na stronie olimpiady, a jury olimpiady je rozszyfrowuje stosując sobie tylko znany klucz prywatny.

4.1.3 KOMPRESJA

Możliwość zapisania w pamięci komputera całej książki spowodowała chęć zapisania całych bibliotek. Możliwość pokazania na ekranie monitora dobrej jakości obrazu rozwinęła się do rozmiarów całego filmu. Można odnieść wrażenie, że korzystanie z pamięci rządzi się pewną odmianą prawa Parkinsona odnoszącą się do pamięci komputerów: *Dane zajmują zwykle całą pamięć możliwą do wypełnienia*.

Alternatywą dla powiększenia pamięci jest **kompresja danych**, czyli minimalizowanie ich objętości przez reprezentowanie w zwięzłej postaci. Gwałtowny rozwój metod i form komunikowania się nie byłby możliwy bez ciągłego ulepszania metod kompresji danych. Odnosi się to zarówno do tradycyjnych form wymiany informacji, takich jak: faks, modem czy telefonia komórkowa, jak i do wymiany informacji za pośrednictwem sieci Internet, w tym zwłaszcza do wymiany informacji multimedialnych. Proces kompresji i dekompresji danych jest często automatycznie (bez wiedzy użytkownika) wykonywany przez komputer. Użytkownik zauważa jedynie, że jego dysk może więcej pomieścić lub pewne operacje transferu danych są wykonywane szybciej. Zazwyczaj czas zużyty na kompresję lub dekompresję danych jest premiowany zwiększoną szybkością transferu skompresowanych danych.

Kompresja danych jest możliwa dzięki wykorzystaniu pewnych własności danych, na przykład często powtarzające się fragmenty można zastępować umownym symbolem lub im częściej jakiś fragment występuje tym mniejsza (krótsza) powinna być jego reprezentacja. Kompresja polega na zastosowaniu **algorytmu kompresji**, który tworzy reprezentację danych, wymagającą mniejszej liczby bitów. Każdemu algorytmowi kompresji odpowiada **algorytm dekompresji** (**rekonstrukcji**), który służy do zamiany skompresowanej reprezentacji danych na dane oryginalne. Tę parę algorytmów zwykło się nazywać algorytmem kompresji.

Historia kompresji sięga wielu lat przed erą komputerów. Ideę oszczędnego reprezentowania informacji odnajdujemy w połowie XIX wieku, gdy **Samuel Morse** (1791-1872) wynalazł **telegraf**, mechaniczne urządzenie do przesyłania wiadomości i posłużył się przy tym specjalnym alfabetem, znanym jako **alfabet Morse'a**, który umożliwia kodowanie znaków w tekście za pomocą dwóch symboli – kropki i kreski. Najważniejszą cechą tego alfabetu jest kodowanie najczęściej występujących znaków w tekście za pomocą możliwie najkrótszych kodów, np. kodem litery E jest kropka, a kodem litery T jest kreska, gdyż są to dwie najczęściej występujące litery w tekstach w języku angielskim. Ponieważ w telegrafie wysyłanie tekstu polega na przekazaniu kluczem kodów kolejnych znaków z tekstu, alfabet Morse'a znacznie zmniejszał liczbę znaków (kropek i kresek) potrzebnych do wysłania wiadomości.

Wadą alfabetu Morse'a jest to, że kody niektórych liter są częścią kodów innych liter, np. każdy kod zaczynający się od kropki zawiera na początku kod litery E. To powoduje, że w tekstach w kodzie Morse'a potrzebny jest dodatkowy znak oddzielający kody kolejnych liter. Tej wady nie ma **kod Huffmana**, zaproponowany w 1952 roku przez **Davida Huffmana** w jego pracy magisterskiej. W tym kodzie również często występujące znaki mają krótkie kody, ale żadan kod nie jest początkiem innego kodu. Kodowanie w tym kodzie nie wymaga więc dodatkowego znaku oddzielającego litery.

Na przykład słowo abrakadabra ma w kodzie Huffmana postać: 00101011001011001000010101100, czyli zamiast 88 bitów w kodzie ASCII wystarczy 29 bitów w kodzie Huffmana.

Algorytm Huffmana jest wykorzystywany w wielu profesjonalnych metodach kompresji tekstu, obrazów i dźwięków, również w połączeniu z innymi metodami. Redukcja wielkości danych przy stosowaniu tego algorytmu wynosi około 50% (w przypadku obrazów i dźwięków kodowane są nie same znaki, ale różnice między kolejnymi znakami).

4.1.4 POCZĄTKI KOMPUTERÓW ELEKTRONICZNYCH

Za jednego z ojców komputerów elektronicznych uważa się **Claude'a Shannona** (1916-2001), który w swojej pracy magisterskiej zaproponował realizację operacji algebry Boole'a w postaci układów przełączających. Jego praca jest uznawana za najważniejszą pracę dyplomową XX wieku.

Shannon był iście renesansowym człowiekiem. Twórca teorii informacji i propagator systemu binarnego do zapisywania obrazów i dźwięku, zafascynowany sztuczną inteligencją, zaprojektował pianino do odtwarzania zaprogramowanych utworów muzycznych oraz samouczącą się mysz, by znajdowała drogę w labiryncie. Shannon był także twórcą komputera szachowego MANIAC.

4.1.5 HISTORYCZNE PROCESORY TEKSTU

Niewiele osób jest świadomych, że dzisiejsze edytory tekstu i klawiatury do wprowadzania tekstów mają wiele elementów wspólnych z wcześniejszymi, mechanicznymi procesorami tekstu, czyli z maszynami do pisania. Faktycznie, popularna klawiatura komputerów, zwana często QWERTY, została po raz pierwszy użyta pod koniec XIX wieku w maszynach do pisania i od ponad pół wieku jest standardową klawiaturą do komunikacji z komputerem, a obecnie również z innymi urządzeniami elektronicznymi, takimi jak smartfony czy tablety. Z klawiaturą QWERTY nie jest kojarzony z nazwiska żaden wynalazca, można jedynie mówić o pierwszych firmach produkujących maszyny do pisania, które używały tych klawiatur: Sholes & Glidden, Remington, Underwood i inne.

Pomysł pisania tekstów różnymi czcionkami pojawił się również na początku rozwoju maszyn do pisania. Realizowano go w postaci wymiennych głowic z różnymi czcionkami, które łatwo wymieniało się podczas pisania. Ten wynalazek również nie ma swojego autora.

Na rysunku 2 jest pokazana maszyna Blickensderfer wyprodukowana w latach 90. XIX wieku w Stanach Zjednoczonych. Widoczne są wymienne głowice, przechowywane w drewnianych pudełkach. W tych maszynach stosowano odmienny typ klawiatury, którą można nazwać klawiaturą Morse'a, gdyż klawisze z najczęściej występującymi literami w tekście znajdują się blisko klawisza spacji (odstępu).

Rysunek 2.
Maszyna do pisania Blickensderfer, z wymiennymi głowicami z czcionkami oraz klawiaturą Morse'a (zdjęcie eksponatu z kolekcji maszyn autora)

4.2 KARIERY Z KLASĄ I KASĄ

W tym rozdziale krótko wspominamy o karierach informatycznych, kojarzonych obecnie przede wszystkim z olbrzymimi dochodami finansowymi. Chcielibyśmy jednak podkreślić, że w każdym przypadku te kariery są związane z wynalazkami i innowacjami w informatyce, które przyczyniły się do znaczącego rozwoju zastosowań informatyki na szeroką (globalną) skalę. Wymieniamy ponownie osoby z listy najbogatszych informatyków i przypisujemy im zasługi w rozwoju narzędzi i zastosowań informatyki.

1. Bill Gates, Steve Ballmer i Paul Allen, wszyscy z Microsoft. Microsoft jest obecnie dostawcą najpopularniejszych rozwiązań w zakresie systemów operacyjnych i sieciowych oraz pakietów użytkowych (biurowych) przeznaczonych dla komputerów osobistych typu IBM PC.
2. Larry Ellison, Oracle. Firma dostarczająca najpopularniejszy system zarządzania bazami danych.
3. Jeff Bezos stworzył firmę internetową Amazon, która na początku zajmowała się sprzedażą książek, nowych i używanych, a później także innych towarów. Była pierwszą księgarnią internetową, która oferowała e-książki dla e-czytników Kindle, które także sprzedawała. Okazało się to strzałem w dziesiątkę, gdyż ludzie zaczęli kupować więcej e-książek niż książek papierowych.
4. Mark Zuckerberger to twórca najpopularniejszego serwisu społecznościowego Facebook, ten serwis dostarcza coraz to nowych usług służących do komunikacji w różnych grupach jego użytkowników.
5. Sergey Brin i Larry Page stworzyli wyszukiwarkę Google, a ich celem jest skatalogowanie wszystkich zasobów informacji (również tych papierowych po digitalizacji) i udostępnienie ich wszystkim użytkownikom sieci. W wyszukiwarce Google zrealizowano nowatorskie algorytmy indeksowania i wyszukiwania informacji, które są przykładem implementacji algorytmów kombinatorycznych na olbrzymią skalę.
6. Michael Dell to jeden z największych producentów komputerów osobistych marki Dell.
7. Steve Jobs i Steve Wozniak twórcy firmy Apple w 1983 roku, producenci pierwszych komputerów osobistych. Dzisiaj Steve Jobs był jednym z czołowych innowatorów w dziedzinie powszechnych zastosowań technologii komputerowej i informacyjno-komunikacyjnej. Do flagowych produktów firmy Apple należą ostatnio: iPod, iPhone i iPad. Produkty tej firmy charakteryzują się zarówno wysoką jakością rozwiązań technologicznych, niezmiernie przyjaznym interfejsem, jak i nienagannym projektem użytkowym. Wszystkie elementy produktów firmy Apple, techniczne i programistyczne, w tym również sieciowe, wychodzą „spod jednej igły" głównego pomysłodawcy. Firma Apple jest niedoścignionym wzorem dla innych firm.

Obok księgarni internetowej Amazon istnieje wiele internetowych serwisów aukcyjnych, takich jak eBay i Allegro.

Najpopularniejszym serwisem społecznościowym w Polsce był przez pewien okres serwis Nasza klasa (obecnie nk), utworzony przez studentów Instytutu Informatyki Uniwersytetu Wrocławskiego. Jego celem było, jak podają twórcy, „umożliwienie użytkownikom odnalezienie osób ze swoich szkolnych lat i odnowienie z nimi kontaktu". Obecnie jest to jeden z serwisów społecznościowych. Główny pomysłodawca tego serwisu, Maciej Popowicz, uczynił ten serwis przedmiotem swojej pracy magisterskiej.

Dużą popularnością cieszy się serwis społecznościowy Twitter, z możliwością mikroblogowania, polegającego na wysyłaniu i odczytywaniu krótkich wiadomości, tzw. *twittów*.

Warto wspomnieć jeszcze o telefonii internetowej Skype, dzięki której jest możliwa darmowa komunikacja *on-line* między dowolnymi komputerami podłączonymi do sieci.

Jak napisaliśmy wcześniej, informatyczne kariery „z kasą" są rezultatem realizacji wynalazków, pomysłów i innowacyjnych rozwiązań, konsekwentnego ich wdrażania i rozwijania. Droga do dobrobytu twórców tych rozwiązań wiedzie przez zrozumienie oraz twórcze i innowacyjne działanie w obszarze rozwiązań informatycznych, które mają swoje społeczne oddziaływanie na dużą skalę.

5 WYZWANIA

W tym rozdziale przedstawiamy kilka problemów i wyzwań, które stoją przed informatyką i informatykami. Wyznaczają one kierunek innowacyjnych działań na polu informatyki, zarówno w zakresie rozwiązań teoretycznych, jak i praktycznych. Wiele z tych problemów to trudne wyzwania, które dotychczas opierały się wszelkim próbom rozwiązania. To nie znaczy jednak, że te nie istnieją. Być może jest potrzebne odnowione spojrzenie, nowatorska metoda, całkiem nietypowe podejście. Historia pokazała – niektóre takie przypadki prezentujemy wcześniej – że droga do odkryć i innowacji bywa na ogół bardzo nietypowa i mogą na nią wejść osoby nieobarczone olbrzymim zasobem wiedzy, wręcz nowicjusze. A zatem

unlock you potential!

odblokuj swoje możliwości.

5.1 WSPÓŁPRACA W SIECI

Dla wielu nierozwiązanych, trudnych lub bardzo złożonych problemów obliczeniowych istnieją w sieci serwisy, których celem jest utworzenie społeczności sieciowych zajmujących się rozwiązywaniem takich problemów. Wymieńmy najpopularniejsze z nich.

1. Szukanie dużych liczb pierwszych – liczb Mersenne'a – http://www.mersenne.org/ – na znalazcę kolejnej dużej liczby pierwszej czeka nagroda 50 000/100 000 $. Ostatnie duże liczby pierwsze znajdowali studenci na swoich komputerach osobistych. Na podanej stronie można zapoznać się z projektem (istnieje wersja po polsku) i pobrać oprogramowanie do swojego komputera, które w wolnym czasie będzie wykonywać postawione mu przez serwis poszukiwanie.

   Obecnie największa liczba pierwsza jest równa $2^{43112609} - 1$. Liczba ta ma 12 978 189 cyfr. Zapisanie jej w edytorze teksu (75 cyfr w wierszu, 50 wierszy na stronie) zajęłoby 3461 stron.

2. Folding@Home – badanie proces zwijania białek – projekt prowadzony przez Uniwersytet Stanforda w Stanach Zjednoczonych. Projekt ma na celu zbadanie mechanizmów, które powodują choroby Alzheimera, Parkinsona i BSE (szalonych krów). Jest to największy projekt obliczeń rozproszonych (czyli przebiegających w różnych komputerach, koordynowanych przez serwer projektu), w którym uczestniczą posiadacze Sony Play Station 3 i komputerów osobistych. Moc komputerów uczestniczących w projekcie przekroczyła pięć razy moc najpotężniejszego superkomputera. Zainteresowanych odsyłamy do strony projektu w wersji polskiej (http://wiki.zwijaj.pl), gdzie można pobrać oprogramowanie na swój komputer i włączyć się do projektu.

   W sieci istnieje wiele innych projektów obliczeniowych, polegających na rozproszeniu obliczeń na wiele niezależnych komputerów. W ten sposób np. NASA bada sygnały nadchodzące z kosmosu.

   Innym projektem rozproszonym jest... Wikipedia – tworzenie encyklopedii internetowej.

   Jeszcze inne projekty są związane z rozwojem wolnego oprogramowania, takiego jak systemy Linux, Moodle i inne.

   Wszystkie wyżej opisane projekty są dostępne dla każdego użytkownika sieci. Zachęcamy do uczestnictwa w społecznościach, które realizują te projekty. Uczestnictwo nie polega tylko na udostępnieniu swojego komputera – jest jednocześnie okazją do aktywnego włączenia się w życie społeczności internetowych zajmujących się wybranym obszarem badań i działań.

5.2 KILKA TRUDNYCH PROBLEMÓW

Podamy tutaj kilka dość prostych problemów, dla których znalezienie rozwiązania nastręcza trudności nawet z użyciem najszybszych komputerów. Te problemy „czekają" na lepsze metody i algorytmy rozwiązywania.

5.2.1 NAJKRÓTSZA TRASA ZAMKNIĘTA

Jednym z najbardziej znanych problemów dotyczących wyznaczania tras przejazdu, jest **problem komiwojażera**, oznaczany zwykle jako **TSP**, od oryginalnej nazwy *Travelling Salesman Problem*. W tym problemie mamy dany zbiór miejscowości oraz odległości między nimi. Należy znaleźć drogę zamkniętą, przechodzącą przez każdą miejscowość dokładnie jeden raz, która ma najkrótszą długość.

Przykładem zastosowania problemu TSP może być zadanie wyznaczenia najkrótszej trasy objazdu kraju przez prezydenta lub premiera po wszystkich stolicach województw (stanów – w Stanach Zjednoczonych, landów – w Niemczech itp.). Prezydent wyjeżdża ze stolicy kraju, ma odwiedzić stolicę każdego województwa dokładnie jeden raz i wrócić do miasta, z którego wyruszył.

Na rysunku 3 jest przedstawiona jedna z możliwych tras, nie ma jednak pewności, czy jest ona najkrótsza. Obsługa biura prezydenta może jednak chcieć znaleźć najkrótszą trasę. W tym celu postanowiono generować wszystkie możliwe trasy – zastanówmy się, ile ich jest. To łatwo policzyć. Z Warszawy można się udać do jednego z 15 miast wojewódzkich. Będąc w pierwszym wybranym mieście, do wyboru mamy jedno z 14 miast. Po wybraniu drugiego miasta na trasie, kolejne miasto można wybrać spośród 13 miast i tak dalej. Gdy zaliczamy ostatnie miasto, to czeka nas tylko powrót do Warszawy. A zatem wszystkich możliwych wyborów jest: 15*14*13*...*2*1. Oznaczmy tę liczbę następująco:

$$15! = 15*14*13*...*2*1$$

a ogólnie

$$n! = n*(n-1)*(n-2)*...*2*1$$

Rysunek 3.
Przykładowa trasa przejazdu prezydenta po stolicach województw

Oznaczenie $n!$ czytamy **„n silnia"**, a zatem $n!$ jest iloczynem kolejnych liczb całkowitych, od jeden do n. Wartości tej funkcji dla kolejnych n rosną bardzo szybko (patrz tabela 2).

Tabela 2.
Wartości funkcji *n*!

n	*n*!
10	3628800
15	$1{,}30767*10^{12}$
20	$2{,}4329*10^{18}$
25	$1{,}55112*10^{25}$
30	$2{,}65253*10^{32}$
40	$8{,}15915*10^{47}$
48	$1{,}24139*10^{61}$
100	$9{,}3326*10^{157}$

Z wartości umieszczonych w tabeli 2 wynika, że posługując się superkomputerem w realizacji naszkicowanej metody, służącej do znalezienia najkrótszej trasy dla prezydenta Polski, otrzymanie takiej trasy zabrałoby mniej niż sekundę. Jednak w olbrzymim kłopocie znajdzie się prezydent Stanów Zjednoczonych chcąc taką samą metodą znaleźć najkrótszą trasę objazdu po stolicach wszystkich kontynentalnych stanów (jest ich 49, z wyjątkiem Hawajów). Niewiele zmieni jego sytuację przyspieszenie superkomputerów.

Znane są metody rozwiązywania problemu komiwojażera szybsze niż naszkicowana powyżej, jednak problem TSP pozostaje bardzo trudny. W takich przypadkach często są stosowane metody, które służą do szybkiego znajdowania rozwiązań przybliżonych, niekoniecznie najkrótszych. Jedna z takich metod, zwana **metodą najbliższego sąsiada**, polega na przejeżdżaniu w każdym kroku do miasta, które znajduje się najbliżej miasta, w którym się znajdujemy. W rozwiązaniu naszego problemu tą metodą, pierwszym odwiedzonym miastem powinna być Łódź, później Kielce, Lublin, Rzeszów..., a nie jak na rysunku 3 mamy Lublin, Rzeszów, Kraków ..., a Łódź gdzieś w trakcie podróży. Trasa otrzymana metodą najbliższego sąsiada jest krótsza niż trasa naszkicowana na rysunku 3. W ogólnym przypadku ta metoda nie gwarantuje, że zawsze znajdzie najkrótszą trasę.

5.2.2 ROZKŁAD LICZBY NA CZYNNIKI PIERWSZE

Liczby pierwsze stanowią w pewnym sensie „pierwiastki" wszystkich liczb, każdą bowiem liczbę całkowitą można jednoznacznie przedstawić, z dokładnością do kolejności, w postaci iloczynu liczb pierwszych. Na przykład, 4 = 2*2; 10 = 2*5; 20 = 2*2*5 = 2*5*2 = 5*2*2; 23 = 23; dla liczb pierwszych te iloczyny składają się z jednej liczby.

Matematycy interesowali się liczbami pierwszymi od dawna. Pierwsze spisane rozważania i twierdzenia dotyczące tych liczb znajdujemy w działach Euklidesa. Obecnie liczby pierwsze znajdują ważne zastosowania w kryptografii, m.in. w algorytmach szyfrujących. Do najważniejszych pytań, problemów i wyzwań, związanych z liczbami pierwszymi, należą następujące zagadnienia, które krótko komentujemy:

1. Dana jest dodatnia liczba całkowita *n* – czy *n* jest liczbą pierwszą (złożoną)?

   Ten problem ma bardzo duże znaczenie zarówno praktyczne (w kryptografii), jak i teoretyczne. Dopiero w 2002 roku został podany algorytm, który jednak jest interesujący głównie z teoretycznego punktu widzenia. Jego złożoność, czyli liczba wykonywanych operacji, zależy wielomianowo od rozmiaru liczby *n*, czyli od liczby bitów potrzebnych do zapisania liczby *n* w komputerze (równej $\log_2 n$). Słabą stroną większości metod, które udzielają odpowiedzi na pytanie: „czy *n* jest liczba pierwszą czy złożoną" jest udzielanie jedynie odpowiedzi „Tak" lub „Nie". Na ogół najszybsze metody dające odpowiedź na pytanie, czy *n* jest liczbą złożoną, w przypadku odpowiedzi „Tak" nie podają dzielników liczby *n* – dzielniki jest znacznie trudniej znaleźć niż przekonać się, że liczba ma dzielniki.

2. Dana jest dodatnia liczba całkowita *n* – rozłóż *n* na czynniki pierwsze.

   Ten problem ma olbrzymie znaczenie w kryptografii. Odpowiedź „Nie" udzielona na pytanie nr 1 nie pomaga w jego rozwiązaniu. Dysponujemy jednak prostym algorytmem, który polega na dzieleniu *n* przez kolejne liczby naturalne i wystarczy dzielić tylko przez liczby nie większe niż √n, gdyż liczby *n* nie można rozłożyć na iloczyn dwóch liczb większych od √n. Algorytm ten ma bardzo prostą postać:

```
var i,n:integer;
i:=2;
while i*i <= n do begin
   if n mod i = 0 then return 1;  {n jest podzielne przez i}
   i=i+1
end;
return 0    {n jest liczba pierwszą}
```

   Ten fragment programu zwraca 0, jeśli *n* jest liczbą pierwszą, i 1, gdy *n* jest liczbą złożoną. W tym drugim przypadku znamy także podzielnik liczby *n*.

   Liczba operacji wykonywanych przez powyższy program jest w najgorszym przypadku (gdy *n* jest liczbą pierwszą) proporcjonalna do √n, a więc jeśli $n = 10^m$, to wykonywanych jest $10^{m/2}$ operacji. Zatem są niewielkie szanse, by tym algorytmem rozłożyć na czynniki pierwsze liczbę, która ma kilkaset cyfr, lub stwierdzić, że się jej nie da rozłożyć.

   Rozkładem liczby złożonej na czynniki pierwsze mogą być zainteresowani ci, którzy starają się złamać szyfr RSA. Wiadomo, że jedna z liczb tworzących klucz publiczny i prywatny jest iloczynem dwóch liczb pierwszych. Znajomość tych dwóch czynników umożliwia utworzenie klucza prywatnego (tajnego). Jednak ich wielkość – są to liczby pierwsze o kilkuset cyfrach (200-300) – stoi na przeszkodzie w rozkładzie *n*. Żaden z istniejących algorytmów, przy obecnej mocy komputerów, nie umożliwia rozkładania na czynniki pierwsze liczb, które mają kilkaset cyfr. Szyfr RSA pozostaje więc nadal bezpiecznym sposobem utajniania wiadomości, w tym również przesyłanych w sieciach komputerowych. Co więcej, wzrost mocy komputerów w najbliższej przyszłości nie jest w stanie tego zmienić.

3. Dana jest dodatnia liczba całkowita *m* – znajdź wszystkie liczby pierwsze mniejsze lub równe *m*.

   To zadanie zyskało swoją popularność dzięki algorytmowi pochodzącemu ze starożytności, który jest znany jako **sito Eratosthenesa**. Generowanie kolejnych liczb pierwszych tą metodą ma jednak niewielkie znaczenie praktyczne i jest uznawane raczej za ciekawostkę.

4. Znajdź największą liczbę pierwszą, a faktycznie, znajdź liczbę pierwszą większą od największej znanej liczby pierwszej. (Zgodnie z twierdzeniem Euklidesa, liczb pierwszych jest nieskończenie wiele, a zatem nie istnieje największa liczba pierwsza.). O tym problemie piszemy w p. 5.1.

5.3 PRAWDZIWE WYZWANIE

Z dotychczasowych rozważań wynika, że dla problemów, które chcemy rozwiązać za pomocą komputera, albo istnieje algorytm dość szybki, albo takiego algorytmu nie ma. Co to znaczy szybki algorytm? Przyjęło się w informatyce, że algorytm jest **szybki** (a problem jest **łatwy**), jeśli liczba wykonywanych działań jest ograniczona

przez wielomian od liczby danych. Na przykład porządkowanie jest problemem łatwym, bo n liczb można szybko uporządkować za pomocą co najwyżej n^2 działań lub nawet $n \log_2 n$ działań. Szybkie są też: algorytm Euklidesa, schemat Hornera i podnoszenie do potęgi. Problemy łatwe zaliczamy do klasy oznaczanej literą **P**.

Z kolei do klasy **NP** zaliczamy problemy **trudne**, czyli takie, które mają wielomianowy algorytm sprawdzania, czy rozwiązanie jest poprawne, ale nie mają wielomianowego algorytmu rozwiązywania. Do tej klasy należy na przykład tzw. decyzyjna wersja problemu komiwojażera, czyli pytanie, czy dla ustalonej liczby k istnieje droga komiwojażera o długości nie większej niż k.

Jednym z otwartych problemów jest pytanie, czy **P = NP**, czyli czy te dwie klasy problemów składają się z tych samych problemów. Równość jest mało prawdopodobna, bo po blisko 40 latach badań, dla żadnego z problemów należących do klasy **NP** nie udało się podać wielomianowego algorytmu rozwiązywania. Pozytywnym skutkiem braku równości tych klas jest to, że system kryptograficzny RSA jest bezpieczny, bo nie istnieje łatwy algorytm znajdowania klucza prywatnego dysponując kluczem publicznym.

Czy **P = NP**, jest jednym z siedmiu problemów matematycznych, tak zwanych Problemów Milenijnych. Instytut Matematyczny Claya w MIT w Stanach Zjednoczonych ogłosił siedem problemów za fundamentalne dla rozwoju matematyki i za podanie rozwiązania któregokolwiek z tych problemów ustanowiono nagrodę w wysokości miliona dolarów. Jeden z tych problemów został już rozwiązany trzy lata temu. Ponad rok temu pojawił się dowód, że **P** nie jest równe **NP**, ale nie został on pozytywnie zweryfikowany.

6 ALGORYTM, ALGORYTMIKA I ALGORYTMICZNE ROZWIĄZYWANIE PROBLEMÓW

Ten rozdział jest krótkim wprowadzeniem do zajęć w module „Algorytmika i programowanie”. Krótko wyjaśniamy w nim podstawowe pojęcia oraz stosowane na zajęciach podejście do rozwiązywania problemów z pomocą komputera.

Algorytm

Powszechnie przyjmuje się, że **algorytm** jest opisem krok po kroku rozwiązania postawionego problemu lub sposobu osiągnięcia jakiegoś celu. To pojęcie wywodzi się z matematyki i informatyki – za pierwszy algorytm uznaje się bowiem algorytm Euklidesa, podany ponad 2300 lat temu. W ostatnich latach algorytm stał się bardzo popularnym synonimem przepisu lub instrukcji postępowania.

W szkole, algorytm pojawia się po raz pierwszy na lekcjach matematyki już w szkole podstawowej, na przykład jako algorytm pisemnego dodawania dwóch liczb, wiele klas wcześniej, zanim staje się przedmiotem zajęć informatycznych.

O znaczeniu algorytmów w informatyce może świadczyć następujące określenie, przyjmowane za definicję informatyki:

informatyka jest dziedziną wiedzy i działalności zajmującą się algorytmami.

W tej definicji informatyki nie ma dużej przesady, gdyż zawarte są w niej pośrednio inne pojęcia stosowane do definiowania informatyki: **komputery** – jako urządzenia wykonujące odpowiednio dla nich zapisane algorytmy (czyli niejako wprawiane w ruch algorytmami); **informacja** – jako materiał przetwarzany i produkowany przez komputery; **programowanie** – jako zespół metod i środków (np. języków i systemów użytkowych) do zapisywania algorytmów w postaci programów.

Położenie nacisku w poznawaniu informatyki na algorytmy jest jeszcze uzasadnione tym, że zarówno konstrukcje komputerów, jak i ich oprogramowanie bardzo szybko się starzeją, natomiast podstawy stosowania komputerów, które są przedmiotem zainteresowań algorytmiki, zmieniają się bardzo powoli, a niektóre z nich w ogóle nie ulegają zmianie.

Algorytmy, zwłaszcza w swoim popularnym znaczeniu, występują wszędzie wokół nas – niemal każdy ruch człowieka, zarówno angażujący jego mięśnie, jak i będący jedynie działaniem umysłu, jest wykonywany według jakiegoś przepisu postępowania, którego nie zawsze jesteśmy nawet świadomi. Wiele naszych czynności potrafimy wyabstrahować i podać w postaci precyzyjnego opisu, ale w bardzo wielu przypadkach nie potrafimy nawet powtórzyć, jak to się dzieje lub jak to się stało[3].

Nie wszystkie zachowania, nazywane algorytmami, są ściśle związane z komputerami i nie wszystkie przepisy działań można uznać za algorytmy w znaczeniu informatycznym. Na przykład nie są nimi na ogół przepisy kulinarne, chociaż odwołuje się do nich David Harel w swoim fundamentalnym dziele o algorytmach i algorytmice [3]. Otóż przepis np. na sporządzenie „ciągutki z wiśniami”, którą zachwycała się Alicja w Krainie Czarów, nie jest algorytmem, gdyż nie ma dwóch osób, które na jego podstawie, dysponując tymi samymi produktami, zrobiłyby taką samą, czyli jednakowo smakującą ciągutkę. Nie może być bowiem algorytmem przepis, który dla identycznych danych daje różne wyniki w dwóch różnych wykonaniach, jak to najczęściej bywa w przypadku robienia potraw według „algorytmów kulinarnych”.

Algorytmika

Algorytmika to dział informatyki, zajmujący się różnymi aspektami tworzenia i analizowania algorytmów, przede wszystkim w odniesieniu do ich roli jako precyzyjnego opisu postępowania, mającego na celu znalezienie rozwiązania postawionego problemu. Algorytm może być wykonywany przez człowieka, przez komputer lub w inny sposób, np. przez specjalnie dla niego zbudowane urządzenie. W ostatnich latach postęp w rozwoju komputerów i informatyki był nierozerwalnie związany z rozwojem coraz doskonalszych algorytmów.

Informatyka jest dziedziną zajmującą się rozwiązywaniem problemów z wykorzystaniem komputerów. O znaczeniu algorytmu w informatyce może świadczyć fakt, że każdy program komputerowy działa zgodnie z jakimś algorytmem, a więc zanim zadamy komputerowi nowe zadanie do wykonania powinniśmy umieć „wytłumaczyć” mu dokładnie, co ma robić.

Staramy się, by prezentowane algorytmy były jak najprostsze i by działały jak najszybciej. To ostatnie żądanie może wydawać się dziwne, przecież dysponujemy już teraz bardzo szybkimi komputerami i szybkość działania procesorów stale rośnie (według prawa Moore’a podwaja się co 18 miesięcy). Mimo to istnieją problemy, których obecnie nie jest w stanie rozwiązać żaden komputer i zwiększenie szybkości komputerów niewiele pomoże, kluczowe więc staje się opracowywanie coraz szybszych algorytmów.

Algorytmiczne rozwiązywanie problemów

Komputer jest stosowany do rozwiązywania problemów zarówno przez profesjonalnych informatyków, którzy projektują i tworzą oprogramowanie, jak i przez tych, którzy stosują tylko technologię informacyjno-komunikacyjną, czyli nie wykraczają poza posługiwanie się gotowymi narzędziami informatycznymi. W obu przypadkach ma zastosowanie podejście do **rozwiązywania problemów algorytmicznych**, które polega na systematycznej pracy nad komputerowym rozwiązaniem problemu i obejmuje cały proces projektowania i otrzymania rozwiązania. Celem nadrzędnym tej metodologii jest otrzymanie **dobrego rozwiązania**, czyli takiego, które jest:

[3] Interesująco ujął to Jay Nievergelt w artykule [5] – *Jest tak, jakby na przykład stonoga chciała wyjaśnić, w jakiej kolejności wprawia w ruch swoje nogi, ale z przerażeniem stwierdza, że nie może iść dalej.*

- **zrozumiałe dla każdego**, kto zna dziedzinę rozwiązywanego problemu i użyte narzędzia komputerowe,
- **poprawne**, czyli spełnia specyfikację problemu, a więc dokładny opis problemu,
- **efektywne**, czyli niepotrzebnie nie marnuje zasobów komputerowych, czasu i pamięci.

Ta metoda składa się z następujących sześciu etapów:

1. *Opis i analiza sytuacji problemowej*. Na podstawie opisu i analizy sytuacji problemowej należy w pełni zrozumieć, na czym polega problem, jakie są dane dla problemu i jakich oczekujemy wyników, oraz jakie są możliwe ograniczenia.
2. *Sporządzenie specyfikacji problemu*, czyli dokładnego opisu problemu na podstawie rezultatów etapu 1.
   Specyfikacja problemu zawiera:
   - opis danych,
   - opis wyników,
   - opis relacji (powiązań, zależności) między danymi i wynikami.

   Specyfikacja jest wykorzystana w następnym etapie jako specyfikacja tworzonego rozwiązania (np. programu).
3. *Zaprojektowanie rozwiązania*. Dla sporządzonej na poprzednim etapie specyfikacji problemu, jest projektowane rozwiązanie komputerowe (np. program), czyli wybierany odpowiedni algorytm i dobierane do niego struktury danych. Wybierane jest także środowisko komputerowe (np. język programowania), w którym będzie realizowane rozwiązanie na komputerze.
4. *Komputerowa realizacja rozwiązania*. Dla projektu rozwiązania, opracowanego na poprzednim etapie, buduje się kompletne rozwiązanie komputerowe, np. w postaci programu w wybranym języku programowania. Następnie testowana jest poprawność rozwiązania komputerowego i badana jego efektywność działania na różnych danych.
5. *Testowanie rozwiązania*. Ten etap jest poświęcony na systematyczną weryfikację poprawności rozwiązania i testowanie jego własności, w tym zgodności ze specyfikacją.
6. *Prezentacja rozwiązania*. Dla otrzymanego rozwiązania należy jeszcze opracować dokumentację i pomoc dla (innego) użytkownika. Cały proces rozwiązywania problemu kończy prezentacja innym zainteresowanym osobom (uczniom, nauczycielowi) sposobu otrzymania rozwiązania oraz samego rozwiązania wraz z dokumentacją.

Chociaż powyższa metodologia jest stosowana głównie do otrzymywania komputerowych rozwiązań, które mają postać programów napisanych w wybranym języku programowania, może być zastosowana również do otrzymywania rozwiązań komputerowych większości problemów z obszaru zastosowań informatyki i posługiwania się technologią informacyjno-komunikacyjną[4], czyli gotowym oprogramowaniem.

Dwie uwagi do powyższych rozważań:

Uwaga 1. Wszyscy, w mniejszym lub większym stopniu, zmagamy się z problemami, pochodzącymi z różnych dziedzin (przedmiotów). W naszych rozważaniach problem nie jest jednak wyzwaniem nie do pokonania, przyjmujemy bowiem, że **problem** jest sytuacją, w której uczeń ma przedstawić jej rozwiązanie bazując na tym, co wie, ale nie jest powiedziane, jak to ma zrobić. Problem na ogół zawiera pewną trudność, nie jest rutynowym zadaniem. Na takie sytuacje problemowe rozszerzamy pojęcie problemu, wymagającego przedstawienia rozwiązania komputerowego.

Uwaga 2. W tych rozważaniach rozszerzamy także pojęcie **programowania**. Jak powszechnie wiadomo, komputery wykonują tylko programy. Użytkownik komputera może korzystać z istniejących programów (np. z pakietu Office), a może także posługiwać się własnymi programami, napisanymi w języku programowania, który „rozumieją" komputery. W szkole nie ma zbyt wiele czasu, by uczyć programowania, uczniowie też nie są odpowiednio przygotowani do programowania komputerów. Istnieje jednak wiele sposobności, by kształcić zdolność komunikowania się z komputerem za pomocą programów, które powstają w inny sposób niż za pomocą programowania w wybranym języku programowania. Szczególnym przypadkiem takich programów jest oprogramowanie edukacyjne, które służy do wykonywania i śledzenia działania algorytmów. „Programowanie" w przypadku takiego oprogramowania polega na dobieraniu odpowiednich parametrów, które mają wpływ na działanie algorytmów i tym samym umożliwiają lepsze zapoznanie się z nimi.

[4] W najnowszej podstawie programowej dla przedmiotu informatyka, przyjętej do realizacji pod koniec 2008 roku, to podejście jest zalecane jako podstawowa metodologia rozwiązywania problemów z pomocą komputera.

LITERATURA

1. Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów*, WNT, Warszawa 1997
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO,* WSiP, Warszawa 2002-2003
3. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
4. Knuth D.E., *Sztuka programowania*, tomy 1–3, WNT, Warszawa 2003
5. Nievergelt J., *Co to jest dydaktyka informatyki?,* „Komputer w Edukacji" 1/1994
6. Steinhaus H., *Kalejdoskop matematyczny*, WSiP, Warszawa 1989
7. Sysło M.M., *Algorytmy,* WSiP, Warszawa 1997
8. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://mmsyslo.pl/Materialy/Ksiazki-i-podreczniki/Ksiazki/Ksiazka-Piramidy-szyszki-i
9. Wirth N., *Algorytmy + struktury danych = programy*, WNT, Warszawa 1980

Czy komputery będą robić biznes

Wojciech Cellary

Katedra Technologii Informacyjnych, Uniwersytet Ekonomiczny w Poznaniu

www.kti.ue.poznan.pl

Streszczenie

Bez komputerów nie da się robić biznesu. Komputery będą coraz powszechniej świadczyć elektroniczne usługi, ale robić będą mogły tylko to, do czego zaprogramują je ludzie. Styk informatyki i biznesu jest jednym z najbardziej fascynujących obszarów aktywności badawczej, w którym mamy do czynienia z największą liczbą innowacji. W dodatku, do zrobienia kariery w tym obszarze, nawet od zera, jest potrzebny przede wszystkim pomysł na biznes – jeśli spotka się z uznaniem na rynku, to w bardzo krótkim czasie kilkuosobowa firma ze znikomym kapitałem założycielskim może przemienić się w globalnego gracza.

Nową koncepcją w sferze zarządzania przedsiębiorstwami są **sieciowe organizacje wirtualne** (SOW). Wirtualne, czyli pozorne – organizacje, które w oczach swoich klientów zachowują się tak, jak każde przedsiębiorstwo, ale naprawdę są pewną strukturą organizacyjną obejmującą wiele niezależnych, wzajemnie uzupełniających się podmiotów gospodarczych i instytucji, z których każda pełni ściśle określoną rolę. Sieciowe organizacje wirtualne wymagają nowej organizacji przedsiębiorstw – rozbicia na małe, wyspecjalizowane jednostki, które można łatwo konfigurować dla osiągnięcia konkretnego, wyłaniającego się, większego celu biznesowego. Wymagają też nowej architektury informatycznej. Jest nią **architektura usługowa SOA** (ang. *Service Oriented Architecture*) oparta na usługach sieciowych. Dzięki architekturze usługowej, niezależne, zbudowane w różnych technologiach systemy informatyczne różnych przedsiębiorstw i instytucji mogą nawzajem świadczyć sobie coraz to nowe elektroniczne usługi. Dzięki temu można skutecznie zarządzać organizacją wirtualną nie naruszając autonomii przedsiębiorstw i instytucji wchodzących w ich skład.

Spis treści

1 WSTĘP

Informatyka w bardzo krótkim czasie od swego powstania, czyli przez około sześćdziesiąt lat, przeniknęła praktycznie wszystkie dziedziny życia i działalności ludzkiej, niektóre z nich bardzo głęboko zmieniając. To zaangażowanie informatyki nie ominęło oczywiście działalności gospodarczej, czyli „robienia biznesu". Robienie biznesu często utożsamia się z „robieniem pieniędzy", z bogactwem. Jest w tym tylko część prawdy. Robienie biznesu polega bowiem przede wszystkim na takim zaspokajaniu potrzeb ludzi, czyli albo na dostarczaniu im produktów, albo na świadczeniu im usług, aby byli skłonni zapłacić za nie więcej niż wynoszą koszty produkcji. Najważniejsze w biznesie jest pojęcie **użyteczności** dla ludzi – jeśli nie ma się nic użytecznego do zaoferowania klientom, to nie można zrobić biznesu. Nadwyżka wpływów ze sprzedaży nad kosztami, po zapłaceniu należnych podatków, stanowi zysk.

Zysk jest głównym narzędziem inwestowania w rozwój:

- wprowadzania na rynek nowych produktów i usług;
- poprawy jakości, efektywności i funkcjonalności przy obniżaniu kosztów produktów i usług obecnych na rynku;
- poprawy procesów wytwórczych, dostawczych i marketingowych.

Przedsiębiorstwu, które się nie rozwija, grozi wyprzedzenie przez konkurencję, a zatem w skrajnym przypadku zniknięcie z rynku. Rozwój jest jednak przede wszystkim potrzebny do podnoszenia jakości życia ludzi. Jest też niezbędny do tworzenia miejsc pracy dla młodych ludzi wchodzących na rynek. Dlatego troska o rozwój ekonomiczny jest tak bardzo powiązana z troską o przyszłość kraju.

Informatyka zrewolucjonizowała sposób zarządzania przedsiębiorstwami, oferując możliwość przechowywania w bazach danych w uporządkowany sposób dowolnie szczegółowej informacji o przedsiębiorstwie i każdym aspekcie jego działalności oraz możliwość automatycznego przetwarzania tej informacji dla celów zarządczych. Fundamentalną zmianą w dziedzinie zarządzania przedsiębiorstwami, jaka dokonała się dzięki informatyce, było odejście od zasady uniformizacji procesów biznesowych na rzecz masowej personalizacji. Na początku XX wieku obowiązywała maksyma Henry'ego Forda, że każdy może wybrać sobie dowolny kolor samochodu pod warunkiem, że jest to kolor czarny. Tak daleko posunięta uniformizacja miała na celu obniżenie kosztów produkcji. Dzięki informatyce i automatyzacji, ta zasada zdezaktualizowała się. Dzisiaj każde przedsiębiorstwo stara się w maksymalnym stopniu spełnić indywidualne oczekiwania klientów i dzięki komputerom wcale nie podnosi to nadmiernie kosztów produkcji. Lepsze spełnianie zróżnicowanych potrzeb klientów daje natomiast większe możliwości eksploatacji przez przedsiębiorstwa potencjalnych możliwości rynkowych, ze wszystkimi pozytywnymi dla nich konsekwencjami takiego stanu rzeczy.

2 ELEKTRONICZNY BIZNES I ELEKTRONICZNA GOSPODARKA

Sposób prowadzenia biznesu zależy od możliwości informacyjnych i komunikacyjnych, a te zależą od medium. Inaczej wyglądał biznes, gdy środkiem komunikacji był pergamin (produkowany z oślej skóry), a inaczej – gdy papier. Wynalazek telefonu, faksu, a ostatnio Internetu za każdym razem głęboko zmieniał sposób prowadzenia biznesu. Sam wynalazek to jeszcze za mało. Musi dojść do jego masowego upowszechnienia – dopiero wówczas mamy do czynienia ze znaczącymi zmianami gospodarczymi, a dalej społecznymi. Internet i telefon komórkowy są wynalazkami, które upowszechniły się bardzo szybko. Obecnie około 1,5 miliarda ludzi korzysta z Internetu, a ponad 3,3 miliarda – z telefonów komórkowych, które można uznać za inną formę dostępu do Internetu. Upowszechnienie informacji i komunikacji elektronicznej spowodowało pojawienie się na rynku produktów i usług cyfrowych, oferowanych i świadczonych przez Internet. Produkty i usługi cyfrowe mogą być oferowane lub świadczone klientom końcowym, na przykład piosenka do posłuchania lub możliwość założenia lokaty w e-banku. Mogą także być narzędziem zarządzania przedsiębiorstwami, czyli narzędziem realizacji procesów biznesowych przez sieć. Najważniejsze kategorie procesów biznesowych to:

promocja i badanie rynku, negocjacje, zamówienia, dostawy – przez Internet oczywiście tylko dostawy produktów i usług cyfrowych – oraz płatności.

Dysponując pojęciem produktu i usługi cyfrowej, możemy zdefiniować **elektroniczną gospodarkę**, jako taką, w której produkt i usługa cyfrowa są środkiem realizacji procesów biznesowych. Elektroniczna gospodarka dzieli się na dwa wielkie sektory: produktów materialnych i niematerialnych. Należy jednak mocno podkreślić, że o tym, czy gospodarka jest elektroniczna, czy nie, decyduje użycie produktów i usług cyfrowych do realizacji procesów biznesowych, a nie końcowy wytwór, który może być materialny. Innymi słowy, kopalnia, której wytworem jest jak najbardziej materialny węgiel, może być częścią elektronicznej gospodarki, jeśli przez Internet promuje się i bada rynek, prowadzi negocjacje, zbiera zamówienia i dokonuje płatności.

Elektronicznym biznesem nazywamy realizację procesów biznesowych przez sieć, ale w skali mikroekonomicznej, czyli na poziomie pojedynczego przedsiębiorstwa. Elektroniczny biznes dzieli się na elektroniczny handel i telepracę. **Elektroniczny handel** zapewnia dostęp do klientów i dostawców przez Internet, co prowadzi do wzrostu popytu na produkty i usługi. Analogicznie, **telepraca** zapewnia dostęp do pracowników przez Internet, co z kolei prowadzi do wzrostu podaży wiedzy, umiejętności i *know-how*. Obie te cechy mają kluczowe znaczenie dla funkcjonowania przedsiębiorstwa.

Procesy biznesowe są realizowane przez świadczenie usług informacyjnych, komunikacyjnych i transakcyjnych. **Usługi informacyjne** polegają na jednokierunkowym przekazaniu informacji, **usługi komunikacyjne** – na wymianie informacji między komunikującymi się stronami, a **usługi transakcyjne** są usługami komunikacyjnymi, pociągającymi za sobą skutki prawne. Dla przykładu, jeśli ktoś kupił coś przez Internet, to musi zapłacić, a sklep, który sprzedał – musi dostarczyć towar.

Istotą realizacji procesów biznesowych przez Internet jest wymiana elektronicznych dokumentów zamiast dokumentów papierowych oraz komunikacja międzyludzka prowadzona przez Internet zamiast bezpośrednich spotkań z klientami, dostawcami i pracownikami.

Główne zalety realizacji procesów biznesowych przez Internet są następujące:

- skrócenie czasu realizacji procesów biznesowych i wydłużenie ich dostępności do 24 godzin na dobę przez siedem dni w tygodniu;
- zmniejszenie kosztów realizacji procesów biznesowych, głównie dzięki eliminacji pośredników z łańcuchów dostaw;
- uniezależnienie procesów biznesowych od odległości geograficznych, co jest motorem globalizacji;
- możliwość automatycznej reakcji na sygnał inicjujący proces biznesowy, co jest motorem masowej personalizacji.

Wyróżniamy trzy rodzaje elektronicznego biznesu:

- przedsiębiorstwo – klient (ang. *Business to Customer* – **B2C**);
- przedsiębiorstwo – przedsiębiorstwo (ang. *Business to Business* – **B2B**);
- wnętrze przedsiębiorstwa, w tym przedsiębiorstwa wirtualne, lub szerzej – sieciowe organizacje wirtualne.

Sieciowe organizacje wirtualne stanowią największą innowację w sferze zarządzania przy wykorzystaniu środków informatyki i telekomunikacji, dlatego dalsza cześć tego wykładu będzie im poświęcona.

3 CHARAKTERYSTYKA ELEKTRONICZNEJ INFORMACJI I KOMUNIKACJI

Punktem wyjścia do zrozumienia przemian, jakie w organizacji przedsiębiorstw spowodowała informatyka i telekomunikacja, jest analiza cech **informacji elektronicznej**, czyli zdematerializowanej, w porównaniu z informacją zapisaną na nośniku papierowym, czyli materialnym.

Pierwszą wyróżniającą cechą informacji elektronicznej jest jej dostępność niezależnie od położenia geograficznego. Tę cechę zapewnia jej Internet realizowany za pomocą telekomunikacji stałej i ruchomej. W Internecie nie ma bowiem odległości geograficznych w naturalnym sensie – miarą odległości jest liczba kliknięć, a nie liczba kilometrów. Internet zapewnia więc użytkownikowi jednakową odległość od informacji niezależnie od jego własnego położenia geograficznego oraz niezależnie od położenia geograficznego źródła informacji.

Informacja elektroniczna jest dostępna przez Internet niezależnie od czasu. W przeciwieństwie do biur i sklepów, które są czynne w określonych godzinach, co jest regulowane prawem lub zwyczajem, systemy komputerowe są dostępne 24 godziny przez 7 dni w tygodniu, co zapewnia stały dostęp do informacji.

Przechowywanie informacji elektronicznej jest tanie, a jego koszt ciągle maleje dzięki postępowi we wszystkich rodzajach technologii pamięci – magnetycznych, optycznych i elektronicznych. Różnica w koszcie przechowywania informacji w postaci elektronicznej i papierowej – na korzyść tej pierwszej – jest jeszcze bardziej widoczna, jeśli weźmie się pod uwagę nie tylko koszt samego nośnika, ale łączne koszty archiwizowania dokumentów. Ponieważ przechowywanie informacji elektronicznej jest tanie, to jest możliwe tworzenie wielkich archiwów i repozytoriów. Dzięki temu na bieżąco może być dostępna informacja archiwalna niezależnie od roku jej wytworzenia. Ważnym aspektem jest też niski koszt dostępu do informacji elektronicznej przez Internet w porównaniu choćby z kosztami delegacji do archiwów gromadzących tradycyjnie przechowywane informacje na nośnikach papierowych.

Kolejną cechą informacji elektronicznej jest łatwość jej klasyfikacji zgodnie z wieloma kryteriami. W przypadku dokumentów papierowych jest konieczne ich fizyczne uporządkowanie zgodnie tylko z jednym wybranym porządkiem, na przykład chronologicznym lub alfabetycznym. Inny system jest realizowany przez ręczne zakładanie indeksów, na przykład indeksu rzeczowego w bibliotekach naukowych, co jednak jest na tyle żmudne i trudne, że rzadko praktykowane. Natomiast w przypadku informacji elektronicznej, porządek logiczny, czyli uporządkowanie dostępu do informacji, jest generalnie niezależny od porządku fizycznego, czyli rozłożenia rekordów informacyjnych na dysku. Budowanie indeksów reprezentujących różne porządki wymagane przez różnych użytkowników informacji jest w wielu przypadkach automatyczne lub półautomatyczne i dlatego często stosowane. Poprawia to w znaczy sposób dostęp do informacji, co ma szczególne znaczenie w przypadku dużych archiwów.

Informacja elektroniczna jest automatycznie wyszukiwalna. Jest możliwe efektywne przeszukiwanie archiwów informacji elektronicznej wspomagane komputerowo, i to zarówno na podstawie zawartości dokumentów, jak i na podstawie opisujących je metadanych. Wyszukiwanie to nie musi ograniczać się do pojedynczego archiwum, ale może mieć charakter zintegrowany w odniesieniu do dowolnej liczby rozproszonych archiwów.

Informacja elektroniczna poddaje się łatwej personalizacji. Na podstawie profilu użytkownika, który może być świadomie określony i na bieżąco uaktualniany dzięki monitorowaniu jego zachowania, jest możliwe filtrowanie informacji tak, aby nie przesyłać użytkownikowi informacji dla niego zbędnej.

Generalnie rzecz biorąc, przewaga informacji na nośniku elektronicznym nad informacją na nośniku papierowym wynika z możliwości jej automatycznego przetwarzania przez komputery zgodnie z założonym algorytmem. Na skutek przetworzenia informacji wzrasta jej wartość, w szczególności biznesowa, gdyż przyczynia się do podejmowania decyzji gospodarczych przekładających się na kondycję ekonomiczną przedsiębiorstwa – przychód, zysk, poszerzenie rynku, dopasowanie produkcji lub świadczonych usług do potrzeb klientów itp.

Podobnie jak elektroniczna informacja, również **cyfrowa komunikacja elektroniczna** ma wiele cech, które wpływają na sposób prowadzenia działalności gospodarczej. Po pierwsze za sprawą upowszechnienia telekomunikacji mobilnej, dzisiejsze połączenia mają charakter: człowiek – człowiek, a nie aparat telefoniczny – aparat telefoniczny. Dzięki temu telekomunikacja stała się niezależna od położenia geograficznego komunikujących się osób. Drugim efektem komunikacji mobilnej jest osiągalność każdego w każdym czasie.

Oczywiście telefon komórkowy można wyłączyć, ale w praktyce, w szczególności biznesowej, sprowadza się to jedynie do przełożenia komunikacji w czasie – odsłuchania nagrania z poczty głosowej i oddzwonienia.

Cyfrowa komunikacja mobilna jest tania, na co wpływa przede wszystkim brak konieczności położenia kabli do końcowych użytkowników. Z kolei technologie światłowodowe zapewniają bardzo niskie koszty przesyłu danych pomiędzy węzłami sieci ze względu na swoje ogromne przepustowości.

Komunikacja cyfrowa jest z natury multimedialna – umożliwia przesyłanie tekstu, głosu i obrazu wideo, a także danych informatycznych. W swojej istocie, komunikacja cyfrowa zapewnia przesyłanie bitów, co jest najbardziej uniwersalne. Interpretacja tych bitów, czyli odtworzenie tekstu, głosu lub obrazu wideo, zależy od końcowego urządzenia, a nie sieci telekomunikacyjnej.

Komunikacja cyfrowa, w zależności od potrzeb, zapewnia połączenia dwu- i wielopunktowe, czyli umożliwia organizowanie telekonferencji tekstowych (czaty), głosowych i wizyjnych.

4 CECHY BIZNESU W WARUNKACH ELEKTRONICZNEJ INFORMACJI I KOMUNIKACJI

Cechy elektronicznej informacji i komunikacji powodują, że przedsiębiorstwa i pracownicy przedsiębiorstw są przez cały czas dostępni w **przestrzeni „bez geografii”**, czyli w przestrzeni pozbawionej ograniczeń geograficznych. Elektronizacja umożliwia bowiem pozyskiwanie informacji zewsząd w czasie rzeczywistym i kontaktowanie się z każdym w każdej chwili. Oczywiście „zewsząd” i „z każdym” dotyczy przede wszystkim świata biznesu i to nawet stosunkowo zaawansowanego świata biznesu, ale to ta część biznesu nadaje ton całej gospodarce i wyznacza jej kierunki rozwoju.

Powszechna, stała dostępność w przestrzeni bez geografii przekłada się na wymaganie dynamizmu i różnorodności w działalności podmiotów gospodarczych. Dlatego w nowoczesnej gospodarce mamy do czynienia z ciągłą, wieloraką zmiennością. Zmienne są rynki, zarówno z perspektywy makroekonomicznej, gdyż często zmieniają się warunki gospodarowania i konkurowania na nich, jak i z perspektywy przedsiębiorstwa, ponieważ nowoczesne przedsiębiorstwa usilnie dążą do wejścia na nowe rynki. Zmienni są klienci, dostawcy i partnerzy biznesowi przedsiębiorstw, gdyż jest ułatwione wyszukiwanie informacji o potencjalnie nowych klientach, dostawcach i partnerach biznesowych oraz kontaktowanie się z nimi drogami elektronicznymi. Zmienne są technologie produkcji i świadczenia usług ze względu na naturalne próby uzyskiwania przewagi konkurencyjnej na drodze innowacji oraz wdrożeń wyników działalności badawczo-rozwojowej. Bardzo duża część tej niestabilności jest wynikiem stałego udoskonalania i rozwoju oprogramowania systemów komputerowych stosowanych do produkcji i świadczenia usług. Niestałe są metody pracy, w czym znowu duży udział ma konieczność nabycia przez pracowników umiejętności posługiwania się nowym oprogramowaniem stosowanym w pracy. Przeobrażeniu ulega organizacja pracy, w szczególności na stanowiskach, na których wynikiem pracy jest pewna informacja lub komunikacja. Przykładem takich zmian jest **telepraca**, czyli świadczenie pracy na odległość. Wreszcie, zmiany technologiczne i organizacyjne w gospodarce w naturalny sposób pociągają za sobą zmiany prawne, do których przedsiębiorstwa muszą się na bieżąco dostosowywać.

Zróżnicowanie działalności przedsiębiorstw objawia się w skali makro- i mikroekonomicznej. W skali makroekonomicznej mamy, po pierwsze, do czynienia ze zróżnicowaniem geograficznym – współczesne przedsiębiorstwa, nawet średnie i małe, prowadzą działalność na różnych kontynentach, w różnych strefach czasowych, w różnych klimatach itp. Działając na skalę międzynarodową, przedsiębiorstwa mają do czynienia ze zróżnicowaniem prawnym – państwa posiadają odmiene systemy prawne. Nawet w ramach jednych organizmów gospodarczych, takich jak Unia Europejska, prawo w różnych krajach jest dalece niejednolite. Ważniejsze, bo bardziej subtelne i przez to trudniejsze do zarządzania, są różnice kulturowe. Znajomość lokalnej specyfiki kulturowej jest prawie zawsze warunkiem udanych przedsięwzięć biznesowych. Bardzo często to właśnie szeroko rozumiana kultura decyduje o akceptacji lub jej braku poszczególnych produktów i usług na różnych rynkach.

W skali mikroekonomicznej naturalną współczesną tendencją jest dążenie do realizacji całościowych potrzeb klientów przez przedsiębiorstwo. U podstaw takiego dążenia leży przekonanie, że kosztownym elementem każdego procesu biznesowego jest pozyskanie klienta przez przedsiębiorstwo. Dlatego jeśli uda się pozyskać klienta, to należy zaoferować mu jak najszerszy zestaw produktów i usług. To jednak oznacza, że takim szerokim i zróżnicowanym zestawem produktów i usług przedsiębiorstwo musi dysponować.

5 SIECIOWE ORGANIZACJE WIRTUALNE

Odpowiedzią na gospodarcze wymaganie dynamizmu i zróżnicowania jest tworzenie **sieciowych organizacji wirtualnych** (**SOW**), czyli zbiorów współpracujących ze sobą przez Internet jednostek gospodarczych, występujących na rynku tak, jakby były jednym przedsiębiorstwem. Sieciowe organizacje wirtualne mogą prowadzić działalność polegającą zarówno na oferowaniu produktów i usług, jak i zamawianiu produktów i usług na swoje potrzeby.

Procesy tworzenia się SOW mogą przebiegać zarówno od góry do dołu, jak i od dołu do góry. W pierwszym przypadku mamy do czynienia z dekompozycją tradycyjnych, dużych organizacji hierarchicznych. Natomiast w drugim przypadku chodzi o integrację małych i średnich przedsiębiorstw.

W epoce, w której dominował papierowy obieg informacji, w szczególności obieg papierowych dokumentów (ta epoka właśnie na naszych oczach się kończy), hierarchiczna organizacja przedsiębiorstw była jak najbardziej właściwa. Ponieważ obieg informacji na nośniku papierowym był wolny, kosztowny i zależny od odległości geograficznych, to ekonomicznie uzasadnione było przyjęcie sztywnych założeń co do ról i funkcji jednostek organizacyjnych, składających się na pewną gospodarczą całość oraz struktury ich powiązań. W ten sposób minimalizowano bowiem koszty obiegu informacji. Innymi słowy, przy takich założeniach, bez komunikacji i wymiany informacji, z góry było wiadomo, kto, co i na kiedy ma zrobić. Gwoli sprawiedliwości warto też zauważyć, że jeszcze do niedawna, przy stosunkowo niskim poziomie informatyzacji i robotyzacji produkcji oraz informatycznego wsparcia świadczenia usług, przedsiębiorstwa nie były zdolne do dokonywania szybkich zmian. Zatem szybki obieg informacji nie był im potrzebny, bo rynek nie był dynamiczny.

W dzisiejszej gospodarce, w której już dominuje informacja i komunikacja elektroniczna, choć jeszcze często w odniesieniu do dokumentów dublowana na papierze, taka sztywna, hierarchiczna organizacja oparta na stałych funkcjach i rolach jest nieefektywna. W epoce informacji elektronicznej decyzje biznesowe mogą i powinny być podejmowane na bazie komunikacji, czyli wymiany informacji, a nie ustalonych z góry ról i funkcji. Skoro bowiem pracownicy przedsiębiorstw mogą przez cały czas pozyskiwać informacje zewsząd w czasie rzeczywistym i skontaktować się z każdym w każdej chwili, to nie powinni być ograniczeni w swojej przedsiębiorczości przez sztywną strukturę organizacyjną, tym bardziej, że współczesna technologia umożliwia dokonywanie szybkich zmian w produkcji i usługach. W epoce elektronicznej informacji i komunikacji płaska organizacja sieciowa oparta na wymianie informacji i komunikacji daje większe możliwości optymalizacji działalności gospodarczej i lepsze dopasowanie do chwilowych, szybko zmieniających się potrzeb rynków.

Równolegle z dekompozycją dużych organizacji hierarchicznych przebiega integracja małych i średnich przedsiębiorstw w sieciowe organizacje wirtualne. Głównym powodem tej integracji jest przekonanie, że małe lub średnie przedsiębiorstwo samo nie jest w stanie sprostać wyzwaniom i podołać konkurencji na dynamicznym i zróżnicowanym rynku. Dlatego jest konieczne włączenie się takiego przedsiębiorstwa w większy organizm gospodarczy. Integracja z sieciową organizacją wirtualną jest bardzo dobrym rozwiązaniem, gdyż umożliwia małemu lub średniemu przedsiębiorstwu zachować swą podmiotowość, na której często właścicielom firmy bardzo zależy. W procesie integracji, małe i średnie przedsiębiorstwa mogą albo dołączać do sieciowych organizacji wirtualnych wynikających z dekompozycji dużych organizacji hierarchicznych, które się na nie otwierają, albo próbować samoorganizować się w SOW w celu oferowania bardziej złożonych produktów i usług, i poprawy swojej pozycji konkurencyjnej na większym rynku.

6 CECHY SOW

Sieciowe organizacje wirtualne charakteryzują się trzema zasadniczymi cechami:

- kulturą biznesową i sposobem funkcjonowania ukierunkowanym na podążanie za nieustannie zmieniającymi się potrzebami klientów;
- skoncentrowaniem się każdej jednostki wchodzącej w skład sieciowej organizacji wirtualnej, na doskonaleniu swoich kluczowych kompetencji i opieraniem się na zaufaniu do partnerów w odniesieniu do pozostałych funkcji;
- standaryzacją danych, systemów informatycznych i procesów biznesowych w skali całej sieciowej organizacji wirtualnej, w celu zapewnienia wysokiej efektywności gospodarczej.

Pierwsza cecha jest spełnieniem wymagania współczesnego rynku, na którym konkuruje się przede wszystkim zdolnością do szybkich zmian. Kultura biznesowa ukierunkowana na podążanie za zmianami wymaga przede wszystkim dowartościowania kreatywności i innowacyjności oraz takiej wewnętrznej organizacji, aby nowatorskie pomysły powstające w przedsiębiorstwie nie były tracone, ale aby były doprowadzane do wdrożeń przemysłowych.

Druga cecha przedsiębiorstw wchodzących w skład wirtualnych organizacji sieciowych, czyli skoncentrowanie się na doskonaleniu swoich kluczowych kompetencji, wymaga od przedsiębiorców porzucenia myśli o samowystarczalności i przekonania, że samemu wszystko zrobi się najlepiej, na rzecz zaufania do wyspecjalizowanych partnerów z sieciowej organizacji wirtualnej. Dopiero po przełamaniu bariery nieufności do partnerów można skoncentrować wysiłki na swojej kluczowej kompetencji i doskonalić ją oraz rozwijać. Przez kluczową kompetencję rozumie się tutaj przede wszystkim te umiejętności, które zapewnią przedsiębiorstwu przewagę konkurencyjną na rynku w przyszłości.

Trzecią główną cechą przedsiębiorstw wchodzących w skład SOW jest konieczność standaryzacji danych, systemów informatycznych i procesów biznesowych w skali całej sieciowej organizacji wirtualnej, w celu zapewnienia jej wysokiej efektywności gospodarczej. Standaryzacja obniża koszty, ponieważ eliminuje procesy tłumaczenia jednych standardów na drugie, umożliwia łatwą integrację systemów informatycznych oraz pozwala na wspólne realizowanie skomplikowanych procesów biznesowych wymagających wsparcia informatycznego.

7 TRANSFORMACJA PRZEDSIĘBIORSTW DO SOW

Efekt koncentracji przedsiębiorstwa na swoich kluczowych kompetencjach uzyskuje się na drodze transformacji funkcjonalnej i operacyjnej. **Transformacja funkcjonalna** polega na wydzielaniu z przedsiębiorstwa funkcji oraz zadań. W przypadku **wydzielania funkcji** (ang. *outsourcing*), jednostka wydzielająca powierza kontrolę nad całym procesem biznesowym partnerowi zewnętrznemu i interesuje się tylko wynikiem jego działań. Natomiast w przypadku **wydzielania zadań** (ang. *out-tasking*), jednostka wydzielająca zachowuje kontrolę nad sposobem wykonania zadania przez zewnętrznego partnera. O ile wydzielanie funkcji jest praktyką biznesową stosowaną od dawna w celu obniżenia kosztów funkcjonowania przedsiębiorstw, to wydzielanie zadań jest charakterystyczne dla sieciowych organizacji wirtualnych, gdyż wymaga zaawansowanej integracji systemów informatycznych współpracujących partnerów.

W celu przeprowadzenia transformacji funkcjonalnej, działania wykonywane w przedsiębiorstwie dzieli się według dwóch kryteriów. Pierwszym kryterium jest ryzyko dla biznesu firmy, które dzieli działania przedsiębiorstwa na takie, które jeśli są nieodpowiednie, to bezpośrednio wpływają na biznes firmy, oraz na takie, które nawet jeśli są źle prowadzone to bezpośrednio na biznes firmy nie oddziaływują. Te pierwsze nazywamy **działaniami krytycznymi** dla misji firmy, a te drugie – **niekrytycznymi**. Drugim kryterium jest odróżnienie od konkurencji, które dzieli działania na takie, które bezpośrednio dotyczą przewagi konkurencyjnej firmy, i takie, które na przewagę konkurencyjną bezpośrednio nie wpływają. Te pierwsze to **działania kluczowe**, a te drugie – **kontekstowe**. Działania kluczowe i krytyczne dla misji przedsiębiorstwa należy w nim pozostawić, gdyż stanowią istotę jego działalności gospodarczej. Działania kontekstowe i niekrytyczne dla misji firmy należy wydzielić jako funkcje i interesować się tylko wynikiem działania partnera wypełniającego tę funkcję. Natomiast działania kluczowe, choć niekrytyczne dla misji, oraz krytyczne, ale kontekstowe, należy wydzielić jako zadania i interesować się nie tylko ich wynikiem, ale także sposobem ich realizacji.

Dla przykładu rozważmy operatora telekomunikacyjnego. Łączenie rozmów jest działaniem kluczowym i krytycznym dla jego misji. Jeśli operator telekomunikacyjny nie będzie łączył (dobrze) rozmów, to klienci odejdą od niego do konkurencji, a finanse firmy ulegną załamaniu. Wystawianie rachunków klientom jest działaniem kluczowym, bo bez nich firma nie uzyska przychodów, ale nie krytycznym dla misji firmy – klienci operatora telekomunikacyjnego nie odejdą od niego tylko dlatego, że rachunki dostają z opóźnieniem, a niektórzy nawet ucieszą się z tego. Księgowanie należności jest przykładem działania kontekstowego i krytycznego dla misji firmy. Jeśli coś zostanie źle zaksięgowane, to zawsze można to poprawić, więc w ostatecznym rozrachunku nie wpłynie to na biznes firmy. Jednak jeśli klienci będą otrzymywać błędne kwoty do zapłaty, to mogą się zdenerwować i przejść do konkurencji. Natomiast sprzątanie siedziby firmy jest przykładem działania kontekstowego i niekrytycznego dla misji firmy i jako takie może być wydzielone jako funkcja. Operator jest zainteresowany tylko tym, aby jego siedziba była czysta, a w jaki sposób jego partner sprząta jest jego sprawą.

8 ARCHITEKTURA USŁUGOWA SOA

Największym wyzwaniem stojącym przed sieciowymi organizacjami wirtualnymi jest odpowiedź na pytanie, jak wydzielić zadanie, ale zachować kontrolę nad sposobem wykonania go przez zewnętrznego partnera. Odpowiedź brzmi – przez integrację systemów informatycznych. Należy bardzo silnie podkreślić, że w warunkach nowoczesnej gospodarki nie wystarcza zintegrowany system informatyczny do zarządzania wnętrzem przedsiębiorstwa. System informatyczny przedsiębiorstwa musi być zdolny do integracji zewnętrznej, czyli do integracji z systemami informatycznymi przedsiębiorstw-partnerów, wchodzących w skład sieciowej organizacji wirtualnej. Warunkiem takiej integracji jest odpowiednia architektura rozproszonych systemów informatycznych – w tym przypadku **architektura usługowa SOA** (ang. *Service Oriented Architecture*). Jest to forma organizacyjna, dzięki której można dynamicznie integrować dostępne działania rozproszonych, niezależnych jednostek w celu świadczenia usług na żądanie.

Technologiczną podstawą architektury usługowej SOA są **usługi sieciowe** (ang. *web services*). Istotą usługi sieciowej jest programowalny interfejs (a nie opisowy, jak to ma miejsce w tradycyjnych rozwiązaniach). Dzięki programowalnemu interfejsowi usługi sieciowej oferowanej przez jedno przedsiębiorstwo, komputer innego przedsiębiorstwa może automatycznie (bez pomocy człowieka) zorientować się, jak korzystać z tej usługi, czyli jakie żądania może wysyłać oraz jakich odpowiedzi w i jakim formacie może się spodziewać. W związku z tym nieznane sobie nawzajem komputery różnych przedsiębiorstw mogą się dynamicznie integrować.

Architektura usługowa SOA jest informatyczną odpowiedzią na następujące wymagania współczesnej gospodarki. Po pierwsze zapewnia obsługę całościowych procesów użytkowników dzięki umożliwieniu współpracy niezależnych jednostek gospodarczych i administracyjnych. Po drugie jest odpowiedzią na nieznany wcześniej dynamizm zmian rynkowych i regulacyjnych w skali całego świata. Po trzecie umożliwia masową personalizację usług. Wreszcie – pozwala na harmonijne łączenie usług świadczonych przez komputery i przez ludzi.

Resumując znaczenie integracji informatycznej należy stwierdzić, że tylko zautomatyzowana wymiana danych zapewnia odpowiednią jakość zarządzania na poziomie całego łańcucha dostaw, a tym samym niskie koszty, krótki czas reakcji i szybkie dostosowywanie się do zmian na rynku. Natomiast tylko zarządzanie całymi łańcuchami dostaw zapewnia spełnienie całościowych potrzeb klientów.

WNIOSKI

Jak wynika z powyższych rozważań, przyszłość należy do małych i średnich przedsiębiorstw, ale tylko takich, które są informatycznie zintegrowane w sieciowe organizacje wirtualne. W globalnej gospodarce małe i średnie przedsiębiorstwa, same sobie nie poradzą. Kluczowo ważna jest dla nich współpraca na skalę międzynarodową. Formą takiej współpracy są sieciowe organizacje wirtualne, a podstawową technologią – technologie elektronicznego biznesu oparte na środkach i metodach informatyki i telekomunikacji. Szansą polskiej gospodarki jest duża liczba młodych, dobrze wykształconych osób, które mogą tworzyć innowacyjne przedsiębiorstwa świadczące usługi oparte na wiedzy, stanowiące znaczące komponenty sieciowych organizacji wirtualnych. Takie przedsiębiorstwa byłyby łącznikiem z gospodarką światową tych tradycyjnych polskich małych i średnich przedsiębiorstw, które nie mają wystarczających kompetencji w zakresie wielokulturowości, interdyscyplinarności, innowacyjności i zdolności prowadzenia biznesu przez Internet. Osoby, które chciałyby takie nowoczesne przedsiębiorstwa tworzyć i w nich pracować muszą łączyć wiedzę z ekonomii i zarządzania z informatyką. Warto pomyśleć o tym jak najwcześniej, ponieważ elektroniczny biznes jest jedną z najbardziej obiecujących dziedzin zastosowań informatyki.

Odpowiedź na pytanie zadane w tytule tego wykładu – *Czy komputery będą robić biznes?* – brzmi – nie, ale bez komputerów nie da się robić biznesu!

LITERATURA

Kraska M. (red.), *Elektroniczna gospodarka w Polsce – Raport 2008*, Wydawnictwo ILiM, Poznań 2009

Algorytmika Internetu

Krzysztof Diks

Instytut Informatyki, Uniwersytet Warszawski

diks@mimuw.edu.pl

Streszczenie

W sieci Internet znajduje się blisko 50 000 000 000 stron, liczba adresów IP zbliża się do 3 000 000 000. Jak to jest możliwe, że pomimo olbrzymiego rozmiaru Internetu jesteśmy w stanie niezmiernie szybko odnaleźć interesujące nas informacje, dzielić się filmami i muzyką, zdobywać przyjaciół w miejscach, do których nigdy nie dotarlibyśmy osobiście?

Okazuje się, że okiełznanie sieci wymagało wynalezienia i zaimplementowania odpowiednich algorytmów. O niektórych z nich jest mowa na tym wykładzie.

Spis treści

1 WSTĘP

Algorytmika jest działem informatyki, który zajmuje się projektowaniem i analizowaniem algorytmów. Komputery bez algorytmów zapisanych w postaci programów okazują się bezużyteczne. Żeby jednak zaprząc komputery do realizowania pożądanych przez nas zadań musimy być przekonani, że wykorzystywane algorytmy są poprawne, tzn. że dla danych spełniających określone kryteria, po wykonaniu algorytmu otrzymamy oczekiwane wyniki, oraz że są one możliwe do wykonania na dostępnym sprzęcie – wykorzystywane komputery dysponują pamięcią o wystarczającej pojemności, a obliczenia zakończą się w akceptowanym przez nas czasie. Zatem główne aspekty analizy algorytmów to ich poprawność i wydajność. Analizy algorytmów dokonujemy abstrahując od sprzętu, na którym będą wykonywane. Powszechnie przyjmuje się, że algorytmy szybkie, to takie, które wykonują się w czasie wielomianowym ze względu na rozmiar danych. Okazuje się, że w dobie Internetu, nawet algorytmy liniowe mogą być nie do wykorzystania na współczesnych komputerach z powodu olbrzymich rozmiarów danych, które muszą być przetwarzane. A jednak potrafimy wyszukiwać w tak monstrualnej sieci jaką jest Internet oraz poznawać jej strukturę. Jak to jest możliwe? Ten wykład to próba naszkicowania odpowiedzi na te pytania.

2 AUTORSKA, BARDZO KRÓTKA HISTORIA ALGORYTMÓW

Historia algorytmów sięga starożytności. Około 350 lat przed naszą erą pojawił się powszechnie dziś wykładany w szkole algorytm Euklidesa, służący do obliczania największego wspólnego dzielnika dwóch liczb naturalnych. W roku 1844 Francuz Gabriel Lamé udowodnił, że liczba dzieleń wymagana w pesymistycznym przypadku do znalezienia największego wspólnego dzielnika dwóch liczba algorytmem Euklidesa jest nie większa niż pięć razy liczba cyfra w zapisie dziesiętnym mniejszej z tych liczb, co oznacza, że działa on w czasie wielomianowym ze względu na rozmiar danych. Wynik Lamé możemy traktować jako pierwszą analizę złożoności czasowej algorytmu Euklidesa i początek teorii złożoności obliczeniowej.

Pojęcie algorytmu, choć w obecnych czasach intuicyjnie oczywiste, wymagało formalizacji po to, żeby algorytmy mogły być badane precyzyjnymi, matematycznymi metodami. W roku 1936 Alan Turing zaproponował teoretyczny model maszyny liczącej znanej od tego czasu jako maszyna Turinga. Mimo swej prostoty moc obliczeniowa maszyny Turinga rozumiana jako zdolność rozwiązywania problemów algorytmicznych przy zadanych zasobach (pamięci i czasie) odpowiada współczesnym komputerom.

W roku 1962 Charles Hoare zaproponował najpopularniejszy dziś algorytm sortowania – QuickSort. Algorytm QuickSort działa pesymistycznie w czasie kwadratowym, ale znakomicie zachowuje się dla przeciętnych (losowych) danych. Mniej więcej w tym samym czasie Jack Edmonds, zajmujący się algorytmami grafowymi, zdefiniował klasę P tych problemów algorytmicznych, które można rozwiązać w czasie wielomianowym. Jednocześnie okazało się, że dla wielu ważnych problemów optymalizacyjnych nie można było znaleźć algorytmów działających w czasie wielomianowym. Wytłumaczenie tego zjawiska pojawiło się w roku 1971, kiedy to Stephen Cook podał pierwszy, tak zwany problem NP-zupełny, wykazując, że jeśli potrafilibyśmy sprawdzać w czasie wielomianowym, czy dana formuła logiczna (zdaniowa) ma wartościowanie, dla którego przyjmuje wartość prawda, to wiele innych problemów, dla których poszukiwano bezskutecznie wielomianowego rozwiązania, dałoby się także rozwiązać w czasie wielomianowym. W tym samym roku Richard Karp wskazał osiem innych problemów NP-zupełnych. Obecnie lista takich problemów liczy tysiące pozycji. To, czy problemy NP-zupełne mają wielomianowe rozwiązanie, jest jednym z siedmiu tzw. problemów milenijnych. Za rozwiązanie każdego z nich zaoferowano milion dolarów. Jeden z tych problemów – hipoteza Poincaré – został już rozwiązany. Jedyne znane algorytmy dla problemów NP-zupełnych działają w czasie wykładniczo zależnym od rozmiaru

danych. Oznacza to, że dla danych dużych rozmiarów, ale wciąż spotykanych w praktyce, nie zawsze jest możliwe znalezienie rozwiązania z pomocą współczesnych komputerów ze względu na nadmiernie długi czas obliczeń.

Dzisiaj nie znamy algorytmu służącego do rozkładania liczby na czynniki będące liczbami pierwszymi w czasie wielomianowo zależnym od długości jej komputerowej reprezentacji. Fakt ten jest podstawą protokołu kryptograficznego RSA wynalezionego w roku 1977. Z drugiej strony, w roku 2002 wykazano, że to, czy liczba naturalna jest liczbą pierwszą, można sprawdzić w czasie wielomianowym. Czy powinniśmy się obawiać o protokół RSA?

3 TRZY PROSTE PROBLEMY ALGORYTMICZNE I ICH ROZWIĄZANIA

W tym rozdziale omówimy trzy proste problemy algorytmiczne, podamy ich rozwiązania oraz zanalizujemy złożoność obliczeniową zaproponowanych rozwiązań.

3.1 LIDER

Dane: dodatnia liczba całkowita n oraz ciąg liczb całkowitych $a[1]$, a[2], ..., $a[n]$.
Wynik: liczba całkowita x, która pojawia się w ciągu a więcej niż połowę razy (czyli taka liczba x, że $|\{i: a[i] = x\}| > n/2$), o ile takie x istnieje, w przeciwnym przypadku dowolny element z ciągu a.

Rozwiązanie, które proponujemy, wykorzystuje następujące spostrzeżenie. Niech u, v będą dwoma różnymi elementami ciągu a. Jeśli x jest liderem w a, to jest także liderem w ciągu a z usuniętymi elementami u i v. Innymi słowy, jeśli x jest liderem i każdy element w ciągu a o wartości różnej od x sparujemy z elementem o wartości różnej od w, to pozostaną tylko niesparowane elementy o wartościach równych x. Oto algorytm wykorzystujący powyższą ideę.

```
Algorytm Lider::
x := pierwszy element ciągu;
licz := 1;                                   //mamy licz niesparowanych elementów o wartości x
while nie koniec ciągu do
{
      y := kolejny element ciągu;
      if licz = 0 then
         { x := y; licz := 1}
      else if x = y then
            licz := licz +1
          else
            licz := licz - 1                 //parowanie
}
return x;
```

Proponujemy zastanowienie się, dlaczego ten algorytm jest poprawny. Skupmy się na jego złożoności. Za rozmiar zadania w tym przypadku przyjmujemy n – długość ciągu a. W każdym obrocie pętli pobieramy jeden element ciągu i wykonujemy na nim stałą liczbę operacji. Zatem złożoność czasowa jest liniowa, co w notacji asymptotycznej wyraża się formułą $O(n)$.

Zauważmy, że algorytm nie gwarantuje tego, że wynik x jest liderem. Żeby to stwierdzić, potrzebny jest jeszcze jeden przebieg przez dane i policzenie liczby wystąpień x w ciągu. W niektórych zastosowaniach do przetwarzania danych w Internecie drugi przebieg nie jest możliwy.

3.2 MNOŻENIE MACIERZY PRZEZ WEKTOR

Dane: liczba naturalna $n > 0$, kwadratowa macierz liczb rzeczywistych $A[1..n,1..n]$, wektor liczb rzeczywisty $x[1..n]$.
Wynik: wektor $y[1..n] = Ax$, gdzie $y[i] = A[i,1]*x[1] + A[i,2]*x[2] + ... + A[i,n]*x[n]$.

Algorytm mnożenia macierzy przez wektor wynika wprost z opisu wyniku.

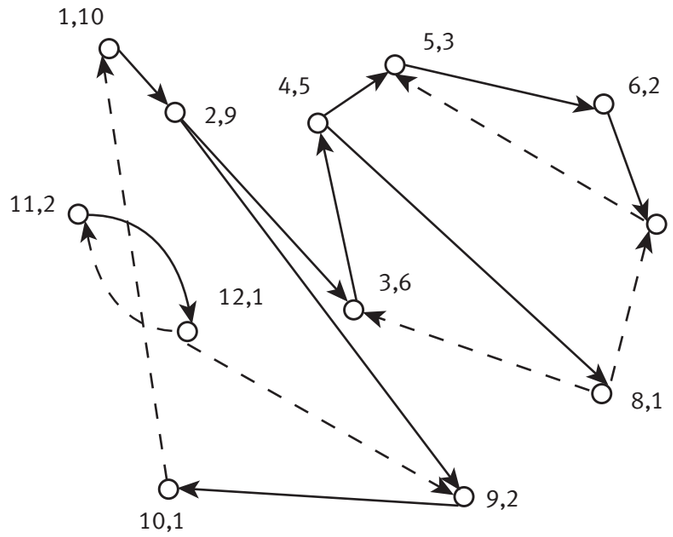
```
Algorytm Macierz_x_Wektor::
  for i := 1 to n do
  {
     y[i] := 0;
     for j := 1 to n do
        y[i] := y[i] + A[i,j]*x[j]
  }
  return y;
```

Zanalizujmy teraz czas działania powyższego algorytmu. Najdroższą operację w tym algorytmie jest operacja mnożenia dwóch liczb rzeczywistych. Wykonujemy n^2 takich mnożeń. Jeśli za rozmiar zadań przyjmiemy właśnie n^2, to nasz algorytm działa w czasie liniowym ze względu na rozmiar danych – liczbę elementów macierzy A. Algorytmy liniowe są uważane za szybkie. Rozważmy czas działania naszego algorytmu dla $n =$ 100 000 000. Dalej się okaże, że nie jest to rozmiar „wzięty z sufitu". Przyjmijmy dodatkowo, że w ciągu jednej sekundy możemy wykonać 10^8 mnożeń. Wówczas czas potrzebny na wykonanie wszystkich mnożeń w naszym algorytmie wyniósłby 100 000 000 sekund, czyli około 1600 dni! Jest jeszcze jeden problem. Gdybyśmy na zapisanie jednej liczby rzeczywistej przeznaczyli 8 bajtów, to musielibyśmy mieć pamięć o rozmiarze $8*10^{16}$ bajtów, czyli więcej niż petabajt! Tak więc, używając konwencjonalnych komputerów, nie bylibyśmy w stanie realnie rozwiązać tego zadania.

W praktyce często pojawiają się macierze, w których wiele elementów to zera. Zauważmy, że jeżeli $A[i,j] = 0$, to wykonanie instrukcji $y[i] := y[i] + A[i,j]*x[j]$ nie zmienia wartości $y[i]$. Używając technik listowych można pominąć mnożenia przez 0. Wówczas liczba mnożeń (i dodawań) jest proporcjonalna do liczby niezerowych elementów macierzy A. Oznaczmy liczbę niezerowych elementów macierzy A przez $Nz(A)$. Czas działania algorytmu w tym przypadku wynosi $O(\max(n,Nz(A))$, co, gdy $Nz(A)$ jest liniowe ze względu na n, daje algorytm mnożenia macierzy w czasie liniowo zależnym od n. Wówczas nawet dla $n =$ 100 000 000 można się spodziewać wyniku w rozsądnym czasie. Macierz, której liczba niezerowych elementów jest liniowa ze względu na wymiar macierzy, nazywamy macierzą rzadką. Używając technik listowych macierze rzadkie można reprezentować w pamięci o rozmiarze proporcjonalnym do liczby niezerowych elementów macierzy, co dla $n =$ 100 000 000 jest znacząco mniej niż petabajt.

3.3 SILNIE SPÓJNE SKŁADOWE

Powiemy, że graf skierowany jest silnie spójny, jeśli dla każdej pary węzłów u i v istnieją skierowane ścieżki z u do v i v do u. Silnie spójną składową skierowanego grafu G nazywamy każdy jego maksymalny (w sensie zwierania) silnie spójny podgraf. Silnie spójnymi składowymi w grafie z rysunku 1 są podgrafy rozpięte na węzłach identyfikowanych przez pierwsze liczby w parach: {1,2,9,10}, {3,4,8}, {5,6,7}, {11,12}.



Rysunek 1.
Graf skierowany

Specyfikacja zadania Silnie spójne składowe jest następująca:
Dane: $G=(V,E)$ – graf skierowany.
Wynik: funkcja $s: V \quad \{1,\ldots,|V|\}$ taka, że dla każdej pary węzłów u, v, $s(u) = s(v)$ wtedy i tylko wtedy, gdy istnieją ścieżki w grafie G z u do v i z v do u.

Złożoność algorytmów znajdowania silnie spójnych składowych w grafie zależy od reprezentacji grafu. Są dwie zasadnicze reprezentacje: tablicowa, w której w tablicy kwadratowej A, indeksowanej węzłami, mamy $A[u,v] = 1$, gdy istnieje krawędź z u do v, natomiast $A[u,v] = 0$, gdy takiej krawędzi nie ma. Niezależnie od liczby krawędzi w grafie rozmiar takiej reprezentacji wynosi $|V|^2$. Ponieważ liczba krawędzi w grafie skierowanym waha się od 0 do $|V|\times(|V|-1)$, reprezentacją uwzględniającą ten fakt są listy sąsiedztwa. W tej reprezentacji dla każdego węzła przechowujemy listę sąsiadów, do których prowadzą krawędzie z tego węzła – tzw. listę „w przód” – oraz listę sąsiadów, od których prowadzą krawędzie do tego węzła – tzw. listę „w tył”. Rozmiar reprezentacji grafu w postaci list sąsiedztwa wynosi $O(|V| + |E|)$ i jest ona szczególnie wygodna i oszczędna w przypadku grafów rzadkich, to znaczy takich, w których liczba krawędzi liniowo zależy od liczby węzłów. W algorytmie wyznaczania silnie spójnych składowych opisanym poniżej przyjmiemy tę drugą reprezentację.

Przedstawimy teraz krótko algorytm, którego pełny opis wraz z dowodem poprawności można znaleźć w książce [1]. Na potrzeby opisu załóżmy, że zbiór węzłów to $V = \{1,2, \ldots, |V|\}$. W celu znalezienia silnie spójnych składowych przeszukujemy graf G dwukrotnie w głąb. Za pierwszym razem wykorzystujemy listy w przód, gdy w następnym przeszukiwaniu korzystamy z list w tył. Przeszukując graf w przód, numerujemy węzły w kolejności odwiedzania i dla każdego węzła liczymy liczbę węzłów w poddrzewie przeszukiwania o korzeniu w tym węźle. Na rysunku 1 pierwsze liczby w parach odpowiadają numerom w kolejności przeszukiwania (tutaj identyfikujemy je z węzłami), a drugie liczby w parach mówią o rozmiarach poddrzew przeszukiwania w przód. Krawędzie lasu przeszukiwania w przód są narysowane ciągłą, pogrubioną kreską. Zauważmy, że numeracja

i liczba węzłów w poddrzewach pozwala na łatwe stwierdzenie, czy węzeł v jest w poddrzewie przeszukiwania o korzeniu w u. Wystarczy sprawdzić, czy numer v jest nie mniejszy od numeru u, ale mniejszy od numeru u plus liczba węzłów w poddrzewie o korzeniu u. Dla przykładu węzeł o numerze 7 jest w poddrzewie węzła o numerze 3, ponieważ $3 \le 7 < 3 + 6$; węzeł o numerze 11 nie jest w poddrzewie węzła o numerze 1, ponieważ $11 \ge 1 + 10$.

Podczas przeszukiwania grafu (po listach) w tył wykrywamy silnie spójne składowe. Identyfikatorem silnie spójnej składowej będzie węzeł o najmniejszym numerze z tej składowej. Dlatego w drugiej fazie przeglądamy węzły w kolejności rosnących numerów. Gdy napotkamy węzeł, który nie był jeszcze widziany w drugiej fazie, to rozpoczynamy z niego przeszukiwanie w tył, ale tylko po węzłach, które są w poddrzewie przeszukiwania w przód o korzeniu w węźle, od którego rozpoczęliśmy przeszukiwanie właśnie wykrywanej składowej. Wszystkie napotkane w ten sposób węzły będą należały do silnie spójnej składowej, której identyfikatorem będzie węzeł, od którego rozpoczynaliśmy przeszukiwanie. Rozważmy jeszcze raz graf z rysunku 1. Rozpoczynamy od węzła 1 i idziemy po krawędziach w tył. W ten sposób dochodzimy do węzła 10 i ponieważ jest on w poddrzewie w przód węzła 1, to zaliczamy go do silnie spójnej składowej o numerze 1. Z węzła 10 próbujemy węzeł 9 i też zaliczamy go do silnie spójnej składowej o numerze 1. Z węzła 9 próbujemy przejść do węzła 12, ale nie jest on w poddrzewie w przód węzła 1, więc nie należy do silnie spójnej składowej o identyfikatorze 1. Kolejny węzeł odwiedzany z węzła 9 to 2 i zaliczamy go do silnie spójnej składowej węzła numer 1. W ten sposób wykryjemy całą silnie spójną składową zawierającą węzeł 1. Wszystkie jej węzły są zaznaczone jako odwiedzone. Przeglądając kolejno węzły, docieramy do węzła 3, jako pierwszego nieodwiedzonego w tył, i teraz z niego odkrywamy silnie spójną składową o identyfikatorze 3 (węzły 3, 4, 8). Następnie zostanie wykryta składowa o identyfikatorze 5 (węzły 5, 6, 7), a na koniec o identyfikatorze 11 (węzły 11 i 12).

Oto formalny zapis omówionego algorytmu.

```
Algorytm SilnieSpójneSkładowe::
Faza I::
W_przód(v: węzeł)
{
  ost_nr := ost_nr + 1; nr[v] := ost_nr;                    // ost_nr – ostatnio nadany numer
  wezly[ost_nr] := v;                                       // porządkowanie węzłów według numerów
  for each u – węzeł na liście w przód węzła v do
    if nr[u] = 0 then   W_przód(u)                          // nr[u] = 0 oznacza, że węzeł nie został odwiedzony;
  w_poddrzewie[v] := ost_nr – nr[v] + 1                     // rozmiar poddrzewa w przód
}
Przeszukiwanie w przód::
  ost_nr := 0;
  for each węzeł v do nr[v] := 0;
  for each węzeł v do
    if nr[v] = 0 then W_przód(v);

Faza II::
W_tył(v: węzeł, id_s: 1..|V|)                               // id_s – id aktualnie wykrywanej składowej
{
  s[v] := id_s;
  for each węzeł u na liście w tył węzła v do
    if ((s[u] = 0)                                          // u nie był jeszcze odwiedzony
        AND                                                 // oraz
```

```
      (nr[id_s] < nr[u] < nr[id_s]+w_poddrzewie[id_s]))          // jest w poddrzewie id_s
    then W_tył(u, id_s);
}

Przeszukiwanie w tył::
  for each węzeł v do s[v] := 0;                                 // s[v] = 0 oznacza, że nie v był odwiedzony
  for i := 1 to |V| do
    if s[wezly[v]] = 0 then W_tył(wezly[v], wezly[v]);
```

Uważny czytelnik dostrzeże, że złożoność czasowa powyższego algorytmu wynosi O(|*V*|+|*E*|) i jest on liniowy ze względu na rozmiar grafu. Ten algorytm bardzo dobrze zachowuje się dla grafów z bardzo dużą liczbą węzłów i liniową względem niej liczbą krawędzi.

W następnych rozdziałach przekonamy się, że trzy przedstawione problemy mają silny związek z Internetem.

4 AUTORSKA, BARDZO KRÓTKA HISTORIA INTERNETU

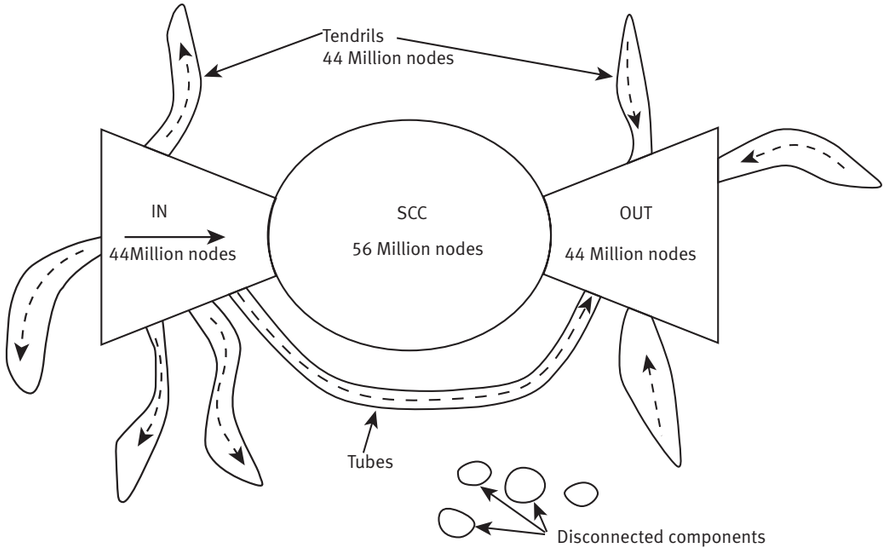
Internet to ogólnoświatowa sieć komputerowa, która z punktu widzenia przeciętnego użytkownika jest jednorodną siecią logiczną, w której węzły (komputery) są jednoznacznie identyfikowane przez swój adres. Początki Internetu sięgają końca lat sześćdziesiątych XX wieku, kiedy to powstaje sieć ARPANET łącząca cztery jednostki naukowe w Kalifornii i w stanie Utah. Sieć ARPANET była poletkiem doświadczalnym dla powstania i rozwoju powszechnie dziś używanej rodziny protokołów komunikacyjnych w sieci Internet – TCP/IP. W roku 1975 powstaje firma Microsoft, której rozwiązania przyczyniły się i nadal przyczyniają do upowszechniania świata komputerów wśród zwykłych ludzi. W roku 1976 pojawił się system operacyjny Unix – dziadek systemów operacyjnych z rodziny Linux. W tym też roku królowa Elżbieta wysłała swój pierwszy e-mail. W roku 1979 powstaje USENET – protoplasta powszechnych dziś sieci społecznościowych. W roku 1981 pojawia się komputer osobisty IBM PC. Bez komputerów osobistych, które możemy postawić na biurku i korzystać z ich dobrodziejstw w domu, globalizacja Internetu nie byłaby możliwa. W roku 1982, wraz z ustaleniem protokołów TCP/IP, nazwa Internet (z wielkiej litery) zaczyna się rozpowszechniać. W roku 1987 liczba hostów w Internecie przekracza 10 000, w roku 1989 przekroczona zostaje granica 100 000 hostów, w roku 1992 liczba hostów sięga miliona. Według Internet System Consortium, w lipcu 2010 roku liczba hostów wynosiła 768 913 036.

W roku 1990 sieć ARPANET przechodzi do historii. Rok 1991 to narodziny sieci WWW – ogólnoświatowej sieci powiązanych stron zawierających różnorodne treści multimedialne. Sieć jest rozwijana z wykorzystaniem Internetu, a strony w sieci WWW są identyfikowane poprzez tzw. adresy URL *(Uniform Resource Locator)*. W roku 1993 pojawia się MOSAIC – pierwsza graficzna przeglądarka stron WWW, a niedługo po niej Netscape. Rok 1994 to narodziny Yahoo. Największy obecnie gracz w Internecie to firma Google, która powstała w roku 1998. W roku 2004 startuje Facebook – najpopularniejsza obecnie w świecie sieć społecznościowa.
Z polskiej historii odnotujmy udostępnienie w roku 2000 komunikatora Gadu-Gadu, powstanie w roku 2001 serwisu gier Kurnik, a w roku 2006 – portalu społecznościowego Nasza-Klasa.

Czym charakteryzuje się Internet, a z punktu widzenia zwykłego użytkownika – sieć WWW? Jakie problemy algorytmiczne należy rozwiązać, żeby zasoby sieci stały się łatwo i szybko dostępne? Próbujemy odpowiedzieć na te pytania w następnym punkcie.

5 CHARAKTERYSTYKA INTERNETU

Zanim zajmiemy się problemami algorytmicznymi związanymi z Internetem, zastanówmy się, jak można opisać tę sieć. Skupimy się tutaj bardziej na sieci WWW, z której na co dzień korzystamy. Sieć WWW możemy przedstawić jako graf skierowany, którego węzłami są strony internetowe, natomiast dwie strony *V*, *W* są połączone krawędzią, jeśli na stronie *V* istnieje dowiązanie (link) do strony *W*. W roku 2000 grupa naukowców z AltaVista Company, IBM Almaden Research Center i Compaq Systems Research Center, opisała strukturę sieci WWW na podstawie danych zebranych przez **szperacze** (ang. *crawlers*) firmy Alta Vista [2]. Szperacze odwiedziły ponad 200 milionów stron, przechodząc po więcej niż półtora miliarda łączach. W wyniku analizy ustalono, że graf sieci WWW ma strukturę przedstawioną na rysunku 2.



Rysunek 2.
Sieć WWW [źródło: 2]

Co możemy wyczytać z tego rysunku? Gdybyśmy zapomnieli o orientacjach krawędzi, to prawie wszystkie strony należałyby do jednej spójnej składowej. Sieć jednak jest grafem skierowanym. Składają się na nią cztery główne części. Największa jest część środkowa SCC – silnie spójna składowa. W składowej SCC, z każdej strony możemy dojść do każdej innej, wykonując być może wiele kliknięć. Z dowolnej strony części IN można przejść przez SCC do dowolnej strony części OUT. Z części IN wychodzą tzw. **wici** (ang. *tendrils*), czyli strony, które są osiągalne z pewnych stron w IN. Podobnie wici prowadzą do OUT – strony, z których można dostać się do stron w OUT. Część stron z OUT jest bezpośrednio osiągalna ze stron IN za pomocą tzw. **rur** (ang. *tubes*). Po zanalizowaniu próbki sieci, Broder i inni doszli do ważnych wniosków [2]:

- sieć WWW jest skierowanym grafem rzadkim, tzn. liczba jej krawędzi liniowo zależy od liczby węzłów (badana sieć zawierała $2{,}7\times10^8$ stron (adresów URL) i $2{,}13\times10^9$ dowiązań,
- odległości w sieci są małe; w sieci zbadanej przez Brodera średnia długość ścieżki skierowanej wynosiła 16 – tyle kliknięć wystarczy, żeby dotrzeć do pożądanej, osiągalnej strony; gdybyśmy zapomnieli o orientacji krawędzi, to średnia długość ścieżki wyniosłaby 6.

Obecnie rozmiar Internetu jest dużo większy niż w roku 2000. Jak już wspomnieliśmy, w lipcu 2010 roku zarejestrowano 768 913 036 hostów. 1 grudnia 2010 w użyciu było 3 300 466 944 adresów IP obejmujących

240 krajów. Szacuje się, że na dzień 1 stycznia 2011 liczba stron poindeksowanych przez firmę Google wynosiła ponad 21 miliardów. Jaki stąd wniosek? Projektując algorytmy dla sieci WWW musimy wiedzieć, że działamy na sieci o miliardach węzłów, która szczęśliwie jest rzadka. Nie jest to zaskakujące. W sieci są miliardy stron, ale na każdej stronie może być od kilku do kilkudziesięciu dowiązań.

Sieć WWW niesie z sobą wiele wyzwań algorytmicznych, zarówno ze względu na ogrom Internetu, ale także dlatego, że Internet jest w pełni rozproszony i bez centralnej administracji. Do podstawowych problemów algorytmicznych związanych z siecią WWW można zaliczyć:

- wyszukiwanie i składowanie stron (zawartości);
- indeksowanie stron i przetwarzanie zapytań;
- odpowiadanie na zapytania w sposób zadowalający użytkownika;
- zgłębianie i analiza sieci WWW.

W następnym punkcie przedstawimy algorytm PageRank, który służy do nadawania rang stronom w taki sposób, żeby po znalezieniu stron w odpowiedzi na pytanie użytkownika wymienić je w kolejności od najważniejszych do najmniej ważnych. Uważny czytelnik natychmiast spostrzeże trudności w tak sformułowanym problemie. Strona, która jest ważna dla jednego użytkownika, może być mniej ważna dla innego. Czy istnieje jakiś w miarę obiektywny ranking stron, który byłby akceptowalny przez większość użytkowników Internetu? Takie pytanie zadali sobie twórcy firmy Google, Sergey Brin i Larry Page. Odpowiedzią na nie był algorytm PageRank, który przyczynił się do powstania i rozwoju Google.

6 ALGORYTM PAGERANK

Przedstawiony poniżej opis algorytmu PageRank został przygotowany już wcześniej i ukazał się w czasopiśmie „Delta” nr 8/2008 pod tytułem *Miara ważności* [3].

Każdy z nas choć raz w życiu użył wyszukiwarki Google. Czy zastanawialiśmy się, dlaczego wyszukiwarka Google podaje adresy stron będących odpowiedzią na zapytanie właśnie w takiej, a nie innej kolejności? Autor przeprowadził eksperyment i zanotował, co Google.pl dała w odpowiedzi na zapytanie matematyka. Oto 5 pierwszych adresów do stron, które autor otrzymał (6.01.2011, godz. 16.45):

1. www.matematyka.pl;
2. www.matematyka.pisz.pl;
3. pl.wikipedia.org/wiki/Matematyka;
4. matematyka.org;
5. www.math.edu.pl.

Przeglądarka Google podała, że wybrała je spośród 4 860 000 kandydatów. Dlaczego właśnie te strony uznano za najważniejsze? Jaka jest miara ważności (ranga) strony? Okazuje się, że podstawą analizy ważności stron w Google jest analiza połączeń w sieci WWW. Przypomnijmy, że sieć WWW jest grafem skierowanym, w którym węzłami są strony, a krawędzie odpowiadają dowiązaniom pomiędzy stronami – strona zawierająca adres internetowy innej strony jest początkiem krawędzi, a strona o danym adresie jest jej końcem. Czy można na podstawie samego grafu powiązań stron powiedzieć, które strony są ważniejsze, a które mniej?

Strona istotna, to strona interesująca. Strona interesująca, to taka, do której łatwo dotrzeć, ponieważ wiele innych stron na nią wskazuje. Na dodatek wiele spośród stron wskazujących na ważną stronę też jest ważnych i pasjonujących itd. Sergey Brin i Larry Page wyrazili to matematycznie w następujący sposób.

Załóżmy, że mamy n stron S_1, S_2, ..., S_n. Niech $w(S_i)$ będzie liczbą rzeczywistą dodatnią mierzącą ważność strony S_i. Wówczas określamy:

$$w(S_i)=\sum_{S_j \in We(S_i)} \frac{w(S_j)}{|Wy(S_j)|},$$

gdzie $We(S_i)$ to zbiór stron zawierających adres strony S_i (czyli zbiór początków krawędzi prowadzących do S_i), zaś $|Wy(S_j)|$ jest liczbą odnośników wychodzących ze strony S_j, czyli krawędzi prowadzących do stron ze zbioru $Wy(S_j)$. Wzór ten oznacza, że ważność strony mierzy się ważnością stron na nią wskazujących. Jeżeli przyjmiemy na początek, że wszystkie strony są jednakowo ważne i dla każdej z nich $w(S_i)=1/n$, gdzie n to liczba wszystkich stron, to ważność stron można obliczyć za pomocą następującej metody iteracyjnej (proces 1):

$$w_{k+1}(S_i)=\sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|Wy(S_j)|}.$$

Na powyższy proces można spojrzeć jak na błądzenie losowe po sieci WWW. Rozpoczynamy z dowolnej strony, a następnie z każdej oglądanej strony ruszamy losowo do jednej ze stron, do których adresy umieszczono na tej stronie. Jeżeli nasz proces będziemy powtarzali bardzo długo, to pewne strony będziemy oglądali częściej niż inne. Strony oglądane częściej są ważniejsze.

Niestety, może się zdarzyć, że przemieszczając się po stronach natrafimy na takie, z których nie ma wyjścia, np. strony zawierające zdjęcia. W takim przypadku zakładamy, że w następnym kroku wybierzemy losowo dowolną ze stron w Internecie. Teraz nasz proces iteracyjny ma następującą postać (proces 2):

$$w_{k+1}(S_i) = \sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|Wy(S_j)|} + \sum_{S_j \in K} \frac{w_k(S_j)}{n},$$

gdzie K oznacza zbiór wszystkich stron końcowych, czyli takich, które nie zawierają dowiązań do żadnych innych stron. Jesteśmy już blisko algorytmu (a raczej jego idei) firmy Google służącego do ustalania ważności stron.

Obserwując zachowanie użytkowników Internetu, Brin i Page zauważyli, że czasami porzucają oni bieżące przeszukiwanie i rozpoczynają nowe od (losowo) wybranej strony. Dlatego zmodyfikowali swój proces iteracyjny wprowadzając parametr α o wartości z przedziału (0,1). W każdej iteracji z prawdopodobieństwem α aktualne przeszukiwanie jest kontynuowane, natomiast z prawdopodobieństwem $(1 - \alpha)$ rozpoczynane jest nowe przeszukiwanie. Ostatecznie postać każdej iteracji jest następująca (proces 3):

$$w_{k+1}(S_i) = \alpha\left(\sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|S_j|} + \sum_{S_j \in K} \frac{w_k(S_j)}{n}\right) + (1-\alpha)\sum_{j=1}^{n} \frac{w_k(S_j)}{n}.$$

Bardziej doświadczeni czytelnicy pewnie już spostrzegli, że nasz proces iteracyjny jest procesem stochastycznym zbieżnym do stacjonarnego rozkładu prawdopodobieństwa. Celem kolejnych modyfikacji procesu było zapewnienie właśnie takiej zbieżności. Końcowy rozkład prawdopodobieństwa odpowiada ważności stron.

Zastanówmy się teraz, jaki jest związek opisanego procesu iteracyjnego z mnożeniem macierzy przez wektor. Sieć WWW na potrzeby tego procesu możemy opisać za pomocą macierzy H o rozmiarach n x n i zdefiniowanej następująco:

$$H[j,i] = \begin{cases} 1/|W(S_i)| & \text{jeśli istnieje dowiązanie z } S_i \text{ do } S_j \\ 0 & \text{w przeciwnym razie.} \end{cases}$$

Oznaczmy przez $w_k[1..n]$ wektor ważności stron po k-tej iteracji, tzn. $w_k[i]$ jest ważnością strony S_i po k-tej iteracji. Zgodnie z opisaną konwencją, na początku wszystkie strony są jednakowo ważne. Zatem dla każdej strony S_i mamy $w_0[i] = 1/n$. Przy takiej notacji proces 1 możemy zapisać następująco:

$$w_{k+1}=Hw_k.$$

Jest tak dlatego, ponieważ *i*-ty wiersz macierzy *H* zawiera odwrotności liczby dowiązań na stronach, z których prowadzą łącza do strony S_i.

Proces 2 uwzględnia fakt, że są strony niezawierające żadnych dowiązań. W macierzy *H* kolumny odpowiadające takim stronom zawierają same 0. Z takich stron do dalszego przeszukiwania wybieramy dowolną stronę z prawdopodobieństwem $1/n$. W notacji macierzowej wystarczy zatem zastąpić w macierzy *H* wszystkie kolumny zawierające same 0 przez kolumny, w których na każdej pozycji jest wartość $1/n$. Oznaczmy tak powstałą macierz przez *S*. Wówczas proces 2 ma postać:

$$w_{k+1}=Sw_k.$$

Proces 3 to zmodyfikowany proces 2, w którym z prawdopodobieństwem α kontynuujemy przeszukiwanie sieci z danej strony, a z prawdopodobieństwem $(1 - \alpha)$ przechodzimy do losowej strony. Oznacza to następującą modyfikację macierzy *S*: każdą pozycję macierzy *S* mnożymy przez α i dodajemy do tego $(1 - \alpha)/n$. Tak otrzymaną macierz oznaczmy przez *G*. Tak więc iteracyjny proces obliczania rang stron w macierzowej reprezentacji zapisuje się następująco:

$$w_{k+1}=Gw_k.$$

To jest zwykłe mnożenie macierzy przez wektor. Wiemy już, że macierz *G* reprezentuje sieć składającą się z ponad 21 miliardów węzłów. Niestety, chociaż sieć jest rzadka (zawiera kilkaset miliardów krawędzi), to macierz *G* nie jest rzadka – do każdego elementu macierzy *S* dodaliśmy bowiem $(1 - \alpha)/n$. Szczęśliwie, używając przekształceń algebraicznych możemy każdą iterację zapisać jako mnożenie rzadkiej macierzy *H* przez wektor plus dwie modyfikacje otrzymanego wektora:

1. $w_{k+1}=Hw_k$,
2. pomnóż każdy element wektora w_{k+1} przez α: $w_{k+1}=\alpha w_{k+1}$,
3. oblicz wartość β_k równą $1/n$ sumy: $(1 -\alpha)$ oraz sumy tych elementów wektora w_k, które odpowiadają stronom niezawierającym dowiązań do innych stron, pomnożonej przez α:

   $$\beta k= \frac{1}{n}\left((1 - \alpha) + (\Sigma_{S_i \in k}\, \alpha\, w_k[i])\right),$$

4. do każdego elementu wektora w_{k+1} dodaj β_k.

Jest jeszcze wiele pytań, które pozostają bez odpowiedzi. Jak szybko powyższy proces zbiega do końcowego rozkładu? Jak dobierać parametr α? Parametr α dobrany w Google wynosi 0.85. Macierz *G* z takim parametrem nosi nazwę macierzy Google. Okazuje się, że w tym przypadku wystarczy kilkanaście iteracji, aby policzyć wektor rang. Ale każda iteracja to setki miliardów operacji arytmetycznych i przetwarzanie olbrzymiego grafu. W jaki sposób możemy przyśpieszyć proces obliczeń? Zauważmy, że mnożąc macierz przez wektor można niezależnie pomnożyć każdy wiersz macierzy przez ten wektor. A gdyby zrobić to równolegle? To już jednak inna historia.

Zainteresowanych dokładną analizą algorytmu PageRank odsyłam do wspaniałej książki autorstwa Amy N. Langville i Carla D. Meyera, *Google's PageRank and Beyond: The Science of Search Engine Rankings* [4].

PODSUMOWANIE

Przedstawiliśmy trzy problemy algorytmiczne, które znajdują bezpośrednie zastosowanie w przetwarzaniu danych w Internecie. Algorytm obliczania silnie spójnych składowych może posłużyć do analizy sieci WWW. Mnożenie macierzy przez wektor wykorzystuje się do obliczania rang stron w sieci WWW tylko na podstawie struktury sieci. Co zadziwiające, również Lider znajduje zastosowanie. Wyobraźmy sobie ruter, który steruje ruchem pakietów w bardzo ruchliwej sieci. Administratorów sieci często interesuje, jakie pakiety generują największy ruch, a dokładniej, pod które adresy jest dostarczana największa liczba komunikatów. Jeśli pakietów są miliardy, to nie jesteśmy w stanie zapamiętać danych o wszystkich z nich, a następnie przetworzyć je. Algorytm przedstawiony w tym opracowaniu jest tak zwanym algorytmem strumieniowym – na bieżąco przetwarzany jest strumień danych i nie ma możliwości powrotu do historii. Jak się okazało można w ten sposób wskazać adres, który najbardziej obciąża sieć, jeśli tylko generuje on ponad połowę ruchu. Niewielka modyfikacja przedstawionego algorytmu umożliwia wskazywanie pakietów, z których każdy generuje np. co najmniej jedną dziesiątą ruchu.

Jaki morał płynie z naszych rozważań? Warto się uczyć algorytmiki. Nawet wydawałoby się dziwne problemy pojawiają się w tak gorących zastosowaniach, jak przetwarzanie Internetu. Niestety nie każdy algorytm, który wydaje się dobry w konwencjonalnych zastosowaniach (np. działa w czasie wielomianowym) nadaje się do przetwarzania takiego ogromu danych, jakie niesie ze sobą Internet. Nawet algorytmy liniowe mogą wymagać wielodniowych obliczeń. W przetwarzaniu danych w Internecie może pomóc równoległość, algorytmy strumieniowe i statystyczne. Algorytm PageRank pokazuje też, że bardzo dobre efekty z punktu widzenia użytkowników dają algorytmy, które naśladują ich naturalne zachowanie. Zachęcam czytelników do poszukiwania naturalnych problemów związanych z Internetem i rozwiązywania ich w stylu algorytmu PageRank.

LITERATURA

1. Banachowski L., Diks K., Rytter W., *Algorytmy i struktury danych*, WNT, Warszawa 2006
2. Broder A., Kumar R., Maghoul F., Raghavan P., Rajagopalan S., Stata R., Tomkins A., Wiener J.L., *Graph structure in the Web*, „Computer Networks" 2000, Vol. 33 (1-6), s. 309-320
3. Diks K., *Miara ważności*, „Delta" 8/2008
4. Langville A.N., Meyer C.D., *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton 2006

Jak wnioskują maszyny

Andrzej Szałas

Instytut Informatyki, Uniwersytet Warszawski

andrzej.szalas@mimuw.edu.pl

Streszczenie

Wykład jest poświęcony wprowadzeniu do logiki z perspektywy jej zastosowań w informatyce i sztucznej inteligencji. Poruszane treści obejmują następujące zagadnienia:

- wprowadzenie do logiki jako nauki o modelowaniu świata rzeczywistego i wnioskowaniu o nim;
- klasyczny rachunek zdań (składnia, semantyka spójników logicznych);
- odniesienie do zbiorów i użycie diagramów Venna, jako metody wnioskowania;
- wyszukiwanie a wnioskowanie na przykładzie wyszukiwarki internetowej Google;
- automatyczne wnioskowanie (informacja o metodzie rezolucji dla rachunku zdań).

Wszystkie zagadnienia są ilustrowane przykładami, w tym związanymi z robotyką i sztuczną inteligencją.

Od słuchaczy nie wymaga się żadnej wstępnej wiedzy z zakresu logiki i matematyki.

Spis treści

1 WPROWADZENIE

Działalność człowieka – począwszy od tej codziennej po najbardziej zaawansowane badania i konstrukcje – zaczyna się od rozpoznawania odpowiedniego zestawu zjawisk, a następnie ich modelowania, po to by uzyskany i sprawdzony model później wykorzystywać. Gdy poruszamy się po własnym mieszkaniu, posługujemy się jego modelem stworzonym w czasie rozpoznawania tegoż mieszkania. Model ten mamy zwykle w głowie, niemniej jednak w porównaniu z rzeczywistością jest on bardzo uproszczony. Nie bierze pod uwagę przepływu cząstek powietrza, zachowania elektronów w poszczególnych cząstkach występujących w stropie i ścianach, nie dbamy o skomplikowane procesy przepływu wody w rurach wodociągowych itd. Prostota schematu pozwala jednak na skuteczne wnioskowanie, poruszanie się po pomieszczeniach i działanie. W informatyce też dba się o to, by dla danego zjawiska czy problemu obliczeniowego wybrać możliwie jak najprostszy, ale zarazem skuteczny model.

Wyobraźmy sobie zadanie polegające na opracowaniu bardzo prostego robota przemysłowego, który „obserwuje" taśmę produkcyjną i którego zadaniem jest przestawianie z niej przedmiotów zielonych na taśmę znajdującą się po lewej stronie, a czerwonych – na taśmę znajdującą się po prawej stronie. Tworzymy więc model – trzy taśmy produkcyjne. Nad środkową taśmą czuwa robot wyposażony w:

- czujniki rozpoznające kolor zielony i czerwony,
- chwytaki służące do chwytania przedmiotów i przestawiania ich na lewą lub prawą taśmę.

Mając ten model możemy teraz opisać działanie robota dwoma prostymi regułami:

- jeśli obserwowany obiekt jest zielony, to przenieś go na taśmę z lewej strony,
- jeśli obserwowany obiekt jest czerwony, to przenieś go na taśmę z prawej strony.

Powyższe reguły nie gwarantują przeniesienia wszystkich przedmiotów zielonych na lewą stronę, a czerwonych na prawą, bo zależy to od prędkości taśm, akcji rozpoznawania koloru i akcji przenoszenia przedmiotów. Zaczynamy jednak już mieć do czynienia z logiką. Przytoczone dwie reguły odzwierciedlają bardzo proste wnioskowanie. W opisanym przypadku niekoniecznie skuteczne, ale za to skuteczne i wystarczające w innym modelu. Mianowicie, jeśli założymy, że robot nie myli się w rozpoznawaniu kolorów i niezawodnie przenosi obiekty oraz że środkowa taśma zatrzymuje się na czas rozpoznawania koloru i przenoszenia przedmiotu przez robota, a na dodatek zatrzymuje każdy przedmiot w zasięgu czujników i chwytaków robota – to takie proste wnioskowanie będzie skuteczne i można je formalnie wykazać. Daje to wiedzę o warunkach, jakie powinno spełniać środowisko robota, by ten mógł działać skutecznie wykorzystując swoje możliwości.

Możemy więc zauważyć, że skuteczność wnioskowań zależy od przyjętego modelu rzeczywistości. I znów zauważmy, że omawiane modele biorą pod uwagę jedynie to, co niezbędne dla skutecznego rozwiązania problemu, zaniedbując to, co z punktu widzenia tej skuteczności nie jest wymagane.

Aby modelować rzeczywistość zwykle zaczynamy od identyfikacji przedmiotów (**obiektów**), rodzajów obiektów (**pojęć**), ich cech (**atrybutów**) i związków między nimi. Tak postępuje się np. w projektowaniu relacyjnych baz danych (jak Access, Oracle, MySQL itp.). Każda baza danych jest modelem pewnej rzeczywistości, a wyniki zapytań kierowanych do baz danych – uzyskanymi informacjami, prawdziwymi w tej rzeczywistości. Podobnie postępuje się w wielu innych obszarach informatyki, w tym choćby w projektowaniu obiektowym, niesłychanie ważnym we współczesnych systemach.

Załóżmy teraz, że zidentyfikowaliśmy dwa rodzaje owoców: *cytryny* i *figi*. W zależności od potrzeb wynikających z rozwiązywanego zadania możemy określać ich atrybuty, jak rodzaj, kolor, smak itp. Powstaje w ten sposób pewna baza danych, zilustrowana w tabeli 1 (oczywiście różne rodzaje obiektów mogą mieć różne atrybuty, co prowadzi do wielu tabel – na przykład, gdyby wśród obiektów występowali ludzie, atrybut *smak* mógłby mieć w ich przypadku mało sensu).

Tabela 1.
Przykładowe obiekty i ich atrybuty

	atrybuty		
obiekt	*rodzaj*	*smak*	*kolor*
o1	*cytryna*	*kwaśny*	*żółty*
o2	*figa*	*słodki*	*brązowy*
o3	*cytryna*	*kwaśny*	*zielony*

Tabelę 1 możemy przekształcić w tabelę 2, w której atrybutami stają się wartości rodzaju, smaku, koloru, zaś wartościami 0 i 1, gdzie 0 oznacza fałsz, zaś 1 – prawdę:

Tabela 2.
Obiekty i ich atrybuty jako wartości

	atrybuty						
obiekt	*cytryna*	*figa*	*kwaśny*	*słodki*	*żółty*	*brązowy*	*zielony*
o1	1	0	1	0	1	0	0
o2	0	1	0	1	0	1	0
o3	1	0	1	0	0	0	1

Zapytanie *cytryna* AND *żółty* wybierze w wyniku obiekt o1, gdyż tylko on jest jednocześnie *cytryną* i jest *żółty*. Zapytanie *cytryna* OR *żółty* wybierze obiekty o1, o3, bowiem wybieramy będące *cytryną* lub mające kolor *żółty*. Natomiast zapytanie ¬*cytryna* wybierze obiekt o2, bowiem symbol '¬' oznacza negację (zaprzeczenie).

Możemy też określić związki między zidentyfikowanymi pojęciami, np:

jeśli dany obiekt jest *cytryną*, to jest *kwaśny* i nie jest *figą*
jeśli dany obiekt jest *żółty*, to jest *cytryną*
jeśli dany obiekt jest *figą*, to nie jest *cytryną*

Znów mamy do czynienia z pewnymi wyrażeniami typu „jeśli ..., to ...", stosowanymi we wnioskowaniu o rzeczywistości złożonej z *cytryn* i *fig*. Dodatkowo takie związki często mogą służyć do uproszczenia tabeli. Na przykład, mając powyższe zależności można opuścić kolumnę *figa*, gdyż jest ona zdefiniowana jako ¬*cytryna* (dlaczego?), a więc występujące w niej wartości można obliczyć na podstawie wartości z kolumny *cytryna*.

Badaniem metod wnioskowania zajmuje się **logika** (od greckiego słowa *logos*, oznaczającego rozum, słowo, myśl). Z jednej strony analizuje się w niej poprawność wnioskowań, a z drugiej strony – dostarcza metod i algorytmów wnioskowania. Korzenie logiki sięgają starożytnej Grecji, ale też Chin czy Indii. Odgrywała istotną rolę w średniowieczu, burzliwy jej rozwój datuje się od końca XIX wieku. George Boole, Charles Sanders Peirce, John Venn, potem Bertrand Russell czy Gottlob Frege są wybitnymi logikami z tego okresu. Pojęcie **obliczalności** jest też ściśle związane z badaniami logicznymi dotyczącymi rozstrzygalności teorii matematycznych, czyli szukania metod algorytmicznych dla automatycznego znajdowania dowodów twierdzeń tych teorii. Wybitnymi przedstawicielami tego kierunku są: Richard Dedekind, Giuseppe Peano, David Hilbert, Arend Heyting, Ernst Zermelo, John von Neumann, Kurt Gödel czy

Alfred Tarski. Pierwsze matematyczne modele maszyn matematycznych zawdzięczamy kontynuacji prac tych logików, prowadzonych przez Emile'a Posta, Alonza Churcha (prekursorzy języków funkcyjnych), jak również Stephena C. Kleenego, Alana M. Turinga czy Claude'a E. Shannona (prekursorzy języków imperatywnych).

2 WYSZUKIWANIE I WNIOSKOWANIE

Można napisać, że logika jest we wnioskowaniu wszechobecna, gdyż fałsz, prawda, wnioskowanie, model, pojęcie, związek (relacja), reguła czy spójniki (AND, OR, ¬) są podstawowymi konceptami rozważanymi w logice. Zaczęliśmy od baz danych i wyszukiwania obiektów mających interesujące nas właściwości. Przejdźmy teraz do wyszukiwarek internetowych. Internet jest ogromną bazą danych. Zasadniczą różnicą w porównaniu z tradycyjnymi bazami danych, zorganizowanymi w bardzo uporządkowany sposób, jest w Internecie ogromna różnorodność zasobów i brak ich jednolitej struktury.

Załóżmy, że interesującymi nas obiektami z Internetu są strony WWW. Wpisując w okienko wyszukiwarki Google zestaw słów, pytamy o te strony, na których występują wszystkie wypisane słowa kluczowe. Jest to tzw. **wyszukiwanie AND**. W języku polskim „and" znaczy „i". W logice taki spójnik nazywamy **koniunkcją** i często w literaturze oznaczamy symbolem ∧.

Aby koniunkcja *p* AND *q* była prawdziwa, prawdziwe muszą być oba zdania składowe: zdanie *p* i zdanie *q*. Jeśli np. wpiszemy dwa słowa: *logika informatyka,* to z punktu widzenia wyszukiwarki Google oznacza to wpisanie wyrażenia *logika* AND *informatyka*, czyli wyszukanie stron (naszych obiektów), na których występuje słowo *logika* i słowo *informatyka*. Tak naprawdę nie zawsze pojawią się oba słowa, co odbiega od logicznego rozumienia koniunkcji. Aby mieć „prawdziwą" koniunkcję powinniśmy wpisać wyrażenie + *logika* + *informatyka* (operator + umieszczony przed danym słowem oznacza, że musi ono wystąpić na wyszukanej stronie).

Co jeszcze pojawia się w Google? Twórcy tej wyszukiwarki oferują też **wyszukiwanie OR**. W języku polskim „or" to „lub", czyli logiczny spójnik **alternatywy**, w literaturze często oznaczany symbolem ∨. Aby alternatywa *p* OR *q* była prawdziwa, prawdziwe musi być co najmniej jedno ze zdań składowych: zdanie *p* lub zdanie *q* (lub oba te zdania). Na przykład wpisanie w Google wyrażenia *logika* OR *informatyka* spowoduje wyszukanie stron na których występuje słowo *logika* lub słowo *informatyka* lub oba te słowa. Spójnik OR wiąże silniej niż spójnik AND[5]. Oznacza to, że wyrażenie

lekcja informatyka OR *logika*

Google rozumie jako

lekcja AND (*informatyka* OR *logika*)

a nie jako (*lekcja* AND *informatyka*) OR *logika*. Wyszukane zostaną więc strony, na których pojawia się słowo *lekcja* oraz co najmniej jedno ze słów *informatyka* lub *logika*.

I mamy jeszcze operator negacji ¬, który umieszczony przed słowem oznacza, że *nie* może ono wystąpić na wyszukanej stronie. W logice spójnik „*nie*" nazywamy **negacją** i oznaczamy często symbolem ¬ (a w Google symbolem –). Negacja ¬*p* jest prawdziwa, gdy zdanie *p nie* jest prawdziwe. Na przykład wpisanie do wyszukiwarki wyrażenia – *logika* spowoduje wyszukanie tych stron WWW, na których nie występuje słowo *logika*. Czyli Google posługuje się logiką, interpretując wyrażenia logiczne i wyszukując zgodnie z nimi interesujące nas zasoby.

[5] Ta konwencja jest charakterystyczna dla Google. Tradycyjnie koniunkcja wiąże silniej niż alternatywa.

Ta logika to uproszczona wersja **klasycznego rachunku zdań**, zajmującego się w swoim podstawowym wariancie spójnikami logicznymi negacji, alternatywy, koniunkcji oraz **implikacji** i **równoważności**. Implikacja to wyrażenie postaci „*jeśli p to q*”, zaś równoważność to wyrażenie postaci „*p wtedy i tylko wtedy, gdy q*”. O tym dokładniej poniżej.

Zadania

1. Przetłumacz na język logiki zdanie określające zbiór obiektów *czerwonych* i *zielonych*, ale takich, które nie są *słodkie*.

   Przykładowe rozwiązanie: *czerwony* AND *zielony* AND -*słodki*.

   Wpisz powyższe wyrażenie do przeglądarki Google i porównaj je z wynikami wyszukiwania dla wyrażenia +*czerwony* AND +*zielony* AND -*słodki*. Zauważ dużą różnicę w liczbie wyszukanych stron. Zinterpretuj tę różnicę.
2. Napisz wyrażenie wyszukujące w Google strony WWW o *restauracjach* lub *barach* na Mazurach, ale nie zawierających słowa *Rzym*.

   Przykładowe rozwiązanie: +Mazury -Rzym restauracja OR *bar*.

   Jakie mogą być inne rozwiązania?

I do samodzielnego rozwiązania:

3. Przetłumacz na język logiki zdanie określające jeden z dni roboczych w tygodniu. Jakie wyrażenie określi wszystkie dni robocze?
4. Napisz wyrażenie wyszukujące w Google strony o prowadzonych kierunkach studiów z informatyki lub biologii, ale nie z matematyki i nie z filologii klasycznej.

3 KLASYCZNY RACHUNEK ZDAŃ

Klasyczny rachunek zdań zajmuje się badaniem prawdziwości zdań złożonych na podstawie zdań składowych i w konsekwencji – badaniem poprawności wnioskowania.

Aby wprowadzić rachunek zdań wprowadza się **zmienne zdaniowe** reprezentujące wartości logiczne *prawda*, *fałsz*, a zarazem zbiory obiektów mających cechy opisywane tymi zmiennymi (w tym ujęciu zmienne zdaniowe odpowiadają cechom, czyli atrybutom obiektów). Bardziej złożone wyrażenia (zwane **formułami**) uzyskujemy stosując **spójniki logiczne** negacji, koniunkcji, alternatywy, implikacji i równoważności. Czasem wprowadza się też inne spójniki. Tak naprawdę wszystkie możliwe spójniki można zdefiniować przy pomocy np. negacji i koniunkcji, jednak przyjęty przez nas zestaw spójników, choć z tego punktu widzenia nadmiarowy, jest z jednej strony prosty i naturalny, a z drugiej wystarczający w wielu typowych zastosowaniach.

Znaczenie (**semantykę**) spójników logicznych podaje się często przy pomocy tablic logicznych, w których w kolumnach podaje się wartości poszczególnych wyrażeń. Przyjmujemy, że wartościami tymi mogą być jedynie 0, 1; 0 – to **fałsz**, a 1 – to **prawda**, patrz tabele 3 i 4.

Tabela 3.
Tablica dla negacji:

p	$\neg p$
0	1
1	0

Tabela 4.
Tablica dla koniunkcji, alternatywy, implikacji i równoważności

		koniunkcja	alternatywa	implikacja	równoważność
p	q	p AND q	p OR q	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Na tej podstawie mamy bardzo skuteczny mechanizm sprawdzania poprawności wnioskowania dla formuł z niewielką liczbą zmiennych zdaniowych. Mianowicie konstruujemy tablice logiczne, w których w pierwszych kolumnach są zmienne zdaniowe, zaś w kolejnych – wyrażenia występujące w badanej formule ułożone w ten sposób, by wartość danego wyrażenia można było policzyć na podstawie wcześniej występujących wyrażeń. Wiersze w tabeli wypełnia się najpierw wszystkimi możliwymi układami wartości logicznych, a następnie wylicza wartości wyrażeń w kolejnych kolumnach.

Formuła nazywa się **tautologią**, jeśli przyjmuje wartość 1 (prawda) niezależnie od wartości wchodzących w jej skład zmiennych zdaniowych. Jest ona **spełnialna**, gdy przyjmuje wartość 1 co najmniej dla jednej kombinacji wartości zmiennych zdaniowych. Jeśli zawsze przyjmuje wartość 0 (fałsz), jest nazywana **kontrtautologią**.

Dla przykładu sprawdźmy jakie wartości przyjmuje formuła $\neg(p \text{ OR } q) \Rightarrow \neg p$ (tab. 5).

Tabela 5.
Tablica logiczna dla przykładowej formuły

p	q	p OR q	$\neg(p \text{ OR } q)$	$\neg p$	$\neg(p \text{ OR } q) \Rightarrow \neg p$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	0	0	1
1	1	1	0	0	1

Badana formuła jest zawsze prawdziwa, jest więc tautologią (każda tautologia jest również spełnialna). Formuły występujące we wcześniejszych kolumnach są spełnialne, ale nie są tautologiami.

Sprawdźmy jeszcze **zasadę dowodzenia przez doprowadzanie do sprzeczności**. Została ona odkryta już przed niemal 2,5 tysiącami lat (przypisuje się ją Zenonowi z Elei). Jednak jest ważna współcześnie. Stosuje się ją w matematyce, ale też odgrywa zasadniczą rolę we wnioskowaniu z baz danych wiedzy. Będziemy z niej korzystać przy omawianiu metody rezolucji, najpopularniejszej współczesnej metody automatycznego wnioskowania. Zasada ta mówi, że w celu wykazania implikacji $p \Rightarrow q$, zaprzeczamy q i wykazujemy, że prowadzi to do fałszu. Innymi słowy, chcemy wykazać formułę:

$$(p \Rightarrow q) \Leftrightarrow ((p \text{ AND } \neg q) \Rightarrow 0).$$

Konstruujemy tablicę logiczną jak w tabeli 6.

Tabela 6.
Tablica logiczna dla formuły uzasadniającej zasadę dowodzenia przez sprzeczność

p	q	$p \Rightarrow q$	$\neg q$	p AND $\neg q$	(p AND $\neg q$) $\Rightarrow 0$	($p \Rightarrow q$) $\Leftrightarrow$ ((p AND $\neg q$) $\Rightarrow 0$)
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	1	0	0	1	1

Badana formuła jest więc rzeczywiście tautologią.

W Internecie można znaleźć wiele appletów konstruujących tablice logiczne dla zadanych formuł. Przykładowy applet można znaleźć pod adresem:
http://highered.mcgraw-hill.com/sites/0072880082/student_view0/chapter1/interactive_demonstration_applet__truth_tables_.html.

Zadania (do samodzielnego rozwiązania)

1. Zbuduj tablice logiczne dla poniższych formuł i stwierdź, które z nich są tautologiami:
   a. ($p \Rightarrow q$) $\Leftrightarrow$ ($\neg p$ OR q)
   b. $\neg$(p OR q) $\Leftrightarrow$ ($\neg p$ AND $\neg q$).
2. Zapisz prawe strony równoważności tak, by były one tautologiami i sprawdź rozwiązanie używając tablic logicznych:
   a. $\neg$(p AND q) $\Leftrightarrow$.....
   b. $\neg$($p \Rightarrow q$) $\Leftrightarrow$.....

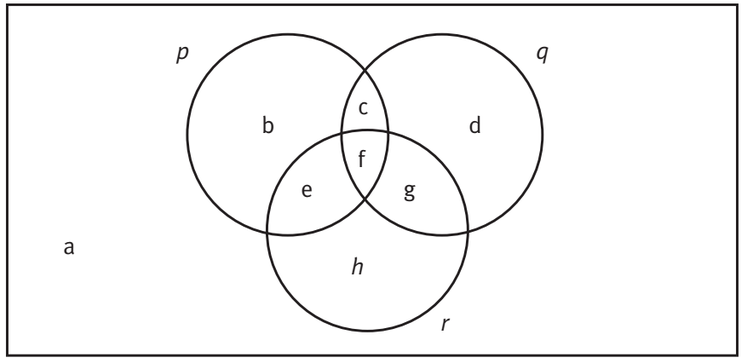
Dlaczego metoda tablic logicznych nie jest dobra dla większych zadań?
W praktycznych zastosowaniach często trzeba sprawdzać spełnialność formuł zawierających dużą liczbę zmiennych. Potrafi ona dochodzić do tysięcy. Załóżmy, że mamy formułę mającą 100 zmiennych. Tabela logiczna będzie więc miała 2^{100} wierszy (dlaczego?). Ile czasu spędziłby na obliczeniach bardzo szybki komputer, wykonujący powiedzmy 2^{34} operacji na sekundę (to więcej niż 10^{10} operacji na sekundę)? Aby wygenerować 2^{100} wierszy potrzebujemy więcej niż $2^{100}/2^{34}$ (= 2^{66}) sekund, czyli więcej niż $2^{66}/60$ minut. To więcej niż $2^{66}/2^6$ (= 2^{60}) minut i więcej niż $2^{60}/2^6$ (= 2^{54}) godzin i więcej niż $2^{54}/2^5$ (= 2^{49}) dób. To z kolei więcej niż 2^{40} lat (czyli więcej niż 10^{12} lat!). Wiek wszechświata szacuje się na 13-14 miliardów lat (czyli nie więcej niż 14^*10^9 lat).

4 DIAGRAMY VENNA

Diagramy Venna ilustrują zależności pomiędzy zbiorami obiektów. Ponieważ na zmienne logiczne możemy patrzeć jako na cechy obiektów, możemy też im przypisać zbiory obiektów mających te cechy. Diagramy Venna mogą ilustrować zbiory obiektów i posługując się nimi można znajdować zależności między rozważanymi pojęciami. Zbiory na tych diagramach reprezentujemy przy pomocy kół: z każdą rozważaną zmienną zdaniową związujemy koło. Jeśli nie mamy żadnych dodatkowych założeń, umieszczamy koła w ten sposób, by wydzieliły wszystkie możliwe zależności (obszary) na danej płaszczyźnie. Obszary te oznaczamy kolejnymi literami lub liczbami. Zbiór wszystkich obiektów jest reprezentowany przez prostokąt otaczający wszystkie koła.

Diagram dla trzech zmiennych zdaniowych p, q, r jest przedstawiony na rysunku 1. Na tym diagramie:

- zbiór obszarów przypisanych zmiennej p to {b, c, e, f}
- zbiór obszarów przypisanych zmiennej q to {c, d, f, g}
- zbiór obszarów przypisanych zmiennej r to {e, f, g, h}.



Rysunek 1.
Diagram Venna dla trzech zbiorów

Zauważmy, że obszar oznaczony przez a reprezentuje wszystkie obiekty leżące poza kołami reprezentującymi p, q oraz r.

Spójniki negacji, alternatywy i koniunkcji na diagramach Venna interpretujemy następująco:

- negacja formuły jest reprezentowana przez zbiór wszystkich obszarów niebędących obszarami reprezentującymi daną formułę; na przykład $\neg p$ reprezentujemy przez zbiór tych obszarów, które leżą poza p, czyli wykluczamy obszary b, c, e, f, otrzymując {a, d, g, h}
- alternatywa dwóch formuł jest reprezentowana przez zbiór obszarów reprezentujących pierwszą *lub* drugą formułę; na przykład p OR r reprezentujemy przez zbiór obszarów {b, c, e, f, g, h}
- koniunkcja dwóch formuł jest reprezentowana przez zbiór obszarów wspólnych dla pierwszej *i* drugiej formuły; na przykład p AND q reprezentujemy przez zbiór obszarów {c, f}.

Spójnik implikacji odzwierciedla zawieranie się zbiorów: $p \Rightarrow q$ oznacza, że zbiór obiektów reprezentujących p zawiera się w zbiorze obiektów reprezentujących q.

Spójnik równoważności odzwierciedla równość zbiorów: $p \Leftrightarrow q$ oznacza, że zbiór obiektów reprezentujących p jest taki sam, jak zbiór obiektów reprezentujących q.

W jaki sposób wnioskujemy stosując diagramy Venna
Diagramy Venna mogą być wykorzystane do znajdowania zależności między formułami.

Jako pierwszy przykład rozważmy formułę (p AND q) $\Rightarrow p$. Użyjmy poprzedniego diagramu (oczywiście koło reprezentujące r moglibyśmy pominąć, gdyż r nie występuje w naszej formule). Już zauważyliśmy, że p AND q reprezentujemy przez zbiór obszarów {c, f} oraz p reprezentujemy przez {b, c, e, f}. Oczywiście zbiór {c, f} zawiera się w zbiorze {b, c, e, f}, a więc badana implikacja jest prawdziwa.

Jako drugi przykład rozważmy formułę ((p AND q) OR r) $\Leftrightarrow$ ((p OR r) AND (q OR r)):

- formuła p AND q jest reprezentowana przez zbiór {c, f}
- formuła r jest reprezentowana przez zbiór {e, f, g, h}

- formuła ((*p* AND *q*) OR *r* jest więc reprezentowana przez zbiór {c, e, f, g, h}
- formuła (*p* OR *r*) jest reprezentowana przez zbiór {b, c, e, f, g, h}
- formuła (*q* OR *r*) jest reprezentowana przez zbiór {c, d, e, f, g, h}
- formuła ((*p* OR *r*) AND (*q* OR *r*)) jest więc reprezentowana przez zbiór {c, e, f, g, h},
  zatem zbiory reprezentujące lewą i prawą stronę równoważności są identyczne, co oznacza, że równoważność ta jest prawdziwa.

Zadania (do samodzielnego rozwiązania)

1. Zbuduj diagramy Venna dla poniższych formuł i stwierdź, które z nich są tautologiami:
   a. (*p* AND *q*) ⇔ ¬(¬*p* OR ¬*q*)
   b. ¬(*p* AND *q*) ⇔ (¬*p* OR ¬*q*).
2. Zapisz prawe strony równoważności tak, by były one tautologiami i sprawdź rozwiązanie używając tablic logicznych:
   a. ¬(*p* OR ¬*q*) ⇔.....
   b. ¬(*p* AND ¬*q*) ⇔.....

Przykład

Rozważmy sytuację, w której:

- robot wybiera obiekt duży lub mały: *p* OR *q*
- robot nie może wybrać jednocześnie obiektu dużego lub małego: ¬(*p* AND *q*)
- jeśli podłoże jest śliskie, to nie wybiera dużych obiektów: *r* ⇒ ¬*p*, czyli ¬*r* OR ¬*p* (dlaczego?)
- jeśli podłoże nie jest śliskie, to wybiera małe obiekty: ¬*r* ⇒ *q*, czyli *r* OR *q*.

Czy koniunkcja powyższych formuł implikuje, że zostanie wybrany mały obiekt? Innymi słowy, pytamy, czy prawdziwa jest formuła:

((*p* OR *q*) AND ¬(*p* AND *q*) AND (¬*r* OR ¬*p*) AND (*r* OR *q*)) ⇒ *q*.

Znów korzystamy z wcześniejszego diagramu:

- formuła *p* OR *q* jest reprezentowana przez {b, c, e, f, g}
- formuła *p* AND *q* jest reprezentowana przez {c, f}, a więc formuła ¬(*p* AND q) jest reprezentowana przez {a, b, d, e, g, h}
- formuła ¬*r* jest reprezentowana przez {a, b, c, d}
- formuła ¬*p* jest reprezentowana przez {a, d, g, h}, a więc formuła (¬*r* OR ¬*p*) jest reprezentowana przez {a, b, c, d, g, h}
- formuła (*r* OR *q*) jest reprezentowana przez {c, d, e, f, g, h}
- koniunkcja występująca z lewej strony implikacji jest więc reprezentowana przez część wspólną zbiorów {b, c, e, f, g}, {a, b, d, e, g, h}, {a, b, c, d, g, h},
  {c, d, e, f, g, h}, czyli przez {g}
- formuła *q* jest reprezentowana przez zbiór {c, d, f, g}, w którym {g} się zawiera, a więc odpowiedź na pytanie o prawdziwość badanej implikacji jest twierdząca.

Metoda diagramów Venna jest bardzo atrakcyjna ze względu na łatwość wizualnego odkrywania stosunkowo złożonych praw logicznych. Doczekała się wielu badań i uogólnień. Więcej na jej temat można znaleźć np. na bardzo interesującej stronie: http://www.combinatorics.org/Surveys/ds5/VennEJC.html, na której przedstawiono m.in., jak można konstruować diagramy Venna dla więcej niż trzech zbiorów, czyli w taki sposób, aby diagram jednego zbioru miał część wspólną z diagramem każdego innego zbioru.

5 AUTOMATYCZNE WNIOSKOWANIE

Dotychczas omawialiśmy metody wnioskowania skuteczne w niewielkich przykładach. Rzeczywiste systemy obsługujące klasyczny rachunek zdań, nazywane SAT Solvers (SAT pochodzi od angielskiego *satisfiability*, czyli **spełnialność**), potrafią sobie radzić z bardzo dużymi formułami występującymi w niemałym obszarze zastosowań (z formułami mającymi kilkaset, czasem nawet tysiące zmiennych). Ich siła polega na stosowaniu dużej liczby algorytmów skutecznych dla wybranych rodzajów formuł. Poniżej przedstawimy jeden z takich algorytmów, bardzo skuteczny i powszechnie stosowany w informatyce i sztucznej inteligencji, zwany **metodą rezolucji**. Metoda rezolucji działa na koniunkcjach klauzul, przy czym klauzula jest alternatywą zmiennych zdaniowych lub ich negacji. Na przykład klauzulą jest:

p OR ¬q OR ¬*r* OR *s*

zaś nie jest: *p* OR ¬¬*q* (dlaczego?) ani też *p* AND ¬*r* (dlaczego?).

UWAGA: pusta klauzula (niemająca żadnych wyrażeń) jest równoważna fałszowi (czyli 0).

Dlaczego klauzule są ważne

Wiedza w systemach sztucznej inteligencji, w tym w bazach wiedzy, systemach eksperckich itd., zwykle ma postać klauzulową. Mianowicie implikacja w postaci:

(p_1 AND p_2 AND ... AND p_k) ⇒ (r_1 OR r_2 OR ... OR r_m)

jest równoważna klauzuli (dlaczego?):

¬p_1 OR ¬p_2 OR ...OR ¬p_k OR r_1 OR r_2 OR ...OR r_m.

Implikacje wspomnianej postaci odzwierciedlają reguły, jakimi posługujemy się codziennie, np.:

gorączka AND *kaszel* ⇒ *przeziębienie* OR *grypa*
deszcz AND *bezwietrznie* ⇒ *parasol* OR *kurtka_z_kapturem*
deszcz AND *wiatr* ⇒ *samochód*

Przekształcanie formuł do postaci klauzulowej

Aby przekształcić formuły do **postaci klauzulowej** zamieniamy występujące w niej podformuły zgodnie z regułami podanymi poniżej aż do momentu uzyskania koniunkcji klauzul.

Reguły (jako zadanie przekonaj się, stosując metodę tablic logicznych, że formuła zastępowana jest zawsze równoważna formule zastępującej):

1. zastąp (A⇔B) przez (¬A OR B) AND (A OR¬B)
2. zastąp (A⇒B) przez (¬A OR B)
3. zastąp¬¬A przez A
4. zastąp¬(A AND B) przez (¬A OR¬B)
5. zastąp¬(A OR B) przez (¬A AND¬B)
6. zastąp A OR (B AND C) przez (A OR B) AND (A OR C)
7. zastąp (B AND C) OR A przez (A OR B) AND (A OR C).

Zakładamy, że usuwane są też zbędne nawiasy, np. ((A)) można zastąpić przez (A), zaś (A OR B) OR C – przez A OR B OR C oraz powtórzenia wyrażeń, np. (A OR A OR B) można zastąpić przez równoważną formułę (A OR B). Na przykład rozważmy formułę: (¬(*p* AND¬*q*) OR (*p* ⇒ *r*)) OR (*r* ⇔ ¬*s*). Jej sprowadzenie do postaci klauzulowej może składać się z kroków:

- ((¬*p* OR¬¬*q*) OR (*p* ⇒ *r*)) OR (*r* ⇔ ¬*s*)
- ((¬*p* OR¬¬*q*) OR (¬*p* OR *r*)) OR (*r* ⇔ ¬*s*)
- (¬*p* OR¬¬*q* OR¬*p* OR *r*) OR (*r* ⇔ ¬*s*)
- (¬*p* OR *q* OR¬*p* OR *r*) OR (*r* ⇔ ¬*s*)
- (¬*p* OR *q* OR¬*p* OR *r*) OR ((¬*r* OR¬*s*) AND (*r* OR¬¬*s*))

- $(\neg p$ OR q OR $\neg p$ OR $r)$ OR $((\neg r$ OR $\neg s)$ AND $(r$ OR $s))$
- $(\neg p$ OR q OR $r)$ OR $((\neg r$ OR $\neg s)$ AND $(r$ OR $s))$
- $(\neg p$ OR q OR r OR $\neg r$ OR $\neg s)$ AND $(\neg p$ OR q OR r OR r OR $s)$
- $(\neg p$ OR q OR r OR $\neg r$ OR $\neg s)$ AND $(\neg p$ OR q OR r OR $s)$.

Powyższą formułę można z łatwością uprościć (jak?).

Zadania (do samodzielnego rozwiązania)

1. Przekształć do postaci klauzulowej formuły:
   a. $(p$ AND $q) \Leftrightarrow \neg(\neg p$ OR $\neg q)$
   b. $\neg(p$ AND $q) \Leftrightarrow (\neg p$ OR $\neg q)$.
2. Zastąp uzyskane klauzule równoważnymi im implikacjami postaci rozważanej w podrozdziale „Dlaczego klauzule są ważne".

Metoda rezolucji

Metoda rezolucji wykorzystuje zasadę dowodzenia przez doprowadzanie do sprzeczności. Jest ona na w swojej istocie oparta na przechodniości implikacji, czyli na regule mówiącej, że z $(p \Rightarrow q)$ oraz $(q \Rightarrow r)$ mamy prawo wnioskować $(p \Rightarrow r)$. Sformułowanie tej reguły w postaci klauzulowej jest następujące:

$$(\neg p \text{ OR } q) \text{ AND } (\neg q \text{ OR } r) \Rightarrow (\neg p \text{ OR } r).$$

Uogólniając, na dowolne klauzule uzyskamy regułę rezolucji mówiącej, że z klauzul:

$$\neg p_1 \text{ OR} \neg p_2 \text{ OR} \ldots \text{OR} \neg p_k \text{ OR } r_1 \text{ OR } r_2 \text{ OR} \ldots \text{OR } r_m$$
$$p_1 \text{ OR} \neg q_1 \text{ OR} \ldots \text{OR} \neg q_n \text{ OR } s_1 \text{ OR } s_2 \text{ OR} \ldots \text{OR } s_m$$

uzyskujemy klauzulę:

$$\neg p_2 \text{ OR} \ldots \text{OR} \neg p_k \text{ OR } r_1 \text{ OR } r_2 \text{ OR} \ldots \text{OR } r_m \text{ OR} \neg q_1 \text{ OR} \ldots \text{OR} \neg q_n \text{ OR } s_1 \text{ OR } s_2 \text{ OR} \ldots \text{OR } s_m,$$

czyli usuwamy jedną ze zmiennych występującą w pierwszej klauzuli wraz z jej negacją występującą w drugiej klauzuli. Dla wygody zapisaliśmy je jako pierwsze, ale miejsce wystąpienia w klauzulach nie ma znaczenia – ważne jest tylko, by wystąpiły one w dwóch klauzulach.

Jak wspomnieliśmy, metoda rezolucji to zasada wnioskowania przez doprowadzanie do sprzeczności. Oznacza to, że najpierw negujemy sprawdzaną formułę, a następnie staramy się z tej negacji uzyskać fałsz, a więc klauzulę pustą. Jeśli się to uda, początkowa formuła jest tautologią. Jeśli pustej klauzuli nie da się w żaden sposób uzyskać, formuła tautologią nie jest.

Ważnym przykładem jest wykazywanie, że pewna formuła wynika z bazy danych wiedzy, w której wiedza jest reprezentowana jako zbiór klauzul. Chcemy sprawdzić czy $D \Rightarrow q$, gdzie D jest bazą wiedzy, a q jest formułą wykazywaną przy założeniu, że wiedza zawarta w D odzwierciedla interesującą nas rzeczywistość. W metodzie rezolucji zaprzeczamy implikacji $(D \Rightarrow q)$. Zaprzeczenie to jest równoważne formule $(D$ AND $\neg q)$. Czyli $\neg q$ dokładamy do bazy wiedzy D (przekształcając $\neg q$ do postaci klauzulowej) i staramy się wyprowadzić klauzulę pustą. Baza D nie zmienia się! Z wydajnościowego punktu widzenia jest to bardzo ważne, bo zwykle taka baza danych jest duża, podczas gdy badane wnioski zwykle stosunkowo małe, gdyż reprezentują one typowe zapytania użytkowników. Zwykle zapytania mają bardzo niewielkie rozmiary w porównaniu z rozmiarem bazy danych.

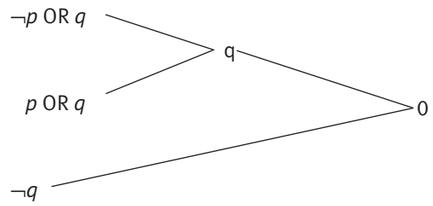
Dla przykładu wykażmy, że z koniunkcji klauzul $(p \Rightarrow q)$ AND $(\neg p \Rightarrow q)$ można wywnioskować q. Jest to formalizacja wnioskowania przez przypadki, bo spełniony jest warunek p albo warunek $\neg p$. Bez względu na to, który z nich jest spełniony, konsekwencją jest q. W życiu często stosujemy takie wnioskowanie. Na przykład, gdy chcemy zabezpieczyć się przed zmoknięciem, bierzemy parasol. W tej sytuacji stosujemy wnioskowanie:

$$(\neg \textit{deszcz} \Rightarrow \neg \textit{zmoknę}) \text{ AND } (\textit{deszcz} \Rightarrow \neg \textit{zmoknę}),$$

a więc wnioskuję, że nie zmoknę bez względu na to, czy będzie deszcz, czy nie.

Zasadę wnioskowania przez przypadki można zapisać w postaci klauzulowej jako:
$(\neg p$ OR $q)$ – pierwsze założenie w postaci klauzulowej
$(p$ OR $q)$ – drugie założenie w postaci klauzulowej
$\neg$q – zaprzeczona konkluzja.

Stosując regułę rezolucji do dwóch pierwszych klauzul uzyskujemy klauzulę $(q$ OR $q)$, usuwamy zbędne powtórzenie q, uzyskujemy więc klauzulę zawierającą jedynie q. Teraz z tej klauzuli oraz z $\neg q$ uzyskujemy klauzulę pustą. Graficznie to wnioskowanie można przedstawić jak na rysunku 2.



Rysunek 2.
Graficzna reprezentacja wnioskowania rezolucyjnego

Zadania (do samodzielnego rozwiązania)

1. Korzystając z metody rezolucji wykaż, że:
   a. $((p$ OR $q)$ AND $r) \Rightarrow ((p$ AND $r)$ OR $(q$ AND $r))$
   b. $((p$ AND $q)$ OR $r) \Rightarrow ((p$ OR $r)$ AND $(q$ OR $r))$.

Przykład

Załóżmy, że mamy następującą bazę wiedzy:

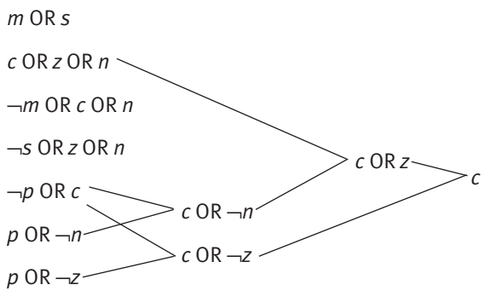
- Pudełka są małe lub średnie $(m$ OR $s)$.
- Każde pudełko jest czerwone, zielone lub niebieskie $(c$ OR z OR $n)$.
- Małe pudełka są czerwone lub niebieskie $(m \Rightarrow (c$ OR $n))$.
- Średnie pudełka są zielone lub niebieskie $(s \Rightarrow (z$ OR $n))$.
- Do przewozu robot wybiera czerwone pudełka $(p \Rightarrow c)$.
- Do aktywności innych niż przewóz robot nie wybiera pudełek niebieskich ani zielonych $(\neg p \Rightarrow (\neg n$ AND $\neg z))$.

Jakie pudełko wybierze robot? Mamy bazę wiedzy zawierającą klauzule (dlaczego?):

$$(m \text{ OR } s), (c \text{ OR } z \text{ OR } n), (\neg m \text{ OR } c \text{ OR } n), (\neg s \text{ OR } z \text{ OR } n), (\neg p \text{ OR } c), (p \text{ OR} \neg n), (p \text{ OR} \neg z).$$

Przykładowe rozumowanie rezolucyjne wykorzystujące te klauzule jest przedstawione na rysunku 3. Z tego rozumowania widzimy, że uzupełnienie bazy wiedzy o klauzulę $\neg c$ dałoby natychmiastowy wywód klauzuli pustej, mamy już bowiem wyprowadzoną klauzulę zawierającą jedynie c. Stąd wnioskujemy, że z naszej bazy wiedzy wynika c (w metodzie rezolucji taka konkluzja – po zaprzeczeniu – trafia do bazy wiedzy).

Zauważmy, że dowód rezolucyjny jest tu naprawdę bardzo krótki. Dla porównania – korzystanie z tablic logicznych wymagałoby skonstruowania $2^6 = 64$ wierszy, mamy bowiem 6 różnych zmiennych.



Rysunek 3.
Przykładowe wnioskowanie rezolucyjne

Przykład

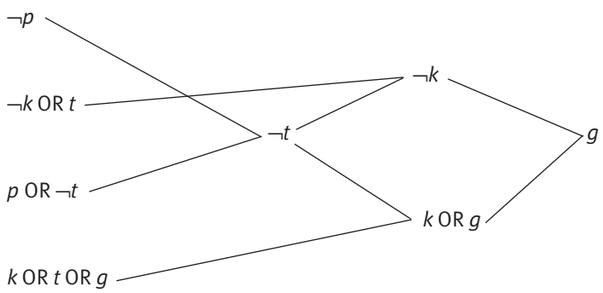
Załóżmy, że nasza baza danych zawiera formuły:

- K twierdzi, że Jan pracuje w soboty (p).
- K twierdzi także, że w soboty Jan czyta książki i nie ogląda telewizji (k AND $\neg t$).
- P twierdzi, że gdy Jan nie pracuje, nie ogląda również telewizji ($\neg p \Rightarrow \neg$t).
- P twierdzi także, że w soboty Jan czyta książki, ogląda telewizję lub gotuje (k OR t OR g).

Wiedząc, że K zawsze kłamie, a P zawsze mówi prawdę, odgadnijmy, co Jan robi w soboty i wykażmy to stosując metodę rezolucji. Skoro K kłamie, a P mówi prawdę, mamy bazę danych klauzul (dlaczego?):

$$\neg p, (\neg k \text{ OR } t), (p \text{ OR} \neg t), (k \text{ OR } t \text{ OR } g).$$

Wnioskowanie metodą rezolucji jest przedstawione na rysunku 4.



Rysunek 4.
Wnioskowanie rezolucyjne dla przykładu o kłamcy i prawdomównym

Z powyższego wywodu uzyskaliśmy jako wniosek g, czyli fakt, że w soboty Jan gotuje. Formalne wykazanie polegałoby na dodaniu zaprzeczonej konkluzji (czyli $\neg g$) do zbioru klauzul i wyprowadzeniu klauzuli pustej, co w obecności klauzuli zawierającej jedynie g jest natychmiastowe.

I znów wnioskowanie rezolucyjne jest krótkie. Oczywiście robot, stosując regułę „na ślepo", może błądzić zanim znajdzie rozwiązanie. Jednak nadal proces dowodzenia jest tu bardzo wydajny. Warto jednak podkreślić, że w pesymistycznym przypadku wymaga on 2^n kroków, gdzie n jest liczbą zmiennych występujących w formule. Nie jest znany żaden algorytm działający efektywnie dla każdej formuły wejściowej.

6 I CO DALEJ?

W wykładzie pojawiła się jedna logika – klasyczny rachunek zdań. Trudno przecenić jego rolę i zakres zastosowań. Wielu naukowców poświęca całą swoją aktywność badawczą np. na poszukiwanie efektywnych systemów wnioskowania dla wybranych rodzajów formuł, pojawiających się w danych zastosowaniach w wyniku modelowania lub tłumaczenia łatwiejszych w użyciu formalizmów. W końcu jeden z kilku problemów milenijnych, za rozwiązanie którego czeka nagroda w wysokości miliona USD, dotyczy wykazania istnienia lub nieistnienia efektywnego algorytmu badania, czy dana formuła klasycznego rachunku zdań jest spełnialna. Niemniej jednak z punktu widzenia wielu ważnych zastosowań jest to bardzo prosty formalizm i – jako taki – nie jest w stanie dobrze modelować złożonej rzeczywistości systemów informatycznych, języka naturalnego, reprezentacji wiedzy czy wnioskowania o świecie rzeczywistym.

W minionych czterdziestu latach w informatyce prowadzono bardzo intensywne badania nad logikami, np. modelującymi wnioskowanie człowieka lepiej niż klasyczny rachunek zdań (znanymi w sztucznej inteligencji jako wnioskowanie zdroworozsądkowe lub niemonotoniczne). Dziedzina ta nadal intensywnie się rozwija, bowiem szuka się coraz lepszych metod modelujących inteligentne zachowania w systemach autonomicznych, jak bezzałogowe helikoptery czy złożone systemy robotyki.

Ale o tym kiedy indziej, na przykład na studiach informatycznych...

LITERATURA

1. Ben-Ari M., *Logika matematyczna w informatyce,* WNT, Warszawa 2004
2. Shannon C.E, *A Symbolic Analysis of Rely and Switching Circuits*, M. Sc. Thesis, MIT 1938

ZAŁĄCZNIK

Poniżej przedstawiamy krótki fragment pracy magisterskiej C.E. Shannona z 1940 roku. Uchodzi ona za najlepszą pracę magisterską XX wieku i przekłada język logiki na obwody, z jakich korzystają inżynierowie konstruujący także współczesne komputery [2, s. 11].

TABLE I

Analogue Between the Calculus of Propositions and the Symbolic Relay Analysis

Symbol	Interpretation in relay circuits	Interpretation in the Calculus of Propositions
X	The circuit X.	The proposition X.
0	The circuit is closed.	The proposition is false.
1	The circuit is open.	The proposition is true.
X + Y	The series connection of circuits X and Y	The proposition which is true if either X or Y is true.
XY	The parallel connection of circuits X and Y	The proposition which is true if both X and Y are true.
X'	The circuit which is open when X is closed, and closed when X is open.	The contradictory of proposition X.
=	The circuits open and close simultaneously.	Each proposition implies the other.

Język językowi nie równy

Jan Madey

Instytut Informatyki, Uniwersytet Warszawski

madey@mimuw.edu.pl

Wstęp

Człowiek od zarania dziejów miał potrzebę komunikowania się z innymi ludźmi, zarówno bezpośrednio, jak i zdalnie. Podobne potrzeby wykazują zresztą i zwierzęta. Różne były formy tej komunikacji, ale niewątpliwie najważniejszą z nich jest **język**, który w swojej wersji pisanej stał się podstawą rozwoju cywilizacji. Językowi są więc poświęcone zaawansowane badania w różnych dyscyplinach naukowych (z językoznawstwem, zwanym także lingwistyką, na czele). Ale język, będąc narzędziem pracy wielu specjalistów różnych dziedzin, stał się też przedmiotem badań niektórych z nich. Szczególną pozycję ma tutaj informatyka, gdyż języki programowania odgrywają w tej dyscyplinie kluczową rolę.

W rozdziale pierwszym niniejszego opracowania zajmiemy się wybranymi ogólnymi aspektami lingwistycznymi, a następnie (rozdz. 2 i 3) prześledzimy – w sposób uproszczony oraz subiektywny – ewolucję języków programowania[6].

Spis treści

[6] Rozdziały 2 i 3 niniejszego tekstu są w dużej części zaczerpnięte z pracy [JM04].

1 JĘZYK, MOWA, PISMO, ALFABET

Bez problemu można sprawdzić w Internecie znaczenie terminu język (przede wszystkim w Wikipedii – wolnej encyklopedii: http://pl.wikipedia.org/wiki/ oraz w WIEM: http://wiem.onet.pl/).

Spójrzmy zatem do mniej obecnie popularnych źródeł papierowych, a w szczególności do tych mało dostępnych (jak na przykład Wielka Encyklopedia Powszechna Ilustrowana, WEPI), by sprawdzić, czy i ponad 100 lat temu tak samo lub co najmniej podobnie rozumiało się te pojęcia.

1.1 DEFINICJE SPRZED PONAD WIEKU

Wielka Encyklopedia Powszechna Ilustrowana, wydawana w latach 1880-1914 (55 tomów) nigdy nie została ukończona – została przerwana na haśle „Patroklos", ze względu na I wojnę światową. Jej wydawcy to Franciszek Juliusz Granowski i Saturnin Józef Sikorski, ale popularnie jest nazywana „encyklopedią Sikorskiego". Oto jak omówiono w niej interesujący nas termin i inne z nim związane (przedstawiamy skróty, z zachowaniem oryginalnej pisowni):

„**Język i języki**, w znaczeniu lingwistycznym, językoznawczym, w znaczeniu *mowy ludzkiej*. Nazwa ta stosuje się pod wpływem przenośni, przyczym główny organ widomy wykonawczy, główne narzędzie wymawiania, *język*, w znaczeniu całej czynności, w znaczeniu procesu i całości mówienia. (...) Oczywiście nazwa ta zjawiła się wtedy, kiedy język, w ustach ludzkich umieszczony, stał się istotnie głównym narzędziem wymawiania. W pierwocinach ludzkości, kiedy wymawianie było umiejscowione głębiej, przeważnie w krtani, trudniej mogłoby się dokonać podobne rozszerzenie nazwy języka. – Skojarzenie nazwy języka, jako głównego narzędzia wymawiania, z nazwą języka, jako mowy ludzkiej, widoczne jest dotychczas w takich wyrażeniach, jak np. polskie: „długi język", „miele językiem", „na końcu języka", „ciągnąć kogo za język", „język mu skołowaciał", „dostać się na języki" (...).– W dalszym ciągu „*język*" znaczy: wiadomość, doniesienie, (...) itd.; np. „zasięgnąć języka", „z językiem przybieżał" (...).– Następnie J. jest to mowa, która odróżnia, z jednej strony, człowieka od człowieka (mówienie indywidualne), plemię od plemienia, naród od narodu (język w ścisłym znaczeniu), z drugiej zaś strony, ludzkość czyli rodzaj ludzki w ogóle (mowa ludzka) od wszystkich innych zwierząt. (...) W znaczeniu języka plemiennego lub narodowego synonimami *języka* lub też wyrazami blizko znacznemi są: *mowa, żywe słowo, narzecze, gwara, dyjalekt* itp., a w znaczeniu języka indywidualnego – *styl*, *sposób mówienia*, itp. („język ostry, zuchwały, rozwiązły, pospolity, banalny, kwiecisty, wzniosły (...)"). Przenośnie mówimy: „J. kwiatów", „J. przyrody", w znaczeniu zaś bardziej zbliżonym do języka ludzkiego – „J. zwierząt". – J., w najobszerniejszym znaczeniu tego wyrazu, w znaczeniu przenośnym, oznacza wszelkie sposoby porozumiewania się, a więc wszelką mimikę i giesty (język palców itp.), pismo (język znaków itp.), głosy mimowolne i odruchowe (...). – Obok tego zwykłego pojmowania społeczno-towarzyskiego można J. określać z rozmaitych stanowisk naukowych. Dla fizjologa mówienie i język wogóle jest funkcją organizmu ludzkiego (...). Dla antropologa, jako przyrodnika, język jest jedną z cech, wyróżniających rodzaj ludzki z pomiędzy całego szeregu istot podobnie organizowanych. Nareszcie dla psychologa język jest usystematyzowanym zbiorem wyobrażeń, a więc zjawiskiem o podstawie wyłącznie psychicznej (...)" [WEPI XXXIII, 1903, s. 266-278].

Mowa – niestety nie udało nam się dotrzeć do tego hasła w WEPI. Ale w wyżej cytowanych zapisach pojawił się ten termin kilkakrotnie i jest traktowany jako synonim języka mówionego: „– Skojarzenie nazwy języka, jako głównego narzędzia wymawiania, z nazwą języka, jako **mowy** ludzkiej, widoczne jest (...)", „Następnie J. jest to **mowa**, która odróżnia (...)", „W znaczeniu języka plemiennego lub narodowego synonimami *języka* lub też wyrazami blizko znacznemi są: ***mowa***, (...)".

Pismo – jak już wiemy, encyklopedia Sikorskiego została przerwana na haśle „Patroklos". Zauważmy jednak, podobnie jak powyżej, że w określeniu pojęcia *język*, pojawia się też termin *pismo*: „– J., w najobszerniejszym znaczeniu tego wyrazu, w znaczeniu przenośnym, oznacza wszelkie sposoby porozumiewania się, a więc wszelką mimikę i giesty (język palców itp.), **pismo** (język znaków itp.), głosy mimowolne i odruchowe (...)."

Pozostał już tylko alfabet. Takie zapisy o alfabecie znajdują się w naszej encyklopedii [WEPI I, 1890, s. 668 i s. 32-36]:
„**Alfabet**, ob. Abecadło".
„**Abecadłem** nazywa się zbiór wszystkich znaków piśmiennych, albo liter na wyrażenie pojedyńczych dźwięków pewnego języka, ułożony w pewien zwyczajem ustalony porządek. Nazwa ta wzięła początek od brzmienia w mowie naszej trzech pierwszych głosek *a b c* w połączeniu z przyrostkiem *–dło* (pochodnym *–adło*). Dawniejszą od tego wyrazu jest nazwa grecka *alfabet*, utworzona z nazw dwóch początkowych liter w abecadle greckim *alfa*, *beta*, odpowiadającym naszym *a b* (...)".

1.2 NIECO WSPÓŁCZESNOŚCI

Z zacytowanych wyżej tekstów wynika, że nie uległo jakiejś istotnej zmianie spojrzenie na interesujące nas terminy. Język, jako narząd w jamie ustnej, biorący przede wszystkim udział w ssaniu, żuciu, połykaniu pokarmów, poprzez swoją dodatkową funkcję istotną w procesie komunikowania się ludzi stał się w sposób naturalny wyrazem używanym dla określania różnych form tej komunikacji. Stąd i następujące znaczenia tego terminu [SJP78, tom pierwszy, s. 844]:

- «zasób wyrazów, zwrotów i form określanych przez reguły gramatyczne, funkcjonujący jako narzędzie porozumiewania się przez członków jednego narodu, społeczeństwa; mowa»: Język polski, angielski, francuski. Język ojczysty. Języki obce (...). ΔJęzyki starożytne, klasyczne (...) ΔJęzyk literacki (...)
- «zasób wyrazów, zwrotów i form, używanych w celu porozumienia się przez ludzi pewnego środowiska, zawodu, regionu; gwara, dialekt, żargon»: Język techniczny, dyplomatyczny, łowiecki, uczniowski, złodziejski (...)
- «sposób wyrażania się, wysławiania się, charakterystyczny dla danego autora, dzieła, epoki; styl»: Język Prusa. Język utworu (...) ΔJęzyk książkowy, pisany, ΔJęzyk potoczny, mówiony
- *dawn.* «jeniec schwytany dla powzięcia wiadomości o nieprzyjacielu».

Możemy jeszcze wskazać wiele innych zwrotów, w których pojawia się wyraz *język* w zbliżonym znaczeniu: język filmu, tłumu, dźwięków, nauki, migowy, biznesu, uczuć itd. Ważny jest także podział na *języki naturalne* i *języki sztuczne*. Tym ostatnim poświęcimy więcej uwagi w dalszych rozdziałach.

Zauważmy teraz, że w najszerszym lingwistycznie znaczeniu termin *język* oznacza zarówno język mówiony, czyli mowę, jak i język pisany, czyli pismo. Z kolei mówiąc o piśmie w naturalny sposób dochodzimy do terminu *alfabet*. Trzeba przy tym zauważyć, że rola języka pisanego jest nie do przecenienia, co oddaje m.in. poniższy tekst [DD72, s. 24]:

„Pismo jest graficznym odbiciem języka. Każdy element systemu pisma: znak pisarski, litera, 'hieroglif', słowo pisane, ma swój odpowiednik w określonym elemencie systemu pierwotnego, czyli mowy (...). Pismo jest więc naturalną metodą, czy raczej najważniejszą z naturalnych metod przekazywania mowy i umożliwia wymianę myśli ludziom, którzy nie mogą się posłużyć mową na skutek odległości w czasie lub przestrzeni bądź z jakichś innych przyczyn".

Bez pisma nie byłby możliwy cywilizacyjny rozwój człowieka. Ale droga do pojawienia się pisma współczesnego była długa i ciernista, a konsekwencje niektórych historycznie błędnych decyzji są ponoszone do dzisiaj, co obrazuje poniższa krótka historia pisma japońskiego.

1.3 PRZYKŁAD HISTORYCZNIE BŁĘDNEJ DECYZJI – PISMO JAPOŃSKIE (KANJI)

Mówiony język japoński ma klarowną strukturę (choć bardzo odmienną od tej, do której jesteśmy przyzwyczajeni) oraz łatwą – dla nas, Polaków – wymowę. Inaczej jednak wygląda sprawa pisma. Zamiast opracować własne, Japończycy – będąc niejako pod względem kulturowym „kolonią" Chin – od około III/IV wieku n.e., zaczęli przejmować chińskie pismo, które zupełnie nie pasowało do ich języka mówionego. Proces ten trwał kilkaset lat i w efekcie powstał bardzo skomplikowany system, obejmujący zarówno ideogramy chińskie (*kanji*), jak i dwie postaci pisma sylabicznego: *hiragana* i *katakana*. Ideogramy wykorzystuje się tylko do zapisywania rzeczowników, przymiotników i pni czasownikowych. Ale japoński ma również końcówki gramatyczne (których brakuje w języku chińskim), przyimki itd., i do ich wyrażania używa się *hiragany*. Natomiast *katakana* służy przede wszystkim do transliterowania nazw własnych (np. nazwisk europejskich) oraz słów obcego pochodzenia.

To jeszcze nie koniec. Każdy znak *kanji* ma w języku japońskim dwie odrębne wymowy: *kun* (japońską) i *ON* (chińską, zwaną też sino-japońską), przy czym ta ostatnia ma niewiele wspólnego ze współczesnym mówionym językiem chińskim – brzmi tak, jak Japończycy słyszeli wtedy, gdy znaki te były przyswajane. Nie ma przy tym jednoznacznych zasad określających, kiedy stosuje się którą wymowę. Ogólnym kryterium jest to, czy dany znak jest samodzielną jednostką znaczeniową (wtedy czyta się go na ogół „po japońsku"), czy też dopiero połączony z innymi taką tworzy (wówczas czyta się go zwykle „po chińsku"). Ponieważ wymowa chińska była i jest różna w różnych prowincjach i ponadto zmieniała się w rozmaitych okresach historycznych, to znak *kanji* ma zazwyczaj kilka wersji czytania *ON*.

Dla ilustracji rozważmy jeden z najprostszych znaków *kanji* – ideogram oznaczający człowieka:

Jego wymowa japońska to *hito*, zaś chińska to *JIN* lub *NIN*.

Weźmy teraz pod uwagę ideogram oznaczający duży (człowiek z rozpostartymi rękoma):

Tutaj mamy następujące wymowy: *ookii* (japońska) oraz *TAI*, *DAI* (chińska).

Zastanówmy się teraz, co oznacza następujące połączenie obu tych znaków w jedną jednostkę znaczeniową:

Jest to duży człowiek, czyli *dorosły*. No i teraz najważniejsze pytanie – jak tę kombinację znaków odczytu-

jemy? Gdyby zastosować wspomnianą wcześniej regułę, to powinno to być *taijin* lub *tajnin*, ew. *daijin* bądź *dainin*. A może jednak *taihito* lub *daihito*? Nic z tych rzeczy! Te dwa ideogramy zapisane razem wymawia się *otona*, czyli bez żadnego związku z japońską i chińską wymową znaków składowych. Dlaczego? Ponieważ po japońsku dorosły to właśnie *otona*...

Powyższe fakty ilustrują tezę, iż przyjęte w Japonii pismo nie pasuje do mowy japońskiej. Dodajmy do tego, że liczebność *kana*[7] (czyli *hiragany* i *katakany*) to 100 znaków, ale już jeśli chodzi o *kanji*, to mamy do czynienia z tysiącami ideogramów (absolwent szkoły musi znać ich około 2000), z których – jak już wiemy – każdy ma czytanie japońskie i chińskie, niekiedy w kilku wariantach. Do tego dochodzi kłopot z samą pisownią – znaki są charakteryzowane „dynamicznie", tzn. trzeba znać kolejność pisania każdej kreski wchodzącej w skład ideogramu oraz jej kierunek (lewo-prawo czy odwrotnie, góra-dół czy odwrotnie itd.). A liczba kresek w znaku może przekroczyć nawet 20.
Aż podziw bierze, że Japończycy są w stanie nauczyć się w szkole czegokolwiek więcej poza własnym pismem!

1.4 JĘZYKI NATURALNE A JĘZYKI SZTUCZNE

Uważa się, że na świecie istnieje obecnie około 2500-3000 języków, a po wielu innych nie pozostało już śladu. Próbą „ludowej" odpowiedzi na pytanie o pochodzenie wielości języków jest opowieść o wieży Babel: „**Wieża Babel**, według biblijnej *Księgi Rodzaju* wieża, jaką zaczęto wznosić w ziemi Szinear w Babilonie (po hebrajsku Babel) z zamiarem, by sięgnęła nieba. Rozgniewany zuchwałością budowniczych Bóg sprawił, że zaczęli mówić różnymi językami i nie mogąc się porozumieć, musieli przerwać budowę" (por. [WIEM]).

Języki naturalne rodziły się niejako spontanicznie, ale były też różne próby opracowania języka uniwersalnego, który stałby się międzynarodowym standardem. Kończyły się one na ogół fiaskiem, a rolę takiego standardu w różnych okresach przejmowały do pewnego stopnia modne bądź użyteczne konkretne języki naturalne – obecnie jest to niewątpliwie angielski. Z języków sztucznych z pewnością na uwagę zasługuje *esperanto*. Podstawy tego języka, zwanego pierwotnie Lingvo Internacia (pol. język międzynarodowy), przedstawił Ludwik Zamenhof[8] pod pseudonimem Dr. Esperanto w 1887 roku. Główną ideą Zamenhofa było opracowanie neutralnego oraz łatwego do nauki języka, przydatnego do międzynarodowej komunikacji, niezastępującego jednak narodowych języków. Choć cel ten nie został nigdy w pełni osiągnięty, to esperanto jest używane przez międzynarodową wspólnotę (której wielkość jest szacowana nawet na dwa miliony osób), podczas kongresów i dyskusji naukowych, a także w muzyce, teatrze, kinie i mediach prasowych. Esperanto doczekało się także formalnego międzynarodowego uznania w postaci dwóch rezolucji UNESCO.

W dalszej części niniejszego opracowania skoncentrujemy się na ważnej grupie języków sztucznych, a mianowicie na **językach programowania**. Zauważmy na wstępie, że w przypadku języków naturalnych, mowa poprzedza pismo, czyli w pewnym sensie semantyka jest pierwotna, a syntaktyka wtórna – do wypowiedzi, których znaczenie jest zrozumiałe dobieramy sposób ich zapisu. Inaczej jest w kwestii języków sztucznych. Tam często zaczynamy od zdefiniowania zapisu (składni), a potem objaśniamy mniej lub bardziej ściśle jego semantykę.

[7] Znaki *kana* – to nie jest prawdziwe pismo sylabiczne, ponieważ nie używa się ich samodzielnie – choć były takie próby – lecz tylko wskazuje się za ich pomocą cechy fleksyjne, takie jak czasy czasowników, przyimki itp., podczas gdy same rzeczowniki, czasowniki i przymiotniki wyraża się zawsze nadal znakami *kanji*.

[8] Warto zauważyć, że Ludwik Zamenhof (właściwie Eliezer Lewi Samenhof, ur. 15 grudnia 1859 w Białymstoku, zm. 14 kwietnia 1917 w Warszawie, lekarz), już w wieku 10 lat napisał dramat *Wieża Babel*, czyli tragedia białostocka w pięciu aktach; uważał bowiem, że główną przyczyną nieporozumień i sporów między ludźmi jest bariera językowa. Jeden, wspólny język miał być jej rozwiązaniem.

2 EWOLUCJA JĘZYKÓW PROGRAMOWANIA

Przez język programowania rozumie się powszechnie **notację** do zapisu algorytmów w celu ich wykonania przez komputer. Notacja ta powinna być możliwie dogodna dla człowieka, ale jednocześnie precyzyjna i jednoznaczna, ze ściśle określoną składnią (syntaktyką) oraz znaczeniem (semantyką). Algorytm zapisany w języku programowania, czyli program, musi być następnie przetłumaczony na język wykonywalny przez procesor komputera, tzn. na jego język wewnętrzny. Proces tłumaczenia przebiega albo jednorazowo, dla całego programu – mówimy wówczas o **kompilacji**, albo etapami, fraza po frazie – mamy wówczas **interpretację**. Stąd i narzędzia realizujące ów proces (translatory) nazywa się odpowiednio **kompilatorami** lub **interpretatorami**. Tekst w języku programowania nosi miano **kodu źródłowego**, zaś jego odpowiednik po translacji, to **kod wynikowy** (lub **kod maszynowy**).

2.1 WCZESNE JĘZYKI PROGRAMOWANIA

Komputery *sensu stricto* pojawiły się w latach czterdziestych ubiegłego wieku i od tego czasu zaczyna się także historia rozwoju języków programowania. Algorytmy były wówczas zapisywane w językach wewnętrznych (zwanych też **językami niskiego poziomu**), tzn. formułowane w kategoriach rozkazów – inaczej instrukcji – procesorów konkretnych komputerów. A repertuar takich rozkazów był i nadal jest dosyć ubogi – w pewnym uproszczeniu można powiedzieć, że składa się on z różnych poleceń organizacyjnych (typu: „przenieś zawartość rejestru X do rejestru Y", czy „przesuń bity w lewo") oraz z prostych operacji arytmetycznych i logicznych. Zarówno instrukcje, jak i ich argumenty, są przy tym reprezentowane jako ciągi zerojedynkowe (binarne). Innymi słowy alfabet tych języków był bardzo ubogi – składał się z dwóch znaków. Co więcej, ten sam ciąg binarny umieszczony w konkretnym miejscu pamięci komputera, może być w ramach tego samego programu zinterpretowany raz jako argument, a raz jako rozkaz – to był właśnie genialny pomysł Johna von Neumanna, pozwalający w szczególności na formułowanie programów, które mogą się same modyfikować!

Jak wyglądają podane we wstępie nasze trzy atrybuty? Notacja – to, jak już wiemy, ciągi zerojedynkowe. Szybko jednak zamieniono ją na nieco bardziej czytelną formę, tzn. dopuszczono używanie znaków alfanumerycznych, przyjmując pewne skróty mnemotechniczne dla oznaczenia rozkazów oraz zapisując liczby w systemie dziesiętnym. Oznaczało to wszakże konieczność tłumaczenia takich tekstów na postać binarną, ale wobec zależności „jeden do jednego" było to zadanie banalne i bez trudu opracowano pierwsze asemblery (pod tym terminem rozumie się zarówno język mnemotechniczny, jak i jego translator).

Komputery tamtych czasów były bardzo dużymi, kosztownymi i zawodnymi urządzeniami. Programowane algorytmy – zwłaszcza w latach II wojny światowej oraz bezpośrednio powojennych – wiązały się z próbą rozwiązania różnych problemów naukowych o strategicznym znaczeniu przede wszystkim dla celów militarnych. Stąd dostęp do komputerów miała tylko wyselekcjonowana grupa ekspertów i kwestia wygody notacyjnej przy programowaniu miała tym samym drugorzędne znaczenie – inne sprawy były znacznie ważniejsze. Ale taka sytuacja nie trwała długo.

2.2 PIERWSZE JĘZYKI WYSOKIEGO POZIOMU

W połowie ubiegłego wieku zaczęły się pojawiać **języki programowania wysokiego poziomu**. Początkowo chodziło jedynie o wprowadzenie bardziej czytelnej notacji, bez zmiany pojęć używanych do zapisu algorytmu. Później zaczęto nieco uogólniać i modyfikować te pojęcia, ale nadal było to jeszcze podejście wstępujące (ang. *bottom-up*) w tym sensie, że w tle były rozwiązania techniczne konkretnego komputera, a programowany algorytm należało wyrażać w kategoriach udostępnianych przez język wewnętrzny tegoż komputera.

Do najbardziej znanych języków tej klasy należy niewątpliwie **Fortran** (***FOR****mula* ***TRAN****slator*), opracowany w latach 1954-57 przez zespół z firmy IBM pod kierunkiem Johna Backusa, w celu ułatwienia korzystania z komputerów serii IBM 704. Architektura tego konkretnego komputera odbiła swoje piętno na rozwiąza-

niach przyjętych w języku Fortran (dotyczy to np. postaci instrukcji warunkowej, czy też nazw kanałów wejściowych). Język ten przeszedł w następnych latach dużą metamorfozę – powstał m.in. Fortran IV, Fortran 77, Fortran 90, Fortran 95 oraz Fortran 2003 (!). Fortran jest więc nadal popularny, zwłaszcza w gronie osób programujących algorytmy dotyczące obliczeń naukowych, a przede wszystkim problemów numerycznych. W trakcie prac nad językiem Fortran Backus wymyślił pierwszą wersję swojej notacji BNF[9], którą wykorzystano do opisu składni tego języka. W nieco zmodyfikowanej przez Petera Naura postaci notacja ta została użyta do opisu języka Algol 60, który omawiamy dalej.

Przełom nastąpił pod koniec lat pięćdziesiątych, gdy międzynarodowe grono specjalistów z Europy i Ameryki Północnej rozpoczęło prace nad językiem **Algol 60** (po doświadczeniach z jego nieco ułomnym pierwowzorem, językiem Algol 58). Już sama nazwa tego języka, będąca skrótem od słów ***ALGO****rithmic* ***L****anguage*, wskazywała na zmianę podejścia – tym razem jego twórcom chodziło o zdefiniowanie notacji do formułowania algorytmów, niemającej żadnego związku z architekturą konkretnego komputera. Innymi słowy zmieniono dotychczasowe podejście wstępujące na zstępujące (ang. *top-down*) i skoncentrowano się na obiektach oraz operacjach potrzebnych do wyrażania algorytmów, które później byłyby automatycznie wykonywane. Język Algol 60 występował przy tym w tzw. wersji wzorcowej, mającej formalnie opisaną składnię (w notacji BNF, co należy uznać za osobisty sukces Petera Naura) i rygorystycznie – choć tylko w języku naturalnym – znaczenie oraz w konkretnych realizacjach, tzn. w wersjach powiązanych już z kompilatorami dla konkretnych komputerów. Przykładem bardzo udanej takiej konkretnej realizacji (implementacji) języka Algol 60 był Gier Algol w swoich kolejnych edycjach, opracowany przez Naura dla duńskiego komputera GIER.

Niezależnie od prac nad językiem Algol, grupa osób w USA pod kierunkiem Grace Murray Hopper, reprezentująca różne organizacje (w tym konkurujące wytwórnie komputerów), rozpoczęła w 1959 roku prace zmierzające do zaprojektowania języka maszynowo-niezależnego. Tym razem chodziło jednak o przetwarzanie danych, czyli o przekształcanie zbiorów danych alfanumerycznych oraz tworzenie różnych raportów, ale przy tym w sposób zrozumiały przez przeciętnego człowieka. To ostatnie założenie spowodowało, że w powstałym języku – nazwanym **Cobol** (***CO****mmon* ***B****usiness* ***O****riented* ***L****anguage*) – aż do przesady są wykorzystywane wyrazy z języka naturalnego (nawet popularne operacje arytmetyczne nie są oznaczane tradycyjnymi symbolami, ale słowami typu ADD lub MULTIPLY). Cobol także przeszedł różne ewolucje i przetrwał aż do dzisiejszych czasów.

Trzeba odnotować jeszcze jedno istotne zdarzenie z końca lat pięćdziesiątych ubiegłego wieku – w 1958 roku John McCarthy z MIT wymyślił abstrakcyjną notację do opisu funkcji rekurencyjnych, która łatwo dawała się rozszerzyć do języka programowania ukierunkowanego na przetwarzanie listowych struktur danych, nazwanego zatem **Lisp** (***LIS****t* ***P****rocessor*). Pod tym akronimem kryje się – podobnie jak ma to miejsce w przypadku języka Fortran – właściwie cała klasa języków, bo Lisp ulegał wielu modyfikacjom oraz pojawiały się jego najróżniejsze wersje i dialekty, takie jak Lisp 1.5, Mlisp, Lisp-A, Interlisp, Common Lisp czy Scheme. Tak zwany czysty Lisp został zaimplementowany na komputerze IBM 704 (około 1960 roku) i – znowu podobnie jak w wypadku języka Fortran – znalazło to pewne odbicie w samym języku (operacje listowe *car* i *cdr*, to instrukcje z repertuaru komputera IBM 704). Natomiast pierwsze większe rozszerzenie języka, Lisp 1.5, zaimplementowano w 1962 roku na komputerze IBM 7090, a później na wielu innych.

2.3 JĘZYKI PROGRAMOWANIA A JĘZYKI SPECYFIKACJI

Zapis algorytmu w konkretnym języku programowania jest tylko jedną z faz procesu wytwarzania oprogramowania. Użycie ścisłej notacji jest niezbędne w całym tym procesie, poczynając od spisywania wyników analizy wymagań (co stanowi jego pierwszą fazę), a na kodzie konkretnego komputera skończywszy.

[9] Akronim ten oryginalnie oznaczał Backus Normal Form. Później jednak, dla podkreślenia zasług Petera Naura w spopularyzowaniu tej notacji przy okazji opisu Algolu, przyjęto traktować BNF jako skrót nazwy Backus–Naur Form.

Rozważmy teraz bardzo krótko kwestię notacji używanej do specyfikacji na różnych poziomach abstrakcji, a dokładniej – związek między tą notacją a językami programowania.

Wspomniany wcześniej Algol 60 początkowo zaistniał tylko w wersji wzorcowej, którą powinno się traktować właśnie jako język specyfikacji. W wersji tej nie było w szczególności operacji wejścia-wyjścia, które pojawiały się dopiero w konkretnych realizacjach[10]. Te ostatnie mogły ponadto wprowadzać różne modyfikacje wersji wzorcowej (ograniczenia, rozszerzenia, zmiany składniowe itp.), po części niezbędnych przy opracowywaniu kompilatorów, a po części wynikających z inwencji twórców tychże. Przykładem bardzo udanej implementacji języka Algol 60 był Gier Algol w swoich kolejnych edycjach opracowany przez Petera Naura dla duńskiego komputera GIER. Z chwilą powstania konkretnych realizacji, język Algol ten stał się w pewnym sensie językiem wykonywalnych specyfikacji, gdyż przejście od specyfikacji do wykonywalnego kodu było już automatyczne, poprzez kompilację. Poprawność kodu względem specyfikacji zależała więc już tylko od poprawności tej kompilacji, co oznaczało konieczność (jednorazowego) zweryfikowania kompilatora.

Efektywność kodu wynikowego po kompilacji (zwłaszcza wobec niezaawansowanych wówczas jeszcze technik tworzenia kompilatorów i przy kiepskich parametrach sprzętu, a jednocześnie wysokiej jego cenie) była przedmiotem dużej troski. Doprowadziło to do wypracowania możliwości włączania do programu w języku wysokiego poziomu wstawek napisanych w asemblerze. Innymi słowy została dana możliwość łączenia specyfikacji (tekst w języku wysokiego poziomu) z kodem (tekst w asemblerze) – techniki, która bywa i dzisiaj stosowana, choć na innym poziomie abstrakcji, i jest niekiedy niesłusznie uważana za nowatorskie rozwiązanie.

Widzimy więc, że rozróżnianie pomiędzy językiem specyfikacji a językiem programowania jest kwestią umowną.

2.4 PODSTAWOWE PARADYGMATY PROGRAMOWANIA

Przytoczone w poprzednich podrozdziałach przykłady wskazują w szczególności na metamorfozę komputerów i ich zastosowań. Z urządzeń elitarnych, dostępnych tylko dla wybrańców, zaczęły z wolna trafiać pod strzechy różnych instytucji, stawać się narzędziem pracy naukowców z wielu dziedzin, a także pojawiły się ich pierwsze praktyczne aplikacje – wynikało to oczywiście ze zmian technologicznych umożliwiających lepsze i tańsze konstrukcje sprzętowe. Zaowocowało to dalszym znacznym zwiększeniem zainteresowania językami programowania. Pojawiły się przy tym próby odmiennego podejścia do formułowania rozwiązywanych problemów.

Języki wewnętrzne komputerów, jak również wspomniane w poprzednim punkcie Fortran, Algol i Cobol, są nazywane **językami imperatywnymi**, ponieważ sposób formułowania w nich algorytmów sprowadza się do wydawania poleceń, typu: „rób po kolei to a to, a następnie – jeśli jest tak a tak – to kontynuuj, zaś w przeciwnym wypadku przejdź do innej, wskazanej fazy obliczeń”. W językach tych mamy zmienne, którym są nadawane wartości w wyniku ewaluacji wyrażeń. W językach imperatywnych wysokiego poziomu występują też różne formy instrukcji warunkowej oraz iteracyjnej, a także mechanizm grupowania kilku instrukcji w jedną, złożoną. W zależności od języka mamy ponadto pewne możliwości definiowania abstrakcyjnych operacji (w postaci dość powszechnego aparatu procedur) oraz struktur danych (ograniczonych w początkowych językach do tablic, służących do reprezentowania macierzy). Języki te stanowią więc najbardziej naturalne „rozszerzenie” komputera.

Natomiast język Lisp (a raczej jego protoplasta **IPL** – ***I****nformation* ***P****rocessing* ***L****anguage*) zapoczątkował inny paradygmat programowania, zwany funkcyjnym, którego podstawą teoretyczną jest tzw. rachunek lambda z typami.

[10] Autorzy języka Algol 60 uważali wówczas, że wprowadzanie danych nie wymaga specyfikacji, ma „charakter przyziemny” i powinno być uwzględniane dopiero przy opracowywaniu kompilatorów, a więc znajduje swoje odbicie w konkretnych realizacjach. Szybko jednak uznano to za błąd — następniki języka Algol 60 zajmowały się już wejściem-wyjściem na poziomie wzorcowym.

Obliczenia polegają tutaj nie na wykonywaniu kolejnych instrukcji, ale na ewaluacji funkcji (często rekurencyjnych). Innymi słowy zapisujemy nie tyle sposób obliczania interesującej nas wartości, co matematyczną funkcję definiującą tę wartość. W czystym języku funkcyjnym znika zmienna i instrukcja imperatywna przypisania, a przy obliczaniu wartości funkcji nie mogą występować tzw. efekty uboczne. Najbardziej typowym przedstawicielem tej klasy jest język **Haskell** z końca lat osiemdziesiątych ubiegłego wieku (nazwany tak dla uczczenia amerykańskiego matematyka i logika, Haskella Curry'ego). Jego najnowsza wersja, to Haskell 98. Króluje jednak „mniej ortodoksyjne" podejście, stanowiące pewną fuzję stylu funkcyjnego z elementami programowania imperatywnego lub/i obiektowego. Reprezentantem tego nurtu jest wspomniany już wcześniej język Lisp z pochodnymi, a także **ML** (***M**eta **L**anguage*) i wywodzące się z niego **SML** (***S**tandard **ML***) oraz **Ocaml** (***O**bjective **CAML***).

Kolejny paradygmat, to programowanie w logice. Tutaj czołowym przedstawicielem jest język **Prolog** (***PRO**gramming **LOG**ic*), zaprojektowany w 1972 roku przez Alaina Colmerauera (francuskiego specjalistę od sztucznej inteligencji) do zastosowań przy przetwarzaniu języka naturalnego. Początkowo Prolog był znany i rozwijany tylko w niewielkim kręgu europejskich naukowców, by w początkach lat osiemdziesiątych stać się niemal światowym przebojem, gdy został nagłośniony japoński projekt „komputerów piątej generacji", w którym Prolog odgrywał kluczową rolę. Choć projekt ten nie okazał się sukcesem, paradygmat programowania w logice zyskał wielu gorących zwolenników, a Prolog stał się sztandarowym reprezentantem tego podejścia. W omawianym podejściu zapisujemy relacje i formuły logiczne, których prawdziwość jest weryfikowana w trakcie wykonywania programu. Programowanie w logice i programowanie funkcyjne mają na tyle dużo wspólnych cech, że pojawiają się języki, które z założenia należą do tych obu klas – koronnym przedstawicielem jest tu stosunkowo nowy język – **Mercury** (opracowany i rozwijany na Uniwersytecie w Melbourne).

Niezwykle ostatnio popularny paradygmat – programowanie obiektowe – ma swoje korzenie zarówno w licznych pracach związanych z modularyzacją, jak i w języku programowania **Simula 67**, którego twórcami byli norwescy naukowcy: Ole-Johan Dahl, Bjørn Myhrhaug i Krysten Nygaard. Trzeba jednak stwierdzić, że oryginalnie Simula stanowił rozszerzenie języka Algol 60 o mechanizmy symulacji. Jednakże jest to pierwszy język, w którym pojawiają się typowe dla programowania obiektowego pojęcia, takie jak obiekt oraz klasa. Cechą charakterystyczną tego stylu programowania jest daleko idąca modularyzacja z „ukrywaniem sekretów" (zwana hermetyzacją lub enkapsulacją), grupowanie obiektów w klasy z możliwością dziedziczenia oraz polimorfizm. Pierwszym językiem jawnie projektowanym jako obiektowy był **Smalltalk**, opracowany w latach siedemdziesiątych ubiegłego wieku w laboratorium naukowym firmy Xerox PARC, dzięki któremu pojawiły się tak powszechne obecnie rozwiązania, jak systemy okienkowe. Mamy też polski akcent – język **Loglan** (lata siedemdziesiąte i późniejsze; zespół Andrzeja Salwickiego z Uniwersytetu Warszawskiego) oraz cała gama innych języków, jak np. (alfabetycznie): C++, C#, Delphi, Eiffel, Java, JavaScript, Oberon, Oclam, Perl, PHP, Python, Visual Basic.

Ostatnim paradygmatem, który wszakże jest niekiedy wymieniany przy okazji innej klasyfikacji języków programowania, to programowanie współbieżne. Dotyczy ono sytuacji, w której algorytmy zapisujemy w postaci zbioru procesów, mogących ze sobą współpracować, ale konkurując przy tym o pewne zasoby. Mamy więc tutaj nie tylko klasyczne problemy do rozwiązania w ramach każdego z procesów, ale dodatkowo cały nowy bagaż kłopotów wynikających z konieczności synchronizacji współbieżnych procesów, zapewniającej ich bezpieczną współpracę i terminową realizację. Modelem wielu konkretnych rozwiązań językowych jest klasyczna propozycja Hoare'a z 1978 roku: **CSP** (***C**ommunicating **S**equential **P**rocesses*), na której bazował w szczególności język Occam. Do najbardziej znanych języków programowania współbieżnego należy niewątpliwie **Ada**, ale w klasie tej jest jeszcze wiele innych, gdyż praktycznie każdy nowoczesny język programowania ma jakieś mechanizmy do wyrażania współbieżności (podobnie jak i obiektowości).

Paradygmaty podzieliły więc języki programowania na kilka klas. Ale nie jest to jedyny możliwy podział. Często spotyka się na przykład klasyfikację ze względu na zastosowania: języki do obliczeń naukowych

(np. Fortran), języki do przetwarzania danych (np. Cobol), języki do programowania systemowego (np. C), języki czasu rzeczywistego (np. Ada), języki specjalistyczne (np. Apt), języki symulacyjne (np. Simula 67) itd. Przy czym zarówno kryteria podziałów nie są na tyle ostre, aby klasy według różnych podziałów nie nachodziły na siebie, jak i są języki, które bądź od początku należą do paru klas (np. Ada pasuje do kilku z nich), bądź w miarę upływu czasu i powstawania nowych wersji nabierają nowych cech (np. dodawana obiektowość). Niektóre języki były projektowane od początku z myślą o kompilacji (np. język Pascal), a niektóre z myślą o interpretacji (np. Python) – te ostatnie nazywa się zwykle językami interakcyjnymi lub konwersacyjnymi.

Jak więc widać mamy wiele różnych, często na siebie nachodzących klasyfikacji, a ponadto rzadko który język występuje w tak „czystej" postaci, że pasuje do dokładnie jednej klasy. Do tego trzeba pamiętać o ewolucji języków – wersja pierwotna i ta po kilkunastu lub kilkudziesięciu latach mają niekiedy niewiele, poza nazwą, wspólnego.

2.5 CZTERDZIEŚCI LAT MINĘŁO...

Lata sześćdziesiąte i siedemdziesiąte ubiegłego stulecia to istna eksplozja nowych języków programowania wysokiego poziomu wraz z różnymi próbami ich precyzyjnego opisu (por. artykuł Petera Landina z marca 1966 roku o znamiennym tytule *The next 700 programming languages*, [PL 66]). Nim zajmiemy się Pascalem, który stanowił swego rodzaju fenomen, wspomnimy krótko o kilku innych ważnych – z różnego punktu widzenia – językach.

- **APL** (***A** **P**rogramming **L**anguage*) – początkowo była to tylko specjalna notacja, opracowana około 1960 roku przez Kena Iversona z Harvardu, do zwięzłego opisywania algorytmów, w których występuje rachunek na macierzach. Z czasem przerodziła się w konwersacyjny język programowania o dość szerokich zastosowaniach, ale przy tym o bardzo udziwnionej i mało czytelnej składni. APL jest nadal rozwijany, mając zagorzałych zwolenników w niektórych środowiskach naukowych.
- **PL/I** (***P**rogramming **L**anguage **I***) – język ten, opracowany przez George'a Radina z IBM w 1964 roku, miał być uniwersalnym „lekarstwem na wszystko" dzięki wykorzystaniu w nim najlepszych pomysłów z języków: Fortran, Algol 60 i Cobol. Co więcej, był to pierwszy w historii język z formalnie opisaną semantyką (przy użyciu notacji VDL – ***V**ienna **D**efinition **L**anguage*). Mimo wielu ciekawych rozwiązań PL/I nigdy nie uzyskał powszechnej akceptacji i nie zdobywszy popularności „zmarł śmiercią naturalną".
- **Basic** (***B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode*) – pomyślany jako łatwy język dla nowicjuszy, do szybkiego nauczenia się prostego programowania. Był zaprojektowany przez Johna G. Kemeny'ego i Thomasa E. Kurtza z Dartmouth College, a jego pierwsza realizacja – na komputerze... IBM 704 – została ukończona w maju 1964. Wraz z pojawieniem się mikrokomputerów Basic stał się bardzo popularny i powszechnie używany, zwłaszcza przez dzieci i młodzież, choć jego walory dydaktyczne były często podważane. Język ten nadal jest wykorzystywany, a dokładniej jego różne dialekty (mające coraz mniej wspólnego ze swoim protoplastą). Szczególnie popularna jest wersja obiektowa z 1987 roku, Visual Basic, stanowiąca zaawansowane i nowatorskie środowisko programistyczne.
- **Logo** – to nie tylko język, ale cała filozofia podejścia do programowania przyjęta przez pedagogów szkolnych na całym świecie (grafika żółwia). Jej twórcą jest naukowiec z MIT, Seymour Papert, choć przy samym języku (lata 1966-1968) pracowało jeszcze kilka innych osób. Logo pozwala w sposób naturalny i intuicyjny przyswajać trudne i ważne pojęcia (jak np. procedury, rekurencję, strukturalne programowanie), przez co stało się naturalnym kandydatem do szkolnej nauki programowania. Język ten jest nadal popularny, choć na niższych niż dawniej etapach edukacji.
- **Algol 68** – język opracowany w latach 1965-1968 (przez międzynarodowy zespół pod kierunkiem holenderskiego matematyka i informatyka Aada van Wijngaardena), który uzyskawszy „pieczątkę" Międzynarodowej Federacji Przetwarzania Informacji (IFIP) jako oficjalny następnik języka Algol 60, nie spełnił jednak oczekiwań i mimo różnych prób jego upowszechnienia zniknął

z pola widzenia. Trzeba jednak lojalnie przyznać, że Algol 68 wniósł wiele do rozwoju metodyki programowania, języków programowania oraz metod ich opisu (por. np. gramatyki dwupoziomowe van Wijngaardena).

- **C** – najkrócej mówiąc, jest to język opracowany przez programistów dla programistów. Był zainspirowany językiem **BCPL** (***B**asic **C**ombined **P**rogramming **L**anguage*) i powstał w roku 1972 w celu ponownego zaprogramowania systemu operacyjnego Unix (dla uzyskania przenośności tego systemu pomiędzy różnymi architekturami). Głównym twórcą języka C był Dennis Ritchie z firmy Bell Labs. Jak wiadomo, ten język osiągnął nadspodziewany sukces i jest do dzisiaj powszechnie stosowany na całym świecie, zarówno do programowania różnych aplikacji, jak i w dydaktyce (pomimo jego istotnych ułomności z tego punktu widzenia). Zapoczątkował on całą linię języków, w tym bardzo popularną wersję obiektową **C++** oraz szlagier ostatnich lat – **C#** (*C sharp*, czyli muzyczne *cis*).

2.6. ERA JĘZYKA PASCAL

W latach sześćdziesiątych zeszłego wieku, jak wspominaliśmy w poprzednim rozdziale, podjęto wiele prób opracowania nowego języka programowania licząc, iż powstanie produkt uniwersalny, powszechnie zaakceptowany. Niektóre z tych projektów miały przy tym wsparcie instytucjonalne (np. PL/I – firma IBM, Algol 68 – IFIP) oraz powstawały przy udziale zespołu specjalistów. I nie przyniosło to oczekiwanego rezultatu.

Tymczasem w 1971 roku szwajcarski informatyk Niklaus Wirth przedstawił w naukowym czasopiśmie nowy język programowania o nazwie **Pascal** (kontynuując zainicjowany przez siebie kilka lat wcześniej taki właśnie sposób honorowania wybitnych postaci historycznych). Język ten wyrósł z prac Wirtha nad następnikiem języka Algol 60 oraz uwzględniał najnowsze wówczas trendy w metodyce programowania. Prawie jednocześnie z tą publikacją powstała też w środowisku Wirtha pierwsza implementacja tego języka dla komputera CDC 6000.

Język Pascal oryginalnie był pomyślany jako notacja wspomagająca nauczanie programowania w systematyczny sposób. Związana z nim filozofia podejścia do programowania (np. unikanie instrukcji skoku, którą Wirth pozostawił wyłącznie „ze strachu”, znając powszechne wówczas nawyki wśród programistów), prostota i klarowność języka oraz pojawienie się w nim zalążków abstrakcyjnych struktur danych, pozwalały wprowadzić nową jakość do metodyki programowania. Wkrótce okazało się, że Wirth trafił w dziesiątkę – język Pascal stał się standardem na całym świecie, który nie tylko dominował w środowisku akademickim, ale i sprawdzał się jako praktyczne narzędzie wykorzystywane do komercyjnych zastosowań. Efektywne kompilatory tego języka – a także towarzyszące im inne narzędzia wspomagające programowanie – były dostępne na praktycznie każdą architekturę, z mikrokomputerami włącznie. Pojawiały się nowe, ulepszone i rozszerzone wersje języka, w różnych paradygmatach programowania (by wspomnieć tylko Concurrent Pascal, Simpascal, Borland Pascal, Delphi).

Pomimo iż pojawiały się nowocześniejsze języki, a w szczególności sam Wirth przedstawił kilka kolejnych propozycji (Modula, Modula-2, Oberon), era języka Pascal trwała około 30 lat – dopiero teraz pojawiają się coraz widoczniejsze oznaki, iż czas ustąpić tronu.

3 DOKĄD ZDĄŻAMY

Pełniejsze rozważania na ten temat powinny być przedmiotem głębszych przemyśleń i oddzielnego opracowania. Poniżej zamieścimy więc tylko bardzo skrótowe refleksje, raczej sygnalizując trendy niż je omawiając.

3.1 JAVA, JAVA, JAVA

Język **Java** pojawił się w 1995 roku, pierwotnie pod nazwą Oak. Jego twórca – James Gosling z firmy Sun Microsystems – dobrze czując słabe punkty rosnącego w popularność języka C++ zaproponował jego dość istotne modyfikacje, obejmujące zarówno ograniczenia, jak i rozszerzenia. Chodziło o to, aby powstał przenośny język obiektowy, notacyjnie przypominający C++, w miarę prosty, o dużej sile wyrazu i nadający się do łatwego formułowania współcześnie rozwiązywanych problemów.

Doświadczenia ostatnich kilku lat wyraźnie wskazują na to, że ów ambitny cel został osiągnięty. Język Java i jego różne mutacje, wraz z ciągle wzbogacanym wspomagającym środowiskiem narzędziowym, zdobywa sobie coraz większą popularność, powoli przejmując kolejne obszary dotychczas okupowane przez inne języki i systemy. Na uwagę zasługuje w szczególności próba zdefiniowania specjalnego podzbioru tego języka dla celów edukacyjnych – znane amerykańskie towarzystwo naukowe ACM powołało zespół zadaniowy (ang. *task force*), który przygotował odpowiedni raport na ten temat[11]. Można oczekiwać, że jeśli wynik pracy tego zespołu zakończy się sukcesem, to Java – tak jak wcześniej język Pascal – stanie się światowym standardem w edukacji.

3.2 NOWE WYZWANIA

Jeszcze kilka lat temu mało kto przewidywał iż telefony komórkowe nie tylko staną się powszechne, ale i będą się scalały z innymi urządzeniami, tworząc swego rodzaju „multimedialne kombajny”. To tylko jeden z przykładów świadczących o tym, że tempo rozwoju technologii teleinformatycznych przerosło nasze najśmielsze oczekiwania.

A za tym rozwojem kryją się oczywiście nowe potrzeby informatyczne i nowe wyzwania dla całego sztabu specjalistów z kolejnych faz procesu wytwórstwa oprogramowania, w tym programistów. Potrzebne są więc języki do wygodnego zapisywania innego rodzaju algorytmów, na innym poziomie abstrakcji, dotyczących nowych zastosowań i takie powstają – albo przez rozszerzenie tradycyjnych, albo jako nowe propozycje. Przykładem niech tu będą języki skryptowe (PHP lub JavaScript) do projektowania witryn internetowych, w tym tworzenia dynamicznych stron WWW – oczywiście kilkanaście lat temu nikt o takiej potrzebie nawet nie śnił!

Natomiast pozornie tylko nowym wyzwaniem jest kwestia komponentów, czyli próby wielokrotnego wykorzystywania dobrze określonych i już sprawdzonych w praktyce „kawałków oprogramowania” oraz konstruowania systemów z takich gotowych elementów. Jeśli się chwilę zastanowimy, to dojdziemy do wniosku, że nie jest to ani nowa potrzeba, ani nowy pomysł. Każdy program jest przecież zbudowany z takich sprawdzonych kawałków – chodzi tylko o ich granulację: wielkość i funkcjonalność. Już biblioteki podprogramów (standardowych i niestandardowych) w pierwszych językach wysokiego poziomu stanowią dobre przykłady komponentów. Jeszcze wyraźniej to widać w koncepcji paradygmatu programowania obiektowego – modularyzacja z hermetyzacją służy właśnie temu, aby powstawały coraz bardziej wyspecjalizowane (ale i sparametryzowane) „cegiełki” do w miarę łatwego składania z nich większych systemów.

ZAKOŃCZENIE

Jak wspomnieliśmy na początku, problematyką języków zajmują się różne dziedziny naukowe, z językoznawstwem na czele. A w zakresie samych języków programowania, to problematyce ich ewolucji można poświęcić tomy – niniejsze opracowanie jest tylko krótkim wprowadzeniem do tej interesującej i obszernej tematyki.

Mamy jednak nadzieję, że Czytelnik podziela teraz w pełni myśl przewodnią:
język językowi nie równy

[11] Por: http://www-cs-faculty.Stanford.edu/~eroberts/java/special-session.pdf.

BIBLIOGRAFIA

[JM04] Madey J., *Czy to sen, czy Java? O ewolucji języków programowania*, w: M.M. Sysło (red.), *Materiały z Konferencji „Informatyka w Szkole XX”*, Wrocław 2004, s. 6-16
[DD72] Diinger D., *Alfabet, czyli klucz do dziejów ludzkości*, PIW, Warszawa 1972
[SJP78] Szymczak M. (red. nauk.), *Słownik języka polskiego*, PWN, Warszawa 1978
[WEPl] Granowski F. J. i Sikorski S. J. (wyd.), *Wielka Encyklopedia Powszechna Ilustrowana,* 1880 – 1914 (55 tomów, przerwana na haśle „Patroklos”)
[WEP65] *Wielka Encyklopedia Powszechna PWN*, PWN, Warszawa 1965
[WIEM] http://wiem.onet.pl/
[Wikipedia] http://pl.wikipedia.org/wiki/
[MMHW1] Marcotty M., Ledgard H., *W kręgu języków programowania*, WN-T, Warszawa 1991
[SR96] Sethi R., *Programming Languages. Concepts and Constructs.* Addison-Wesley, Reading 1996
[VOEEV68] Vaccari O., Vaccari E. E., *Pictorial Chinese-Japanese Characters*, Vaccari’s Language Institute, Tokio 1968
[PL66] Landin P., *The next 700 programming languages*, „Communication of the ACM” 1966, Vol. 9(3)
http://www-cs-faculty.Stanford.edu/~eroberts/java/special-session.pdf

Programowanie współbieżne w informatyce i nie tylko

Marcin Engel

Instytut Informatyki, Uniwersytet Warszawski

mengel@mimuw.edu.pl

Streszczenie

Program współbieżny to zestaw wykonujących się w tym samym czasie „zwykłych'' programów. Techniki współbieżne stosuje się przy tworzeniu wielu współczesnych programów, na przykład opracowując interfejs użytkownika, programując gry czy aplikacje sieciowe. Tworzenie programów współbieżnych wymaga od programisty większej dyscypliny i wyobraźni, niż pisanie programów sekwencyjnych. Oprócz zagwarantowania poprawności poszczególnych składowych programu współbieżnego, trzeba jeszcze dobrze zsynchronizować ich działanie oraz przewidzieć wszystkie możliwe scenariusze wykonania. Nie jest to łatwe – przekonamy się, jak często podczas analizowania programów współbieżnych może zawieść nas intuicja!

W trakcie zajęć przedstawimy podstawowe pojęcia programowania współbieżnego. Zdefiniujemy pojęcie procesu i wyjaśnimy, jak mogą być wykonywane programy współbieżne. Powiemy także, jak współczesne systemy operacyjne radzą sobie z wykonywaniem wielu zadań na jednym procesorze. Na przykładzie dwóch klasycznych problemów współbieżności: wzajemnego wykluczania oraz pięciu filozofów omówimy pojęcia związane z analizą programów współbieżnych: przeplot, poprawność, bezpieczeństwo oraz żywotność. Przekonamy się, że z tymi pojęciami oraz problemami synchronizacyjnymi spotykamy się na co dzień, na przykład ucząc się, piekąc ciasto albo obserwując ruch samochodów na ulicach.

Zajęcia będą miały formę wykładu, ale w jego trakcie będziemy wspólnie uruchamiać niektóre programy współbieżne na „wirtualnym komputerze wieloprocesorowym", którego procesorami będą słuchacze.

Spis treści

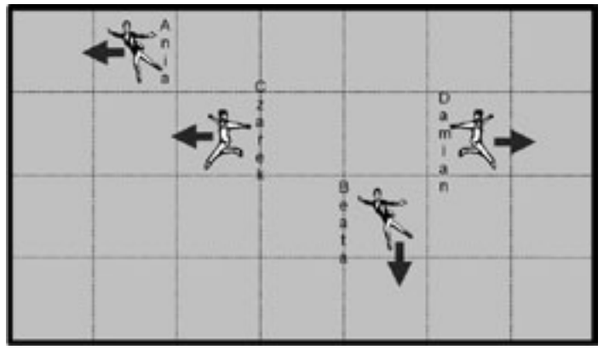
1 CO TO JEST PROGRAMOWANIE WSPÓŁBIEŻNE

1.1 MODEL KOMPUTERA

Na rysunku 1 przedstawiono obrazek z popularnej gry. Widzimy na nim kilka poruszających się niezależnie od siebie łyżwiarzy. Zastanówmy się, w jaki sposób można zaprogramować taką scenę.

Rysunek 1.
Obrazek z popularnej gry Sims 2

Aby się o tym przekonać poczyńmy pewne upraszczające założenia. Przyjmijmy, że lodowisko jest prostokątem i że jest podzielone na pola. Znajdują się na nim cztery osoby. Każda z nich stoi początkowo nieruchomo w pewnym polu lodowiska zwrócona twarzą w kierunku wskazanym strzałką jak na rysunku 2.



Rysunek 2.
Uproszczony schemat gry

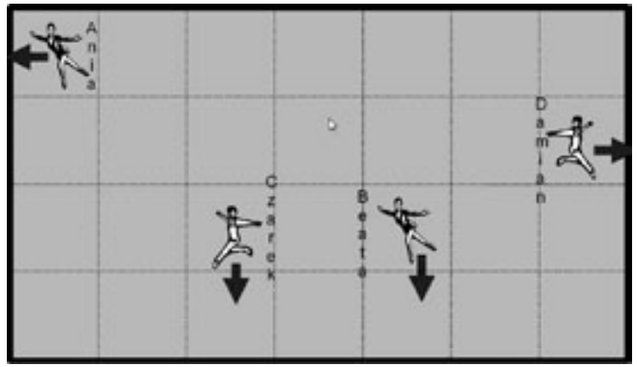
Załóżmy, że każda z tych osób potrafi:
- wykonywać jeden krok do przodu przechodząc do kolejnego pola,
- obracać się o 90^{o} w prawo lub w lewo.

Przy lodowisku stoi trener i nadzoruje ruch łyżwiarzy, którzy nie wykonują żadnego kroku ani obrotu bez jego wyraźnego polecenia. Lodowisko ze znajdującymi się na nim łyżwiarzami będzie stanowić model komputera, a trener gra rolę programisty.

Trener wydaje łyżwiarzom polecenia, na przykład: „Beata, wykonaj jeden krok naprzód”, „Czarek, obróć się w lewo”. Możemy myśleć o takich poleceniach jak o instrukcjach pewnego języka programowania. Przyjmijmy, że każda instrukcja składa się z dwóch elementów: imienia łyżwiarza oraz polecenia dla tego łyżwiarza, które może być jednym z: krok naprzód, obrót w lewo, obrót w prawo. Instrukcje są wykonywane na przedstawionym modelu komputera powodując przemieszczanie się łyżwiarzy. Przykładowy program w tym języku programowania może mieć następującą postać:

Ania, krok naprzód
Czarek, obrót w lewo
Czarek, krok naprzód
Damian, krok naprzód

Jeżeli wykonamy ten program w sytuacji początkowej, przedstawionej na rysunku 2, to doprowadzi on do końcowego położenia łyżwiarzy jak na rysunku 3.



Rysunek 3.
Pozycja łyżwiarzy po wykonaniu prostego programu

Jazda na łyżwach bywa niebezpieczna. Łyżwiarze powinni uważać, żeby nie wpaść na bandę ani nie zderzyć się ze sobą. Ponieważ jednak wykonują oni polecenia trenera, więc to on odpowiada za wszelkie wypadki. Ta sytuacja również dobrze modeluje rzeczywistość: komputery popełniają błędy, ale zawsze są one skutkiem błędu programisty.

1.2 PROGRAM SEKWENCYJNY

Budowanie całej sceny za pomocą takich prostych poleceń wydawanym poszczególnym łyżwiarzom jest bardzo pracochłonne. Wygodniej byłoby przygotować krótką choreografię dla każdego łyżwiarza i kazać mu ją powtarzać. Jednak ze względu na bezpieczeństwo, trener nie może tego zrobić bez dodatkowych narzędzi. Spróbujmy więc ułatwić mu zadanie wprowadzając do języka jeszcze jedną instrukcję: „jeśli pole przed Tobą jest wolne, to wykonaj krok naprzód, w przeciwnym razie obróć się w lewo”. Teraz choreografia (czyli program dla łyżwiarzy) może wyglądać następująco:

Powtarzaj
    Ania, jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo
    Beata, jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo
    Czarek, jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo
    Damian, jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo

Nazwiemy go **programem sekwencyjnym**, bo poszczególne instrukcje są wykonywane jedna po drugiej. Jeśli program ten powtórzymy dużą liczbę razy, to uzyskamy scenę z czterema jeżdżącymi wokół lodowiska łyżwiarzami.

Takie rozwiązanie ma jednak pewne wady. Po pierwsze wszyscy łyżwiarze poruszają się w ten sam sposób. Trudno byłoby nam zapisać w postaci równie krótkiego co poprzednio programu scenę, w której trójka łyżwiarzy jeździ wokół lodowiska, a Beata kręci piruet. Po drugie, wszyscy łyżwiarze jeżdżą w tym samym tempie. Z tych powodów scena nie jest realistyczna. Wreszcie programista przygotowujący program musi jednocześnie myśleć o wszystkich łyżwiarzach. Spróbujmy więc nieco innego podejścia do wyreżyserowania scenki.

1.3 PROGRAM WSPÓŁBIEŻNY

Przypuśćmy teraz, że trener przygotowuje program dla jednego łyżwiarza. Program jest napisany w tym samym języku programowania, co do tej pory, choć teraz nie trzeba poprzedzać polecenia imieniem konkretnej osoby. Przykładowy program dla jednego łyżwiarza wygląda teraz tak:

Powtarzaj
    jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo

Następnie trener każe wszystkim łyżwiarzom wykonywać programy, które od niego otrzymali. Jeśli wszyscy otrzymali powyższy program, to efekt będzie identyczny (prawie identyczny, ale o tym później), jak w przypadku programu sekwencyjnego. Na czym więc polega różnica?

W przypadku programu sekwencyjnego komputer (czyli lodowisko) wykonywał w danym momencie tylko jedno polecenie. „Czarek, krok naprzód” powodowało, że tylko Czarek wykonywał ruch. Innymi słowy na komputerze wykonywał się tylko jeden program. W drugim rozwiązaniu sytuacja jest zupełnie inna. Trener przygotowuje cztery programy i wręcza je łyżwiarzom, którzy wykonują je jednocześnie. Takie właśnie wykonanie nazywamy **współbieżnym**. Dokładniej, wykonanie współbieżne to realizacja kilku (co najmniej dwóch czynności) w taki sposób, że kolejna czynność rozpoczyna się przed zakończeniem poprzedniej.

Co w opisanej sytuacji zyskujemy pisząc zamiast „zwykłego” programu sekwencyjnego program współbieżny? Przede wszystkim elastyczność, czyli możliwość łatwej modyfikacji zachowania jednego łyżwiarza niezależnie od pozostałych. Jeśli chcemy, żeby Beata kręciła piruety, po prostu wręczymy jej inny program do wykonania, na przykład taki: „powtarzaj: wykonaj obrót w lewo”. Przygotowując program trener może teraz skupić się w danym momencie na roli jednego łyżwiarza, nie przejmując się pozostałymi. To bardzo ważna zaleta! Współczesne programy komputerowe są tworami bardzo złożonymi. Ich tworzenie byłoby niemożliwe, gdyby programista musiał myśleć jednocześnie o wszystkich szczegółach. Dlatego właśnie duże programy tworzy się stopniowo skupiając się na jednym problemie, chwilowo ignorując pozostałe, lub też zakładając, że znane są już ich rozwiązania.

1.4 PROGRAMY I PROCESY

Kontynuujmy przykład lodowego komputera. Zastanówmy się teraz, w jaki sposób taki komputer może zrealizować program współbieżny. Zanim to jednak zrobimy wprowadźmy pewne ważne i powszechnie stosowane w informatyce odróżnienie. Informatycy wyraźnie odróżniają pojęcie programu od wykonania programu. **Program** to przepis, ciąg instrukcji, który ma zostać przeprowadzony. W naszym przykładzie programami były scenariusze zapisane na kartkach wręczanych łyżwiarzom. **Wykonanie programu** to obiekt dynamiczny zwany **procesem**. W naszym przykładzie każdy łyżwiarz to osobny proces. Każdy proces wykonuje pewien program. Dwa różne procesy mogą wykonywać różne programy, mogą też wykonywać ten sam program. Konkretny program może być w danej chwili realizowany przez wiele procesów lub przez jeden proces. Każdy proces pracuje **sekwencyjnie**, tzn. wykonuje swoje instrukcje jedna po drugiej. W wykonaniu współbieżnym, zgodnie z tym co powiedzieliśmy poprzednio, bierze udział kilka procesów.

1.5 RÓŻNE SPOSOBY WYKONYWANIA PROGRAMU WSPÓŁBIEŻNEGO

W jaki zatem sposób procesy realizują swoje programy? Pierwsza możliwość to wykonanie **asynchroniczne**. Wyobraźmy sobie, że trener przygotował programy dla poszczególnych łyżwiarzy i wręczył im je. Łyżwiarze realizują następnie te programy w sposób niezależny od siebie, każdy we własnym tempie. Dzięki temu możemy uzyskać realnie wyglądającą scenę. Druga możliwość to wykonanie **synchroniczne**. Aby wytłumaczyć, na czym ono polega, zaangażujemy jeszcze jedną osobę – kierownika lodowiska. Łyżwiarze po otrzymaniu programów nie robią nic, czekając na sygnał od kierownika. Na sygnał (klaśnięcie w dłonie) każdy łyżwiarz wykonuje jedną instrukcję swojego programu. Tak przygotowana scena bardzo przypomina tę z programu sekwencyjnego: wszyscy łyżwiarze poruszają się w tym samym tempie.

Są także inne możliwości. Przypuśćmy, że kierownik tym razem nie klaszcze, ale wykrzykuje po kolei imiona poszczególnych łyżwiarzy. Każdy z nich po usłyszeniu własnego imienia wykonuje jedną instrukcję swojego programu. Kierownik może wywoływać cyklicznie wszystkich łyżwiarzy – wtedy uzyskujemy wykonanie przypominające wykonanie synchroniczne – albo może za każdym razem podejmować decyzję, który łyżwiarz ma się poruszyć niezależnie od tego, co działo się do tej pory. Powstaje wtedy scenka przypominająca wykonanie asynchroniczne.

Przyjrzyjmy się, czym różnią się te dwie ostatnie możliwości od wykonania synchronicznego i asynchronicznego. Pierwsza najważniejsza różnica jest taka, że w wykonaniu synchronicznym i asynchronicznym dochodziło do jednoczesnych działań wielu łyżwiarzy. Używając fachowego języka powiemy, że jednocześnie, czyli dokładnie w tym samym czasie, wykonuje się wiele procesów. Mówimy wówczas, że jest to wykonanie **równoległe**. W rozwiązaniu z kierownikiem wybierającym zawodnika, który ma wykonać ruch w danej chwili działa tylko jeden proces. W dalszym ciągu jednak jest to wykonanie współbieżne, gdyż działa wiele procesów i kolejny rozpoczyna się zanim zakończy się pierwszy. Nie jest ono jednak równoległe, bo w danej chwili wykonuje się tylko jeden proces. O takim wykonaniu powiemy, że jest to wykonanie **w przeplocie**. Nazwa pochodzi stąd, że instrukcje poszczególnych procesów przeplatają się ze sobą. Sposób przeplotu może być zawsze taki sam, jeśli kierownik zawsze wywołuje łyżwiarzy w tej samej kolejności, lub też za każdym razem inny, jeśli kierownik wywołuje łyżwiarzy w losowej kolejności. Pojęcie przeplotu jest bardzo ważne w programowaniu współbieżnym i często będziemy do niego wracać na tych zajęciach.

Co jeszcze rzuca się w oczy, gdy porównujemy wykonania równoległe z wykonaniami w przeplocie. Widać, że jeśli łyżwiarze poruszają się w przeplocie uzyskujemy scenę mniej płynną i realną niż przy wykonaniu równoległym. Ale jeżeli kierownik będzie wywoływał łyżwiarzy bardzo szybko i będą oni szybko wykonywać swoje ruchy, nasze oczy przestaną zauważać różnicę między wykonaniem równoległym a wywołaniem w przeplocie i scena odzyska płynność. Taką właśnie technikę bardzo częstego przeplatania instrukcji poszczególnych procesów stosuje się we współczesnych systemach operacyjnych.

Tak naprawdę z wykonaniem współbieżnym zarówno równoległym, jak i w przeplocie spotykamy się bardzo często w życiu codziennym. Przykładowo wszystkie zgromadzone na sali wykładowej osoby możemy traktować jak procesy realizujące pewne programy i wykonujące się równolegle. Ciekawym przykładem procesów wykonujących się równolegle są samochody przejeżdżające przez skrzyżowanie. Z wykonaniem w przeplocie spotyka się każdy z nas realizując swoje codzienne obowiązki. Zazwyczaj mamy wiele rzeczy do zrobienia (na przykład uczeń musi przyswoić wiedzę z wielu przedmiotów). Mózg ludzki nie jest przystosowany do nadzorowania wielu czynności jednocześnie, więc czasochłonne czynności z reguły dzielimy na etapy i przeplatamy z innymi obowiązkami. Przykładowo uczeń nie uczy się matematyki dopóki nie opanuje całego materiału szkoły średniej, ale przeplata naukę matematyki nauką historii, polskiego czy innych przedmiotów. W podobny sposób postępuje kucharz przygotowujący obiad złożony z wielu dań.

1.6 ZNACZENIE PROGRAMOWANIA WSPÓŁBIEŻNEGO

Dlaczego programowanie współbieżne stało się obecnie ważną techniką programowania? Kiedy 15 lat temu przeciętny użytkownik uruchamiał komputer, jego oczom ukazywał się mniej więcej taki sam widok, powszechnie stosowanym wówczas systemem operacyjnym był MS-DOS firmy Microsoft. Komputer oczekiwał na polecenie użytkownika, którym mogło być na przykład uruchomienie określonego programu. Proces wykonujący ten program musiał wykonać się do końca zanim użytkownik mógł uruchomić kolejny. Taki system operacyjny nazywa się **systemem jednozadaniowym**. Zatem przeciętny użytkownik nie miał możliwości współbieżnego wykonywania programów, więc programiści przygotowujący aplikacje pod system MS-DOS nie musieli (a nawet nie mogli) stosować technik programowania współbieżnego. Nie znaczy to jednak, że techniki te nie były znane lub niepotrzebne. W tym samym czasie istniały również systemy umożliwiające uruchamianie wielu programów, jednak przeznaczone one były na większe komputery. Powodowało to, że umiejętność programowania współbieżnego była dość ezoteryczna i zarezerwowana dla programistów systemowych (tworzących systemy operacyjne) i piszących aplikacje na duże maszyny.

Od tego czasu jednak wiele się zmieniło. Przede wszystkim nastąpił znaczący postęp w dziedzinie sprzętu. Porównajmy dla przykładu parametry typowego współczesnego komputera i komputera sprzed 15 lat. Szybszy procesor umożliwia szybsze wykonanie procesów. Każdy proces, który jest wykonywany przez komputer, musi mieć zarezerwowaną pamięć na swoje dane. Im więcej pamięci tym więcej procesów można utworzyć, a szybki procesor umożliwia szybkie, niezauważalne dla użytkownika przeplatanie ich wykonania.

Rozwój sprzętu to jednak nie wszystko. Tak naprawdę już kilkanaście lat temu na komputerze PC, który wówczas był zazwyczaj wyposażony w procesor 16 MHz, pamięć 1 MB i dysk twardy o pojemności 60 MB, można było wykonywać wiele programów współbieżnie. Potrzebny był jedynie system operacyjny, który by to umożliwiał – tzw. **system wielozadaniowy**. I takie systemy zaczęły się powoli pojawiać, a jednym z pierwszych był Linux.

Jak wygląda sytuacja dzisiaj? Po uruchomieniu komputera widzimy graficzny interfejs jednej z wielu wersji systemu Windows, Linux lub innych systemów. Po chwili pracy najczęściej na ekranie jest otwartych wiele okienek, na przykład przeglądarka internetowa, edytor tekstu, arkusz kalkulacyjny, kalkulator i inne. Nie trzeba już zamykać jednego programu, aby uruchomić kolejny. Współbieżne wykonanie wielu programów jest czymś powszechnym. Co więcej, nawet jedna aplikacja, na przykład arkusz kalkulacyjny, może składać się z wielu procesów. Często jeden z nich jest odpowiedzialny za obsługę interfejsu użytkownika. Gdy użytkownik zleci za pomocą myszki wykonanie jakiejś czynności, to proces ten tworzy nowy proces i zleca mu wykonanie polecenia użytkownika, a sam jest gotowy na przyjmowanie kolejnych poleceń. Dzięki takiej konstrukcji oprogramowania aplikacja może reagować na polecenia użytkownika nawet w czasie realizacji czasochłonnych obliczeń!

Kolejnym elementem, który powoduje, że obecnie każdy programista musi znać podstawowe techniki programowania współbieżnego to rozwój sieci, w tym także Internetu.

1.7 JAK KOMPUTERY WYKONUJĄ PROGRAMY WSPÓŁBIEŻNE

Odnieśmy teraz różne wykonania programu współbieżnego, przedstawione na przykładzie lodowiska, do wykonania na prawdziwych komputerach. Jak wiadomo sercem każdego komputera jest **procesor**. To właśnie procesor jest odpowiedzialny za przeprowadzenie poszczególnych instrukcji. Tradycyjnie skonstruowany procesor jest w stanie dokonać w danej chwili tylko jedną instrukcję. Natychmiastowym wnioskiem z tego jest fakt, że komputer może wykonywać równolegle tyle instrukcji, w ile procesorów jest wyposażony. Typowe komputery domowe mają jeden procesor co oznacza, że realizowanie równoległe nie jest możliwe. (Tak naprawdę nawet w takich komputerach pewne czynności są wykonywane równolegle. Przykładowo karty graficzne mają odrębne procesory, które wykonują obliczenia równolegle z procesorem centralnym. Jednak z perspektywy programisty nie ma to najczęściej znaczenia, dlatego nie rozważamy tego). Ostatnio coraz powszechniejsze nawet w zastosowaniach domowych stały się jednak **procesory wielordzeniowe**. Są to procesory, które są w stanie wykonywać wiele rozkazów maszy-

nowych jednocześnie – tyle, ile mają rdzeni. Komputery wyposażone w takie procesory potrafią więc to samo. Na nich wykonanie równoległe jest w pełni możliwe. Jeszcze inny typ komputerów, na razie raczej niespotykanych w zastosowaniach domowych, to komputery z wieloma procesorami. Różnica między komputerem z jednym procesorem wielordzeniowym a komputerem wieloprocesorowym jest dla potrzeb tego wykładu nieistotna, ważne jest jedynie, że komputery wieloprocesorowe także mogą wykonywać wiele procesów równolegle.

No dobrze, ale nawet na komputerze z jednym procesorem można uruchomić wiele procesów. Jak to się dzieje? Odpowiedzią jest właśnie wykonanie w przeplocie. Funkcję kierownika wybierającego proces, który ma wykonać kolejną instrukcję pełni wtedy system operacyjny, a dokładnie jego moduł zwany **modułem szeregującym**. Otóż system operacyjny zapamiętuje informacje o wszystkich uruchomionych procesach utrzymując tak zwaną **kolejkę procesów gotowych**. Pierwszemu procesorowi w tej kolejce jest przydzielany procesor, to znaczy, system operacyjny decyduje, że teraz będzie wykonywał się ten właśnie proces. Odpowiada to sytuacji, w której kierownik wybrał łyżwiarza, który ma wykonać kolejny krok. Proces nie wykonuje jednak tylko jednego rozkazu jak w przykładzie z lodowiska, ale wykonuje się przez pewien ustalony czas, zwany **kwantem czasu**. Po upływie kwantu czasu system operacyjny zapamiętuje stan wykonywanego procesu, umieszcza go na końcu kolejki procesów gotowych i przydziela procesor kolejnemu procesowi z kolejki. W ten sposób procesor wykonuje w danej chwili tylko jeden rozkaz naraz, przy czym jest to rozkaz jednego z procesów lub systemu operacyjnego. Gdy kwanty czasu są odpowiednio małe, to procesy są przełączane często i ich użytkownicy nie zauważają opóźnień. Taki mechanizm nazywa się **podziałem czasu**. System operacyjny z podziałem czasu gra więc rolę kierownika z naszego przykładu. W odróżnieniu od dotychczasowego naszego modelu, po wywołaniu łyżwiarza włącza stoper na określony czas. W tym czasie kroki wykonuje wybrany łyżwiarz aż do chwili, gdy kierownik wywoła kolejnego łyżwiarza. Czasami mówi się, że system z podziałem czasu dostarcza **wirtualne procesory**. Taki wniosek jest uzasadniany tym, że z punktu widzenia użytkowników taki system nie różni się od systemu działającego na komputerze z wieloma, choć odpowiednio wolniejszymi procesorami.

Przedstawiony tutaj proces przełączania procesów jest mocno uproszczony. W prawdziwych systemach operacyjnych odbywa się to w bardziej złożony sposób. W systemie Linux, na przykład, nie ma jednej kolejki procesów gotowych lecz... ponad 100.

Dlaczego w systemie z podziałem czasu proces dostaje procesor na pewien czas, a nie na wykonanie pojedynczego rozkazu? Odpowiedź jest prosta. Wymiana (przełączenie) procesów trwa jakiś czas. Gdyby procesy mogły wykonać tylko jeden rozkaz, to komputer zajmowałby się głównie zapamiętywaniem stanu procesów i przełączaniem między nimi i jedynie od czasu do czasu wykonywałby jakąś pożyteczną akcję któregoś procesu.

Wiemy już, że zarówno przy wykonaniu równoległym, jak i wykonaniu z podziałem czasu mamy do czynienia ze współbieżnością. Czy jest to jednak wykonanie synchroniczne czy asynchroniczne? Odpowiedź nie jest jednoznaczna. Na poziomie sprzętowym najczęściej jest to wykonanie synchronicznie. Funkcję kierownika z naszego przykładu z lodowiska pełni tu zegar systemowy. Generuje on impulsy z określoną częstotliwością, a różne elementy sprzętowe (procesory, pamięci, magistrale) pracują w ich rytmie. Im częściej tyka zegar, tym częściej procesy wykonują rozkazy, a więc tym szybsze ich wykonanie. Z punktu widzenia programisty wykonanie jest jednak asynchroniczne. Poza tym procesory mogą być taktowane zegarami o różnych częstotliwościach albo procesy mogą otrzymywać kwanty różnej długości, nie wolno więc założyć nic o szybkościach pracy poszczególnych procesów, a co za tym idzie o liczbie wykonywanych naraz instrukcji poszczególnych procesów. Programista nie wie, na jakim komputerze będzie wykonywany jego program musi więc go tak napisać, aby działał poprawnie niezależnie od sposobu wykonania.

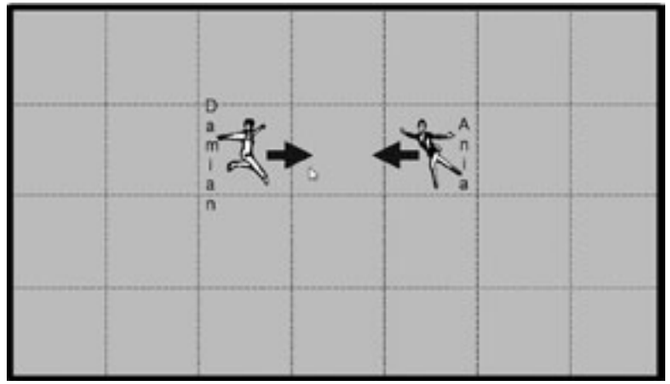
2 KŁOPOTY Z PROGRAMAMI WSPÓŁBIEŻNYMI

2.1 PROBLEMY SYNCHRONIZACYJNE

Widzieliśmy już, że wprowadzenie współbieżności może ułatwić programiście pracę, spowodować, że program będzie miał lepszą strukturę, będzie czytelniejszy, a przez to łatwiej będzie znaleźć w nim ewentualne błędy, wprowadzić nową funkcjonalność lub zmienić pewne jego elementy. Jednak współbieżne wykonanie wielu procesów wprowadza także nowe problemy – niespotykane przy programowaniu sekwencyjnym. Są to problemy synchronizacyjne i komunikacyjne. Skupimy się tutaj jedynie na wybranych problemach z pierwszej z tych grup. Umiejętność programowania współbieżnego polega w istocie na zdolności dostrzegania potencjalnych problemów wynikających z interakcji między procesami i użycia odpowiednich metod, zapobiegających wystąpieniu niepożądanej sytuacji. Przedstawmy problem na przykładzie łyżwiarzy. Rozważmy przypadek, gdy każdy łyżwiarz wykonuje własny program:

```
Powtarzaj
    jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo
```

Przyjmijmy dla potrzeb tego przykładu, że procesy wykonują się równolegle w sposób synchroniczny. Jeśli uruchomimy procesy łyżwiarzy w sytuacji przedstawionej na rysunku 3, to wszystko przebiegnie pomyślnie i łyżwiarze wkrótce zaczną jeździć wzdłuż bandy. Jeśli jednak uruchomimy go w sytuacji przedstawionej na rysunku 4, to dojdzie do wypadku.



Rysunek 4.
Sytuacja łyżwiaży prowadząca do wypadku

Każdy z łyżwiarzy na dany przez kierownika znak stwierdzi, że może wykonać krok do przodu, bo miejsce przed nim jest wolne. Łyżwiarze wykonają więc ten krok i... wpadną na siebie. Zawiodła synchronizacja procesów!

Czy do zderzenia mogłoby dojść, gdyby procesy wykonywały się równolegle i asynchronicznie? Oczywiście tak, bo wykonanie synchroniczne jest szczególnym przypadkiem wykonania asynchronicznego. A co w przypadku wykonania w przeplocie? Gdy kierownik wybierze do wykonania łyżwiarza Damiana, to ten stwierdzi, że przed sobą ma wolne miejsce i wykona krok do przodu. Gdy teraz Ania usłyszy swoje imię, to stwierdzi, że miejsce przed nią jest zajęte i obróci się w lewo. Z pozoru wszystko wygląda w porządku. Ale niestety tak nie jest.

2.2 PROBLEM Z BRAKIEM ATOMOWOŚCI INSTRUKCJI

Programy najczęściej pisze się w wysokopoziomowych językach programowania. Przykładem takiego języka jest właśnie wprowadzony przez nas język pisania scenariuszy dla łyżwiarzy (choć w rzeczywistym

języku programowania instrukcje są na znacznie niższym poziomie szczegółowości). Procesor nie rozumie jednak poleceń języka wysokiego poziomu. Producenci procesorów wyposażają je w zestaw bardzo szczegółowych rozkazów. Taki język nazywa się często **językiem maszynowym**. Rozkazy są bardzo niskopoziomowe, to znaczy, że programowanie w nich wymaga dużej wiedzy o budowie konkretnego procesora. Program w języku wysokopoziomowym jest kompilowany, czyli tłumaczony na język maszynowy i dopiero taki przetłumaczony (a ponadto jeszcze dodatkowo przygotowany) program może być wykonany na komputerze. Najczęściej jest tak, że jedna instrukcja wysokopoziomowa (np. „jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo") jest tłumaczona na wiele rozkazów maszynowych. Ponieważ procesor wykonuje rozkazy, a nie instrukcje, więc wykonanie procesu w systemie z podziałem czasu może zostać przerwane po dowolnym rozkazie, a zatem gdzieś w trakcie wykonania instrukcji języka wysokiego poziomu. Z tego wynika, że programista nie może założyć, że instrukcje, z których buduje program, są niepodzielne i wykonują się od początku do końca. W szczególności łyżwiarz wykonujący instrukcję „jeśli pole przed Tobą jest wolne, to krok naprzód, w przeciwnym razie obróć się w lewo" może uznać, że krok do przodu daje się wykonać, ale zanim go wykona inny łyżwiarz może zająć to pole (po uprzednim upewnieniu się, że jest ono jeszcze wolne).

Zatem niezależnie od przyjętego modelu wykonania analizowany przez nas program współbieżny może prowadzić do zderzenia łyżwiarzy. W praktyce programy współbieżne zawsze analizuje się tak, jakby były wykonywane w przeplocie – w dowolnym przeplocie. Nie wolno przy tym założyć nic na temat niepodzielności instrukcji języka, w którym programujemy. Okazuje się, że taki sposób analizy jest odpowiedni także dla wykonań równoległych. Jeśli więc chcemy wykazać, że program jest niepoprawny, to wystarczy znaleźć taki przeplot albo inaczej scenariusz wykonania, który prowadzi do błędnej sytuacji. Aby uzasadnić, że program jest poprawny, trzeba udowodnić, że będzie dobrze działał w każdym możliwym przeplocie. Jest to trudne, bo z reguły tych przeplotów jest nieskończenie wiele. W naszym przykładzie znaleźliśmy scenariusz prowadzący do sytuacji błędnej, więc program okazał się być niepoprawny.

Powiedzieliśmy już, że programowanie współbieżne polega na umiejętności wykrywania sytuacji prowadzących do niepożądanego zachowania programu i takim zsynchronizowaniu procesów, aby uniknąć niepożądanej sytuacji. Jednak często zachowanie programów współbieżnych jest nieintuicyjne i zdarza się, że nawet doświadczony programista nie dostrzeże źródła problemu. Co gorsza, błąd może w ogóle nie ujawnić się podczas testów! Wyobraźmy sobie, że uruchamiamy 100 razy nasz program dla łyżwiarzy i za każdym razem Damian zdąży wykonać całą instrukcję zanim ruch będzie wykonywać Ania. Wówczas błąd nie ujawni się, a programista zyska pewność, że program jest poprawny. Tymczasem uruchamiając program na innym komputerze, pod innym systemem operacyjnym, a nawet po raz 101. na tym samym komputerze może dojść do feralnego przeplotu i program zakończy się błędem. Takie sytuacje zdarzają się dość często w praktyce. W literaturze opisano na przykład błąd, który ujawnił się tuż przed pierwszym startem promu kosmicznego. Polegał on na tym, że komputer pomocniczy nie otrzymywał danych od komputera głównego. Było to o tyle dziwne, że system był uprzednio poddany intensywnym wielodniowym testom. Start opóźnił się o dwa dni, tyle czasu zajęło programistom ustalenie przyczyny błędu. Okazało się, że błąd pojawiał się jedynie wówczas, gdy moment włączenia jednego komputera trafiał w okno czasowe o szerokości 15 ms od chwili włączenia drugiego! Nic dziwnego, że testy tego nie wykryły, tym bardziej że nikt nie wyłączał komputera między kolejnymi testami!

Przykłady te pokazują, że powszechna praktyka znajdowania błędów w programach sekwencyjnych poprzez krokowe uruchamianie programu pod kontrolą specjalnego programu uruchomieniowego (ang. *debugger*) w przypadku programów współbieżnych nie sprawdza się. Po pierwsze, można nie być świadomym istnienia błędu pomimo intensywnego testowania. Po drugie, błędny scenariusz może ujawniać się bardzo rzadko i być trudny do odtworzenia, a to jest niezbędne, żeby wykonać program krok po kroku.

2.3 PROBLEM Z JEDNOCZESNĄ MODYFIKACJĄ ZMIENNYCH GLOBALNYCH

Aby jeszcze dokładniej zilustrować, jak bardzo nieintuicyjne są błędy w programach współbieżnych, rozważmy dwa rzeczywiste przykłady. Żyjemy w dobie Internetu, elektroniczne przelewy są w dzisiejszych czasach codziennością. Przypuśćmy, że program obsługujący bank, w którym mamy konto, pisał programista zafascynowany współbieżnością, ale nie do końca zdający sobie sprawę z subtelności problemów, które trzeba rozwiązać. Dla uproszczenia przyjmijmy też, że w banku znajduje się tylko nasze konto, a jego aktualny stan jest utrzymywany w zmiennej *saldo*. **Zmienna** to miejsce w pamięci komputera, w którym jest przechowywana pewna wartość, powiedzmy, że w tym przypadku 5000. Typowe języki programowania zawierają **instrukcję przypisania**, która służy do nadania zmiennej pewnej wartości. Na przykład, jeśli pobieramy z konta kwotę 1000 zł, to w programie realizującym taką operację powinno znaleźć się przypisanie zapisywane jako *saldo* := *saldo* – 1000, czyli: od aktualnej wartości zmiennej *saldo* odejmij 1000 i wynik wpisz znów do zmiennej *saldo* (będzie ona wówczas równa 4000). Podobnie przelew tysiąca złotych na nasze konto zostanie zrealizowany jak *saldo* := *saldo* + 1000. Program obsługujący konto jest współbieżny i umożliwia jednoczesne wykonywanie kilku operacji na koncie, na przykład tworząc do obsługi każdej operacji osobny proces. Jeśli teraz zdarzy się, że w tym samym czasie pojawi się zlecenie przelewu na konto tysiąca złotych i pobrania z tego konta tysiąca złotych, dojdzie do współbieżnego wykonania dwóch procesów, z których jeden zmniejsza zmienną *saldo* o 1000, a drugi zwiększa ją o 1000. I znów z pozoru wszystko jest w porządku. Jaką kwotę mamy na koncie na skutek obu tych operacji? Nie powinno się nic zmienić i w dalszym ciągu powinno to być 5000. Tymczasem okazuje się, że współbieżne wykonanie takich instrukcji przypisania może spowodować, że zmienna *saldo* ma wartość faktycznie 5000, ale równie dobrze może to być 6000 (jeśli mamy szczęście) lub 4000 (jeśli mamy pecha).

Czym jest spowodowany błąd? Otóż programista założył, że przypisanie wykonuje się w całości. Wcale tak być nie musi! Instrukcja odjęcia wartości 1000 od zmiennej *saldo* może zostać przetłumaczona na 3 rozkazy maszynowe:

załaduj *saldo* do AX
odejmij 1000 od AX
prześlij AX do *saldo*

Jeśli procesy wykonają się w całości jeden po drugim, to końcowa wartość zmiennej *saldo* będzie faktycznie równa 5000. Jeśli jednak przed przesłaniem do zmiennej *saldo* wartości AX przez pierwszy proces, drugi proces pobierze do BX wartość zmiennej *saldo* (w dalszym ciągu 5000), to wykona się tylko jedna operacja: zmniejszenie albo zwiększenie w zależności od tego, który z procesów jako drugi prześle wartość rejestru do zmiennej *saldo*. Skutki takiego „drobnego" przeoczenia programisty mogą więc być dość poważne!

W rzeczywistości dane o kontach klientów przechowywane są w bazie danych. Dostęp do takiej bazy danych może być realizowany współbieżnie, ale wszelkie modyfikacje zapisów (rekordów) w bazie są wykonywane za pomocą niepodzielnych transakcji, czyli ciągu operacji, które wykonują się jako jedna niepodzielna całość.

Podobny (i chyba jeszcze bardziej nieintuicyjny) przykład polega na wielokrotnym zwiększaniu wartości pewnej zmiennej o 1. Wyobraźmy sobie, że działają dwa procesy. Każdy z nich pięciokrotnie zwiększa wartość zmiennej *x* o 1.

Pytanie: Jeśli początkowo zmienna *x* miała wartość zero, to jaką wartość będzie miała po zakończeniu obu procesów? Narzucająca się odpowiedź to 10, bo każdy proces pięciokrotnie zwiększy *x* o 1, więc łącznie *x* zostanie zwiększone o 10. Gdy jednak przypomnimy sobie, że operacja zwiększania *x* o 1 nie musi być niepodzielna i może składać się z trzech rozkazów maszynowych jak w poprzednim przykładzie, to z łatwością dostrzeżemy, że równie dobrze końcową wartością *x* może być 5 (jeśli oba procesy będą wykonywać się „łeb w łeb", czyli na przemian po jednym rozkazie jak w wykonaniu synchronicznym). Równie łatwo spostrzeżemy,

że tak naprawdę w zależności od przeplotu końcową wartością *x* może być dowolna wartość między 5 a 10. Ale już zupełnie niezgodna z intuicją jest prawidłowa odpowiedź na postawione pytanie. Końcową wartością zmiennej *x* jest dowolna wartość między 2 a 10. Zachęcam do znalezienia przeplotu, który prowadzi do uzyskania na końcu wartości 2.

W podobną, choć nieco bardziej subtelną pułapkę związaną ze współbieżnym zwiększaniem pewnej zmiennej, wpadł jeden z moich znajomych. Napisał on pod systemem operacyjnym Linux w języku C program, w którym działało wiele procesów (a dokładniej były to wątki, które w systemie Linux nieco różnią się od procesów, choć dla nas ta różnica jest dzisiaj nieistotna). Każdy z nich korzystał z tej samej zmiennej, nazwijmy ją *x,* i zawierał instrukcję *x++*, która w języku C oznacza zwiększenie *x* o 1. Program pomimo tego, co przedstawiliśmy poprzednio, działał poprawnie. Tak się bowiem złożyło, że kompilator języka C tłumaczył tę instrukcję na jeden rozkaz maszynowy o nazwie symbolicznej inc. I wszystko było dobrze, do chwili... wymiany komputera na lepszy. Można sobie wyobrazić, jaka jest zdroworozsądkowa reakcja programisty, który po zakupie nowego sprzętu stwierdza, że program, który dotychczas działał bez problemu, nagle przestał działać. Jednak po wymianie płyty głównej program dalej nie działał. Kluczem do rozwiązania zagadki okazało się to, że nowy komputer miał procesor wielordzeniowy. Po analizie dokumentacji procesora okazało się, że w architekturze wielordzeniowej większość rozkazów maszynowych, w tym rozkaz inc, nie jest już niepodzielnych (nie blokują magistrali). Efektem było „gubienie" niektórych operacji zwiększenia dokładnie tak, jak w omawianym przed chwilą przykładzie.

Powyższe dwa przykłady, oprócz trudności z analizą programów współbieżnych, pokazują jeszcze jedną ważną dla programisty rzecz. Nie wolno dopuścić do współbieżnego modyfikowania tej samej zmiennej przez wiele procesów. Zauważmy ponadto, że gdyby instrukcja zwiększania i zmniejszania zmiennych wykonywały się w sposób niepodzielny to problemu nie byłoby. Albo innymi słowy, w dowolnym momencie co najwyżej jeden proces może w tym samym czasie wykonywać pewien kluczowy fragment programu. Mówimy, że ten fragment stanowi **sekcję krytyczną**, a sam problem właściwej synchronizacji procesów nosi nazwę problemu **wzajemnego wykluczania**.

3 WZAJEMNE WYKLUCZANIE

Niektóre problemy synchronizacyjne pojawiają się tak często w praktyce, że omawia się je na każdym wykładzie na temat programowania pod nazwą **klasyczne problemy współbieżności**. Problem wzajemnego wykluczania pojawia się w codziennych sytuacjach. Na przykład w czasie towarzyskiej, kulturalnej dyskusji wielu osób, co najwyżej jedna z nich w danym momencie powinna zabierać głos. Jeśli do komputera jest podłączona jedna drukarka, to korzystać z niej może co najwyżej jeden proces. Fragment programu modyfikujący zmienną, z której korzysta wiele procesów może naraz wykonywać co najwyżej jeden proces (to ostatnie wymaganie można nieco osłabić, ale nie będziemy o tym mówić tutaj). Problem wzajemnego wykluczania formułuje się zazwyczaj na przykładzie dwóch procesów. Każdy z nich wykonuje cyklicznie fragment nazwany tutaj *własne sprawy*, a następnie fragment nazwany *sekcją krytyczną* zgodnie z poniższym schematem:

Powtarzaj:
    własne sprawy
    sekcja krytyczna

Własne sprawy jest fragmentem programu, który nie interesuje nas z punktu widzenia synchronizacji. Zakłada się, że proces może wykonywać *własne sprawy* dowolnie długo, może nawet zakończyć się tam w wyniku

błędu. **Sekcja krytyczna** to fragment wymagający synchronizacji. Zakłada się, że każdy proces, który rozpocznie jej wykonanie po ustalonym czasie ją skończy (a więc w szczególności nie zakończy się w niej z błędem). Trzeba wstawić odpowiedni fragment przed *sekcją krytyczną* i po *sekcji krytycznej* tak, aby zapewnić jej wzajemne wykluczanie.

Rozwiązanie wydaje się proste. Przedstawimy go używając pojęć znanych z życia codziennego. Powiedzmy, że przed *sekcją krytyczną* stoi sygnalizator (w programie jest to po prostu zmienna przyjmująca jedną z dwóch wartości) wyświetlający światło zielone, jeśli nikogo nie ma w *sekcji krytycznej* i czerwone w przeciwnym razie. Początkowo światło jest oczywiście zielone. Proces, który chce wejść do *sekcji krytycznej* najpierw sprawdza światło. Jeśli jest zielone, to przełącza go na czerwone i wchodzi do *sekcji krytycznej*. Jeśli jednak światło jest czerwone, to proces zatrzymuje się i czeka aż światło zmieni się na zielone. Proces wychodzący z *sekcji krytycznej* przełącza światło na zielone.

Rozwiązanie to jest jednak niepoprawne. Pierwszy proces, który będzie chciał wejść do *sekcji krytycznej* sprawdzi światło. Zobaczy zielone. Jeśli teraz zanim proces zdąży przełączyć światło, drugi proces zapragnie wejść do *sekcji krytycznej*, to sprawdzi sygnalizator, zobaczy jeszcze światło zielone i uzna, że droga wolna. Oba procesy przełączą światło na czerwone i spokojnie wykonają *sekcję krytyczną*. Mamy program, w którym wbrew wymaganiom w *sekcji krytycznej* przebywają dwa procesy.

4 POPRAWNOŚĆ PROGRAMÓW WSPÓŁBIEŻNYCH

4.1 WŁASNOŚĆ BEZPIECZEŃSTWA

Rozwiązania, które nie spełniają warunku wzajemnego wykluczania nazywamy **rozwiązaniami niebezpiecznymi**, a sam warunek przebywania w *sekcji krytycznej* co najwyżej jednego procesu w tym samym czasie – **warunkiem bezpieczeństwa**. W ogólności warunek bezpieczeństwa to warunek specyfikujący sposób synchronizacji procesów. Rozważmy inne przykłady warunków bezpieczeństwa z życia codziennego. W przypadku łyżwiarzy poruszających się po lodowisku własność bezpieczeństwa to po prostu brak zderzeń z innymi łyżwiarzami i z bandą. Skrzyżowanie dróg można też traktować jako pewien system współbieżny. Procesami są tu samochody (dla uproszczenia przyjmijmy, że przejeżdżają one skrzyżowanie na wprost i że obie drogi są jednokierunkowe), a własność bezpieczeństwa oznacza brak kolizji. Zauważmy, że przykłady te dobrze uzasadniają nazwę **własność bezpieczeństwa**, gdyż jej brak prowadzi z reguły do groźnej sytuacji. Podobnie jest w przypadku systemu komputerowego niespełniającego własności bezpieczeństwa.

4.2 WŁASNOŚĆ ŻYWOTNOŚCI

Powróćmy do przykładu wzajemnego wykluczania. Zmodyfikujmy nieznacznie poprzednie rozwiązanie – niech sygnalizator wyświetla początkowo światło czerwone. Okazuje się, że uzyskaliśmy rozwiązanie bezpieczne! Faktycznie, w *sekcji krytycznej* przebywa w dowolnej chwili najwyżej jeden proces. Tak naprawdę nikt nigdy do niej jednak nie wejdzie. Ewidentnie nie jest to rozwiązanie, które bylibyśmy skłonni zaakceptować, choć w myśl dotychczasowych rozważań jest ono poprawne.

Aby uniknąć tego typu rozwiązań musimy doprecyzować pojęcie poprawności programu współbieżnego. Już wiemy, że rozwiązanie musi mieć własność bezpieczeństwa. Do tego dokładamy jeszcze **własność żywotności**, która w przypadku wzajemnego wykluczania brzmi tak: „każdy proces, który chce wejść do *sekcji krytycznej*, w końcu do niej wejdzie". Zauważmy, że nie żądamy, aby każdy proces w końcu wszedł do *sekcji krytycznej*, ale ograniczamy to polecenie jedynie do tych procesów, które chcą tego, czyli zakończyły wykonanie *własnych spraw*. Proces może bowiem nigdy nie zakończyć *własnych spraw*, więc żądanie, by każdy proces w końcu dostał się do *sekcji krytycznej* jest zbyt silne.

W ogólnym przypadku żywotność oznacza, że każdy proces, który dotarł do fragmentu programu wymagającego synchronizacji, w końcu przez ten fragment przejdzie. Popatrzmy na inne przykłady. W przypadku skrzyżowania żywotność oznacza, że każdy proces w końcu przez nie przejedzie, w przypadku łyżwiarzy – że każdy z nich w końcu wykona jakiś krok lub obrót.

4.3 PRZYKŁADY BRAKU ŻYWOTNOŚCI

Zakleszczenie przy dostępie do zasobów

Czym może przejawiać się brak żywotności? W naszym rozwiązaniu doszło do **zakleszczenia**, czyli sytuacji, w której nic w systemie się nie dzieje. Żaden proces nie wykonuje pożytecznej pracy, wszystkie czekają na zdarzenie, które nigdy nie nastąpi. Jeszcze lepiej sytuację zakleszczenia ilustruje nieco inny przykład. Przypuśćmy, że w systemie działają dwa procesy. Każdy z nich w pewnym momencie potrzebuje do dalszego działania dostępu do drukarki i skanera. Pierwszy proces zgłasza najpierw zapotrzebowanie na skaner. Takie zgłoszenia obsługuje system operacyjny, który stwierdza, że skaner jest wolny, więc przydziela go pierwszemu procesowi. Następnie drugi proces zgłasza chęć skorzystania z drukarki. I znów system stwierdza, że drukarka jest wolna, więc przydziela ją procesowi. Teraz pierwszy proces prosi o drukarkę i musi czekać, bo drukarkę dostał drugi proces. Gdy drugi proces poprosi o skaner, to również będzie musiał czekać, bo skaner dostał pierwszy proces. Ale pierwszy proces nie zwolni skanera dopóki nie otrzyma drukarki, a drugi proces nie zwolni drukarki, póki nie otrzyma skanera. Mamy zakleszczenie!

Zakleszczenie w ruchu drogowym

Przykłady zakleszczeń nietrudno znaleźć w życiu codziennym. Przeanalizujmy na przykład artykuł 25 punkt 1 Kodeksu drogowego: „Kierujący pojazdem, zbliżając się do skrzyżowania, jest obowiązany zachować szczególną ostrożność i ustąpić pierwszeństwa pojazdowi nadjeżdżającemu z prawej strony (...)”. Zatem w sytuacji, gdy do skrzyżowania równorzędnego dojeżdżają jednocześnie pojazdy ze wszystkich czterech wlotów, mamy zakleszczenie. Żaden pojazd nie może ruszyć, dopóki nie odjedzie ten z prawej strony. Ze względu na symetrię tego układu nikt nigdy z takiego skrzyżowania nie powinien odjechać.

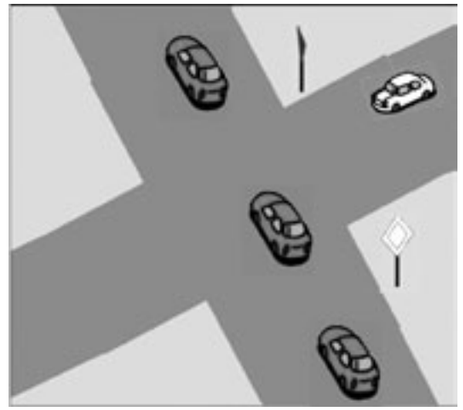
Jeszcze dotkliwiej sytuację tę widać na rondzie. Kodeks drogowy nie wyróżnia tu konieczności innego zachowania, więc na rondzie domyślnie pierwszeństwo mają pojazdy wjeżdżające. Bardzo szybko może doprowadzić to do całkowitego zakorkowania skrzyżowania i w konsekwencji zakleszczenia. Szczęśliwie, na większości tego typu skrzyżowań pod znakiem „skrzyżowanie o ruchu okrężnym” znajduje się również znak „ustąp pierwszeństwa przejazdu”, co powoduje, że pierwszeństwo mają pojazdy znajdujące się już na rondzie. Ale... czy to rozwiązuje problem potencjalnego zakleszczenia? Okazuje się, że nie!

Zagłodzenie w ruchu drogowym

Inny przejaw braku żywotności jest znacznie bardziej subtelny. O ile zakleszczenie było w pewnym sensie globalnym brakiem żywotności i powodowało przestój całego systemu, o tyle **zagłodzenie** polega na tym, że jest pewien „pechowy” proces (lub procesy), które będą w nieskończoność oczekiwać na możliwość wznowienia wykonania.

Rozważmy znów przykład skrzyżowania. Załóżmy, że droga prowadząca na wschód jest drogą z pierwszeństwem przejazdu. Zgodnie z artykułem 5 Kodeksu drogowego „Uczestnik ruchu i inna osoba znajdująca się na drodze są obowiązani stosować się do poleceń i sygnałów dawanych przez osoby kierujące ruchem lub uprawnione do jego kontroli, sygnałów świetlnych oraz znaków drogowych (...)”.

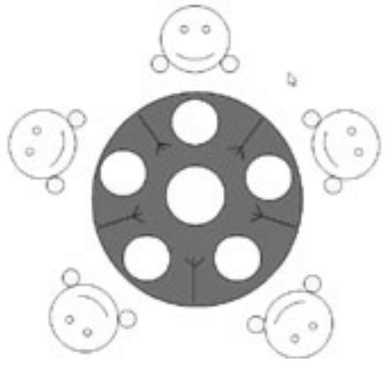
Zatem pojazd na rysunku 5, nadjeżdżający drogą z prawej strony, nie może wjechać na skrzyżowanie, jeśli zbliżają się do niego pojazdy jadące drogą nadrzędną. Jeśli droga ta jest drogą ruchliwą, to może się zdarzyć, że ciągle na skrzyżowanie przybywają nowe pojazdy i ten pojazd nigdy go nie przejedzie. Dochodzi do jego zagłodzenia.



Rysunek 5.
Możliwe zagłodzenie pojazdu nadjeżdżającego ulicą z prawej strony

Problem pięciu filozofów

Bardzo dobrze pojęcia zakleszczenia i zagłodzenia ilustruje inny klasyczny problem współbieżności, **problem pięciu filozofów**. Jest on przedstawiany w postaci następującej anegdoty (patrz rysunek 6). Przy okrągłym stole siedzi pięciu filozofów. Pośrodku stołu znajduje się (ciągle uzupełniany) półmisek z rybą. Przed każdym filozofem leży talerz, a między każdymi dwoma talerzami leży widelec. Tak więc na stole znajduje się pięć talerzy i pięć widelców. Każdy filozof najpierw myśli, a gdy zgłodnieje sięga po oba widelce znajdujące się obok jego talerza, nakłada sobie rybę, zjada ją, po czym odkłada widelce i wraca do myślenia itd.



Rysunek 6.
Pięciu myślących filozofów przy stole z rybą (na środku)

Widać, że każdy widelec jest użytkowany przez dwóch filozofów. Może się zatem zdarzyć, że głodny filozof nie będzie mógł rozpocząć jedzenia natychmiast, bo będzie musiał poczekać, aż skończy jeść jego sąsiad. Warunek bezpieczeństwa w przypadku tego problemu wyraża fakt, że filozof może rozpocząć jedzenie tylko wtedy, gdy ma oba widelce (znajdujące się przy jego talerzu) oraz że w dowolnej chwili każdym widelcem je co najwyżej jeden filozof. Żywotność oznacza, że każdy filozof, który jest głodny, w końcu będzie mógł rozpocząć jedzenie. Zakładamy, że filozof w skończonym czasie skończy jedzenie, natomiast myślenie może trwać dowolnie długo i w tym czasie może zdarzyć się wszystko (łącznie z awarią procesu).

Rozważmy teraz kilka różnych możliwych zachowań filozofów (czyli programów przez nich wykonywanych). Pierwszy sposób „działania” filozofa jest następujący. Gdy tylko zgłodnieje sięga po lewy widelec. Jeśli go nie ma na stole (bo używa go sąsiad), to filozof czeka. Następnie z lewym widelcem w garści filozof sięga po prawy. I znów, jeśli widelec jest zajęty, to filozof musi poczekać. Będąc w posiadaniu obu widelców filozof zjada rybę, po czym odkłada widelce na stół. Zastanówmy się, czy taki schemat postępowania filozofa jest poprawny. Własność bezpieczeństwa jest zachowana, jednak nie ma żywotności. Może bowiem zdarzyć się tak, że wszyscy filozofowie jednocześnie zgłodnieją i każdy z nich sięgnie po lewy widelec. Ponieważ widelce znajdują się na stole, więc każdy filozof podniesie lewy widelec. Ale teraz na stole nie ma już żadnego widelca i wszyscy filozofowie oczekują na prawy widelec. Ponieważ żaden z nich nie odda już podniesionego widelca, to mamy zakleszczenie.

Przeanalizujmy teraz nieco inny schemat działania filozofa. Załóżmy, że głodny filozof sprawdza, czy na stole są oba potrzebne mu widelce. Jeśli tak, to podnosi je jednocześnie i rozpoczyna jedzenie. Jeśli jednak nie ma choć jednego widelca, to czeka nie podnosząc żadnego widelca. Przyjmujemy przy tym, że sprawdzenie, czy widelce są na stole i ich podniesienie jest realizowane w sposób niepodzielny. To założenie gwarantuje spełnienie własności bezpieczeństwa. Czy to rozwiązanie jest żywotne? Okazuje się, że nie, choć tym razem nie dojdzie do zakleszczenia. Istnieje jednak scenariusz, w którym dwóch filozofów może „zmówić się” przeciwko trzeciemu siedzącemu między nimi. Aby prześledzić ten przeplot ponumerujmy filozofów zgodnie z ruchem wskazówek zegara od 1 do 5 począwszy od filozofa u góry stołu. Najpierw głodnieje filozof numer 1. W związku z ty, że oba widelce są dostępne, podnosi je i rozpoczyna jedzenie. Następnie głodnieje filozof numer 2 – jego właśnie spróbujemy zagłodzić (sic!). Ponieważ nie ma prawego widelca (używa go filozof 1), to musi poczekać. Następnie głodnieje filozof numer 3. Oba jego widelce są dostępne, więc może rozpocząć jedzenie.

Jeśli teraz filozof numer 1 zakończy jedzenie, to odłoży widelec. Niestety filozof numer 2 nie może rozpocząć jedzenia, bo nie ma widelca, którym aktualnie je filozof numer 3. Zanim filozof numer 3 odłoży swoje widelce, znów głodnieje filozof numer 1. I ponownie filozof numer 2 nie może jeść z tego samego powodu co na początku: braku prawego widelca. Gdy w dalszym ciągu filozofowie 1 i 3 będą jedli na zmianę, to nie pozwolą nigdy najeść się filozofowi numer 2. Mamy zatem scenariusz prowadzący do zagłodzenia filozofa numer 2.

Poprawne rozwiązanie jest nieco zmodyfikowanym wariantem pierwszym rozwiązania. Zauważmy, że do niepożądanej sytuacji dochodziło wtedy, gdy głodnieli wszyscy filozofowie. Wyobraźmy sobie teraz, że filozof, który myśli odsuwa się lekko od stołu. Gdy zgłodnieje, sprawdza najpierw, czy przy stole są już wszyscy pozostali. Jeśli tak, to czeka, a w przeciwnym razie przysuwa się do stołu i postępuje jak w wariancie pierwszym: próbuje podnieść najpierw lewy, potem prawy widelec, je, odkłada oba widelce. Następnie odsuwa się od stołu zwalniając przy nim miejsce i znów rozmyśla.

Rysunek 7.
Możliwy przeplot myślenia i jedzenia

PODSUMOWANIE

Z racji upowszechnienia się systemów wielozadaniowych, sieci, Internetu oraz wzrostu mocy obliczeniowej współczesnych komputerów programowanie współbieżne bardzo zyskało na znaczeniu. Programy współbieżne często mają lepszą strukturę niż programy sekwencyjne. Powstają w sposób bardziej modularny, pozwalając programiście skupić się na jednym aspekcie rozwiązywanego problemu. Ponadto niektóre algorytmy (na przykład sortowanie przez scalanie) w sposób naturalny zapisuje się w postaci programu współbieżnego.

Program współbieżny można wykonywać na komputerze wieloprocesorowym, a także na komputerze wyposażonym tylko w jeden procesor. Gdy faktycznie dochodzi do wykonywania kilku czynności w tym samym czasie mówimy o wykonaniu równoległym (możliwym na wielu procesorach lub procesorze wielordzeniowym). Inną techniką realizacji współbieżności jest wykonanie w przeplocie implementowane przez systemy operacyjne z podziałem czasu.

Oprócz zalet techniki programowania mają także wady. Programowanie współbieżne jest trudne. Programy współbieżne są trudne do analizy. Często ich działanie jest niezgodne z intuicją. Utrudnione jest również wykrywanie i usuwanie błędów. Błędny scenariusz może bowiem ujawniać się bardzo rzadko. To z kolei powoduje, że niełatwo jest go odtworzyć w celu znalezienia błędu podczas krokowego wykonania programu.

Program współbieżny może mieć wiele scenariuszy wykonań. Poprawność oznacza brak niepożądanych zachowań w żadnym z możliwych przeplotów. Program musi być bezpieczny, czyli spełniać wszystkie stawiane mu wymagania synchronizacyjne, oraz żywotny. Ta druga własność oznacza, że żaden proces nie oczekuje w nieskończoność na zajście zdarzenia, które pozwoli mu kontynuować działanie.

Aby ułatwić programistom tworzenie poprawnych programów współbieżnych, wymyślono wiele różnych mechanizmów synchronizacji i metod komunikacji procesów. Ich omówienie i analiza wykracza jednak poza zakres tych zajęć.

LITERATURA

1. Ben-Ari M., *Podstawy programowania współbieżnego i rozproszonego*, WNT, Warszawa 2009
2. http://osilek.mimuw.edu.pl/index.php?title=Programowanie_współbieżne_i_rozproszone/PWR_Wykład_1

Jak informatyka pomaga zajrzeć do wnętrza ludzkiego ciała

Ryszard Tadeusiewicz
Katedra Automatyki, Akademia Górniczo-Hutnicza im. S. Staszica
Kraków
rtad@agh.edu.pl

Streszczenie

Obok wielu zadań, jakie dziś spełniają komputery w różnych dziedzinach techniki i gospodarki, mogą one także rejestrować, prezentować i analizować sygnały z wnętrza ludzkiego ciała. Dawniej kontakt lekarza z organizmem pacjenta kończył się na powierzchni skóry. Można było chorego obserwować i badać, ale to, co się działo we wnętrzu jego ciała pozostawało tajemnicą. Dziś tomografia komputerowa, metody ultradźwiękowe, obrazowanie magnetyczne, techniki izotopowe, termowizja i wiele innych technik medycznych pozwalają zwiedzać wnętrze ciała człowieka tak, jak się ogląda wnętrze budynku. Możemy wejść, gdzie chcemy, widzieć to, co chcemy i pomagać ludziom przezwyciężać choroby tak skutecznie, jak nigdy dotąd. A wszystko dzięki temu, że nauczyliśmy się zamieniać różne sygnały na liczby, a liczby na obrazy.

Na wykładzie przedstawione będą różne urządzenia służące do pozyskiwania obrazów medycznych oraz pokazane będą efekty komputerowej obróbki tych obrazów. Omówiona będzie także możliwość automatycznego rozpoznawania obrazów przez komputer wraz z dyskusją, jaką rolę taki system automatycznego rozpoznawania powinien pełnić: Czy ma zastępować lekarza, czy być tylko jego inteligentnym doradcą?

Spis treści

1 CYFROWA REPREZENTACJA OBRAZÓW

Komputery są dziś powszechnie stosowane do przechowywania, przetwarzania i przesyłania także obrazów. Napisano „także”, bo obraz nie jest naturalnym obiektem, którym komputer może się posługiwać ze względu na swoją budowę i pierwotne przeznaczenie. Komputery miały operować tylko liczbami i do tego przystosowany jest ich element przetwarzający informacje (mikroprocesor), pamięć oraz urządzenia komunikacyjne. Te stwierdzenia wydają się oczywiste, ale często uczniowie nie zastanawiają się nad tym i warto im to od razu na początku uświadomić.

To, że dzisiaj komputery służą również do pisania tekstów albo odtwarzania muzyki wynika z faktu, że teksty i sygnały (na przykład dźwięki) mogą być zamienione na serie liczb. Komputer operuje liczbami, a my widzimy litery na ekranie albo słyszymy ulubioną melodię. W taki sam sposób „oswojono” komputery z obrazami, które też są zamieniane na liczby, a zbiorowość tych liczb po odpowiednim przedstawieniu może być podziwiana jako rysunek albo cyfrowe zdjęcie.

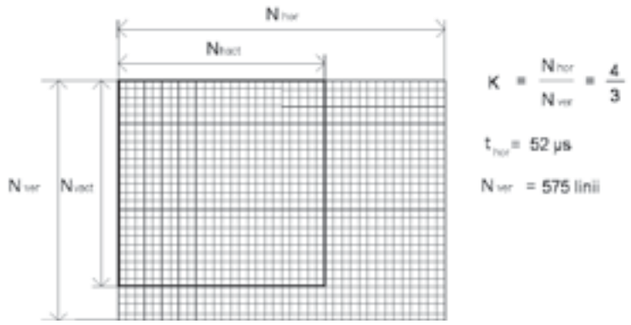
W systemie komputerowym obraz jest zawsze reprezentowany w postaci **próbkowanej** (to znaczy jasność albo barwa są podawane tylko w niektórych punktach) oraz **skwantowanej** (czyli jego jasność oraz barwa może przyjmować wyłącznie niektóre, z góry zadane wartości – rys. 1).

Rysunek 1.
Sposób tworzenia obrazu cyfrowego poprzez dyskretyzację i kwantyzację

Niewątpliwie technika obrazów cyfrowych jest bardzo zaawansowana i ma bardzo duże znaczenie, niemniej trzeba pamiętać, że obraz cyfrowy jest zawsze zubożony w stosunku do obrazu analogowego, z którego powstał, chociaż to zubożenie może uczynić dowolnie małym. Porównanie obrazu analogowego i cyfrowego (celowo bardzo niedoskonałego) można obejrzeć na rysunku 2. Na tej ilustracji widać (w sposób karykaturalnie wyolbrzymiony) różnicę między obrazem analogowym i obrazem cyfrowym.

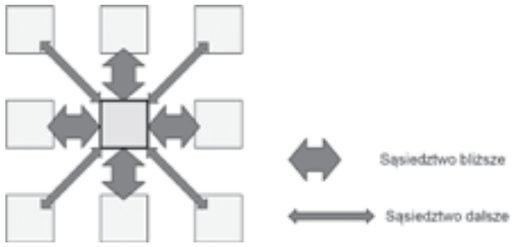
Rysunek 2.
Obraz analogowy i cyfrowy [źródło: 4]

Poznajmy teraz kilka szczegółów na temat cyfrowej reprezentacji obrazów. Przy każdej cyfrowej reprezentacji obrazu jego powierzchnia dzielona jest na rozłączne obszary, zwane **pikselami**. Najczęściej stosowany jest układ pikseli prostokątny lub kwadratowy (rys. 3).

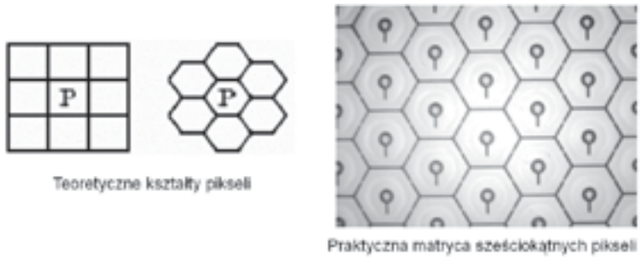


Rysunek 3.
Prostokątny lub kwadratowy układ pikseli [źródło: 4]

Przy pikselach kwadratowych lub prostokątnych jest jednak pewien problem. Otóż podczas operacji wykonywanych przez komputer na obrazie trzeba uwzględniać, że dystans od danego piksela do pikseli sąsiednich zależy od tego, czy jest mierzony w pionie, w poziomie czy po przekątnych. Na obrazie występuje więc dwojakiego rodzaju sąsiedztwo: bliższe i dalsze (rys. 4). Sprawia to kłopot przy wielu obliczeniach, dlatego w użyciu jest także niekiedy podział obrazu (raster), przy którym każdy piksel jest heksagonalny (sześciokątny) – rysunek 5. Raster z sześciokątnymi pikselami gwarantuje, że wszystkie piksele sąsiednie są tak samo odległe od piksela centralnego. Mimo licznych zalet nie jest on jednak szczególnie popularny.

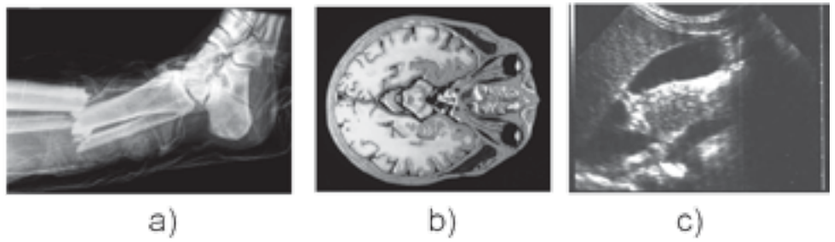


Rysunek 4.
Dwojakiego rodzaju sąsiedztwo: bliższe i dalsze przy pikselach kwadratowych



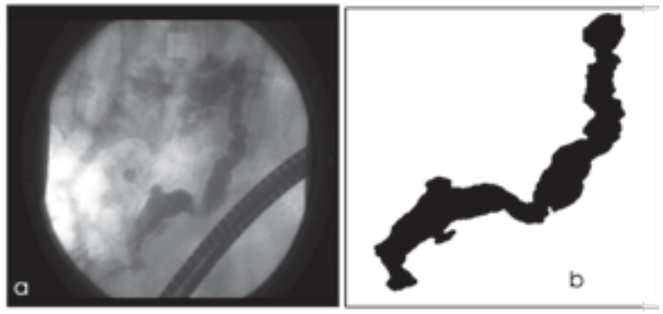
Rysunek 5.
Wykorzystanie sześciokątnych pikseli [źródło: 4]

Niniejsze opracowanie dotyczy obrazów medycznych. Istnieje ich wiele rodzajów, ale dla potrzeb tego opracowania wyróżniamy trzy typy cyfrowych obrazów medycznych (rys. 6): a) rentgenogram (z tradycyjnego aparatu rentgenowskiego wykorzystującego promienie X przenikające przez ciało człowieka), b) tomogram (z tomografu komputerowego wykorzystującego zjawisko rezonansu magnetycznego), c) ultrasonogram (z ultrasonografu sondującego wnętrze ciała człowieka wiązkami ultradźwięków). Każdy z nich powstaje w wyniku wykorzystania innego zjawiska fizycznego, inaczej wygląda i wymaga nieco odmiennych sposobów reprezentacji w pamięci komputera oraz innych metod przetwarzania i analizy.



Rysunek 6.
Trzy typy cyfrowych obrazów medycznych

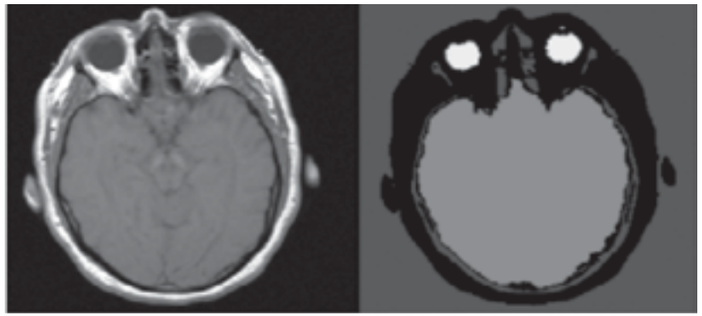
Obrazy (zwłaszcza medyczne) zawierają zwykle bardzo wiele szczegółów, co utrudnia ich interpretację (zwłaszcza gdy ta interpretacja ma być przeprowadzana automatycznie przez komputer). Dlatego obok obrazów prezentujących narządy wnętrza ciała człowieka w pełnej skali stopni szarości (lub rzadziej barw) stosuje się tak zwane obrazy binarne. Na **obrazach binarnych** do zapisu wartości wszystkich pikseli wystarczą wyłącznie symbole 0 i 1. Obrazy binarne są wykorzystywane przy zaawansowanych analizach, gdy obraz został już wstępnie przetworzony do takiej postaci, że można na nim wyróżnić punkty należące do rozważanego obiektu (tym przypisuje się wartość 1) oraz punkty traktowane jako nieistotne elementy (tym przypisuje się wartość 0). Wygląd przykładowego obrazu binarnego przedstawiono na rysunku 7.



Rysunek 7.
Wygląd obrazu binarnego (b) oraz obrazu oryginalnego, z którego ten obraz binarny wydobyto (a). Obraz przedstawia wypełniony kontrastem przewód trzustkowy uwidoczniony w badaniu rentgenowskim nazywanym ERCP

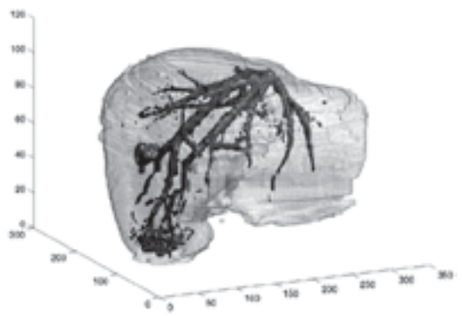
Przy **cyfrowej reprezentacji obrazu szarego** najczęściej przeznacza się jeden bajt na jeden piksel (rys. 6). W istocie jest to kodowanie trochę rozrzutne, gdyż cały bajt pozwala rozróżnić 256 poziomów szarości, podczas gdy ludzki wzrok potrafi odróżnić maksymalnie około 60 poziomów szarości. Jednak ze względu na organizację pamięci komputera nie opłaca się „upychać” kilku pikseli w jednym bajcie. Natomiast jeśli na obrazie

wyodrębni się (metodą automatycznej komputerowej segmentacji) kilka rodzajów interesujących nas obiektów – to można każdemu z tych obiektów przypisać inną wartość, którą można potem na obrazie prezentować jako inną barwę. Są to barwy umowne, niemające nic wspólnego z rzeczywistymi kolorami odpowiednich narządów wewnątrz ciała człowieka, ale są one przydatne do tego, żeby uwidocznić i uwypuklić budowę anatomiczną badanej części ciała. Taki sposób prezentacji obrazu medycznego pokazano na rysunku 8.



Rysunek 8.
Wygląd obrazu szarego (po lewej) oraz obrazu ze sztucznie wprowadzonymi barwami dla poszczególnych wykrytych struktur anatomicznych

Obrazy pozyskiwane z pomocą technik obrazowania medycznego są z reguły płaskie. Jednak stosując metody grafiki komputerowej można z szeregu takich płaskich obrazów zrekonstruować widok narządów wewnętrznych tak prezentowanych, jakby badający widział je jako obiekty trójwymiarowe. Na rysunku 9 można zobaczyć taką trójwymiarową rekonstrukcję narządu uzyskaną za pomocą komputera na podstawie serii płaskich obrazów z tomografu komputerowego.



Rysunek 9.
Wygląd trójwymiarowej rekonstrukcji narządu (wątroby) [źródło: http://www.vislab.uq.edu.au/research/liver/images/3D_liver_model.jpg]

Zaletą cyfrowej reprezentacji obrazów jest to, że obrazy przedstawione cyfrowo można niezwykle sprawnie przetwarzać (rys. 10). Pokazane na rysunku 10 przekształcenia obrazu znanego pod nazwą Lena (najczęściej używanego we wszystkich książkach i artykułach naukowych do przedstawienia efektów działania różnych algorytmów komputerowej obróbki obrazu) nie mają żadnego zastosowania i są czystą ciekawostką. Jednak w przypadku wielu obrazów rzeczywistych (w tym między innymi medycznych) modyfikacje takie mogą mieć istotne znaczenie praktyczne.

Rysunek 10.
Cyfrowe obrazy można dowolnie przetwarzać za pomocą komputerów [źródło: 4]

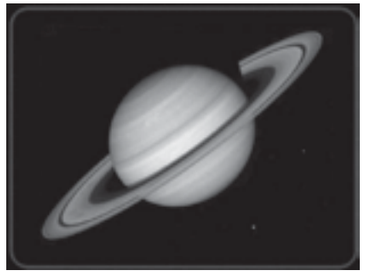
2 KOMPUTEROWE METODY OBRAZOWANIA MEDYCZNEGO

Przejdziemy teraz do zagadnień **wykorzystania** komputerowych zdolności manipulowania obrazami.

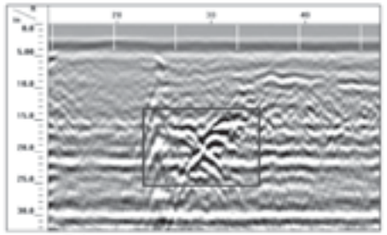
Wcześniej komputery wykorzystywały swoją zdolność zamieniania zbiorów liczb na obrazy tylko w obszarach grafiki komputerowej (rys. 11). W tym przypadku ich rola sprowadzała się wyłącznie do rozrywki. Komputerowe przetwarzanie obrazów stosowane jest też do cyfrowej obróbki zdjęć i filmów (nagrań wideo) i jego rola głównie wiąże się z dostarczaniem wiadomości (obrazy odległych miejsc i zdarzeń – rys. 12).

Rysunek 11.
Cyfrowa reprezentacja obrazu jest podstawą grafiki komputerowej, ale te zastosowania służą głównie rozrywce [źródło: http://extra-bajki.pl/wp-content/uploads/2010/07/760_the_shrek.jpg – dostęp lipiec 2011]

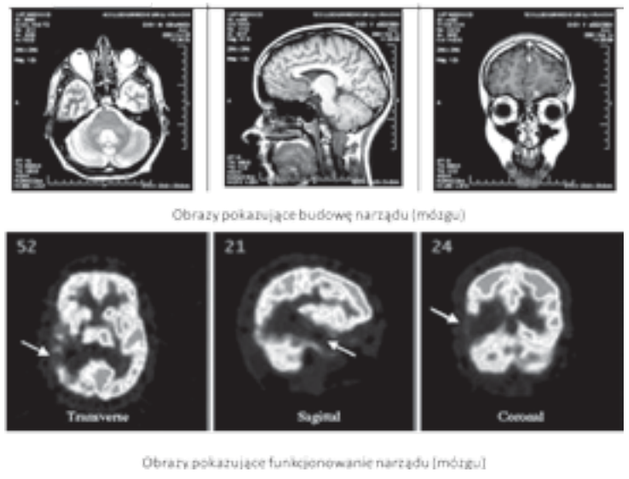
Komputery potrafią jednak coś więcej. Mogą przedstawiać jako obrazy liczby, które powstały z rejestracji i odpowiedniego przetwarzania różnych sygnałów (rys. 13). Mogą to być niedostrzegalne dla naszych zmysłów sygnały z kosmosu i dzięki temu poznajemy wszechświat. Mogą to być sygnały z wnętrza Ziemi i w ten sposób odkrywamy prawa fizyki – znajdujemy pod ziemią złoża surowców mineralnych, zabytki archeologiczne albo prawa geofizyki – na przykład mechanizmy wędrówki kontynentów. Ale najciekawsze jest to, że mogą to być sygnały z wnętrza ludzkiego ciała i w ten sposób widzimy, jak są zbudowane i jak działają nasze narządy (rys. 14).



Rysunek 12.
Przetwarzaniu komputerowemu podlegają też obrazy ilustrujące różne wiadomości. Widoczny obraz planety Saturn z sondy kosmicznej po przesłaniu na Ziemię był bardzo zniekształcony, ale został oczyszczony metodą komputerową [źródło: http://jaas.blox.pl/resource/saturn.jpg]



Rysunek 13.
Obrabiane komputerowo obrazy będące przedmiotem badań naukowych. Pokazano obraz z tak zwanego georadaru ujawniający różne obiekty ukryte pod ziemią, który wymaga obróbki komputerowej [źródło: http://geounda.com/wp-content/uploads/2010/12/forenzika22.jpg]

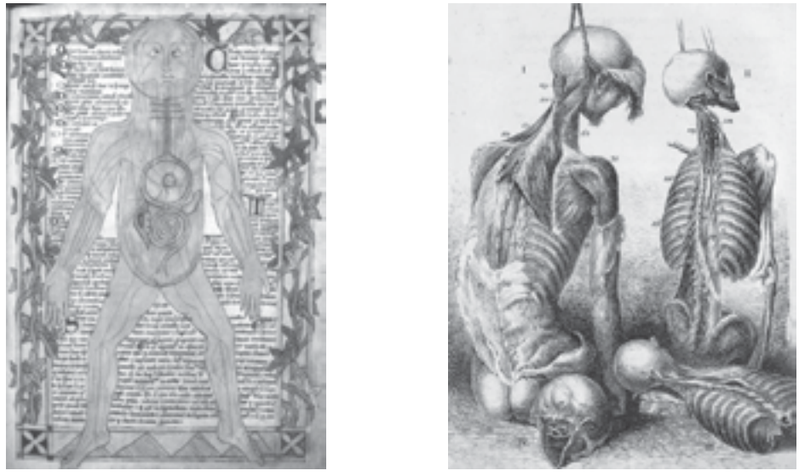


Rysunek 14.
Sygnały z wnętrza ludzkiego ciała przedstawione jako obrazy

Wyjaśnijmy, dlaczego wspomagane komputerowo metody obrazowania medycznego są takie ważne. Dawniej kontakt lekarza z organizmem pacjenta kończył się na powierzchni skóry. Można było chorego obserwować i badać, ale to, co się działo we wnętrzu jego ciała, pozostawało tajemnicą, chociaż zdolny i dobrze wykształcony lekarz potrafił odtwarzać obrazy narządów wewnętrznych oraz ich patologicznych zmian, korzystając ze swojej wiedzy i wyobraźni (rys. 15).

Rysunek 15.
Bezpośrednie oglądanie wnętrza ciała pacjenta było przez całe stulecia niemożliwe. Lekarz na podstawie badania wyobrażał sobie jedynie chory narząd

Bezpośrednie oglądanie wnętrza ciała pacjenta było jednak przez całe stulecia niemożliwe. Ogólna wiedza na temat budowy wewnętrznych narządów człowieka (anatomia) powstała wprawdzie wcześniej, gdyż żądni wiedzy naukowcy i lekarze prowadzili tysiące sekcji zwłok dla zbadania, jak wygląda i jak funkcjonuje ten niewiarygodnie wspaniały mechanizm, jakim jest ludzkie ciało. Na rysunku 16 przedstawiono zaczerpnięte z Wikipedii obrazy anatomiczne pochodzące odpowiednio z XIII i z początku XIX wieku.



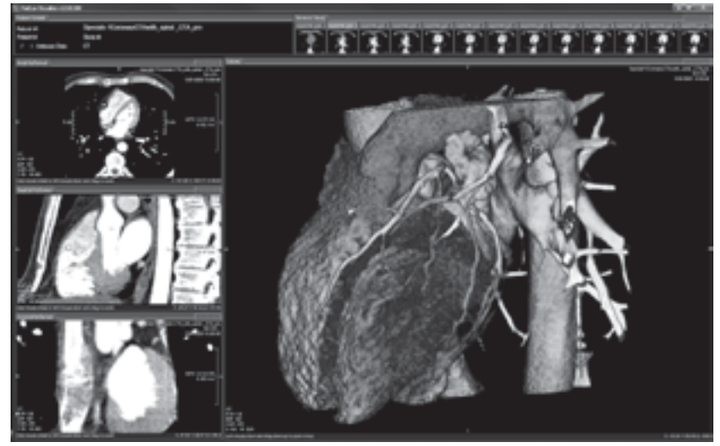
Rysunek 16.
Obrazowanie wnętrza ciała człowieka przez dawnych anatomów [źródło: http://pl.wikipedia.org/wiki/Grafika:13th_century_anatomical_illustration.jpg, http://pl.wikipedia.org/wiki/Grafika:John-bell-II-B-6.jpg]

Jak widać lekarze usiłowali poznać (zobaczyć!) wnętrze ciała człowieka, ale tylko w wyjątkowych przypadkach było to możliwe w odniesieniu do działających narządów żywych ludzi. Na rysunku 17 pokazano taką właśnie wyjątkową sytuację.

Rysunek 17.
Przypadek kobiety z otworem w klatce piersiowej umożliwiającym między innymi oglądanie pracującego serca [źródło: http://impaedcard.com/issue/issue27/aquilinao2/AquilinaO.htm]

Porównanie tego, o czym marzyli dawni anatomowie, z tym, co jest dzisiaj osiągalne za sprawą komputerów, ilustrować może zestawienie znanego obrazu „Lekcja anatomii profesora Tulpa", który namalował Rembrandt Harmenszoon van Rijn (1606-1669) (rys. 18) ze współczesną wizualizacją wnętrza ciała pacjenta (rys. 19).

Rysunek 18.
Poznawanie wnętrza ciała człowieka w średniowieczu [źródło: http://bi.gazeta.pl/im/7/4610/z4610417X.jpg]

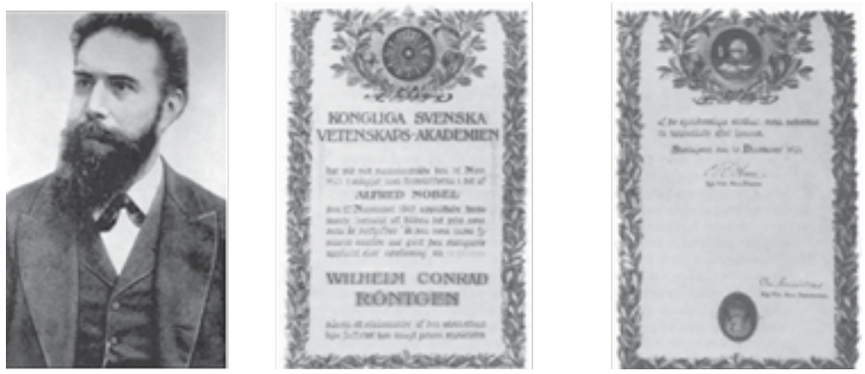


Rysunek 19.
Poznawanie wnętrza ciała człowieka współcześnie za pomocą tomografii komputerowej [źródło: http://rsna2008.rsna.org/exbdata/1947/images/heart01.jpg]

Pokazany na rysunku 19 obraz (będący cyfrową rekonstrukcją) powstał w następstwie komputerowej obróbki serii obrazów pochodzących z tomografii komputerowej. Zanim jednak przejdziemy do tak wyrafinowanych obrazów, przedstawimy i przedyskutujemy kilka uwag o roli komputerów we współczesnym obrazowaniu medycznym.

3 KOMPUTEROWA OBRÓBKA OBRAZÓW RENTGENOWSKICH

Osoby, które niewiele słyszały o metodach pozyskiwania obrazów narządów wewnątrz ciała człowieka, mogą twierdzić, że rola informatyki nie jest tu taka ważna, bo najważniejsze odkrycia zrobili fizycy. Jest to jednak prawda wysoce niepełna. Istotnie, jest prawdą, że jako pierwszy wnętrze ciała żywego człowieka zobaczył Wilhelm C. Röntgen, fizyk, odkrywca promieni X (rys. 20). Było to wielkie, wręcz przełomowe odkrycie, za które Röntgen otrzymał nagrodę Nobla (w 1901 roku).



Rysunek 20.
Wilhelm C. Röntgen i kopia jego dyplomu Nagrody Nobla z 1901 roku [źródło: http://www.roentgen-museum.de/]

Jednak samo zastosowanie metody fizycznej, odkrytej przez Röntgena, dostarcza obrazów o bardzo złej jakości, często trudnych do interpretacji i z tego powodu mniej przydatnych w medycynie. Dopiero zastosowanie obecnie komputerowej obróbki obrazu spowodowało, że obraz z aparatu rentgenowskiego stał się prawie tak samo czytelny, jak ilustracja w atlasie anatomicznym. Popatrzmy na rysunek 21. Przedstawiono na nim dwukrotnie ten sam obiekt anatomiczny – rękę kobiety z obrączką. Zdjęcie po lewej przedstawia obraz uzyskany z pomocą samego tylko obrazowania fizycznego (w istocie jest to historyczne pierwsze zdjęcie w promieniach X ręki żony Röntgena, zamieszczone w artykule naukowym ilustrującym istotę jego metody), zdjęcie po prawej przedstawia taki sam obiekt zobrazowany za pomocą aparatu, w którym obraz uzyskany z pomocą promieni X jest dodatkowo polepszany komputerowo.

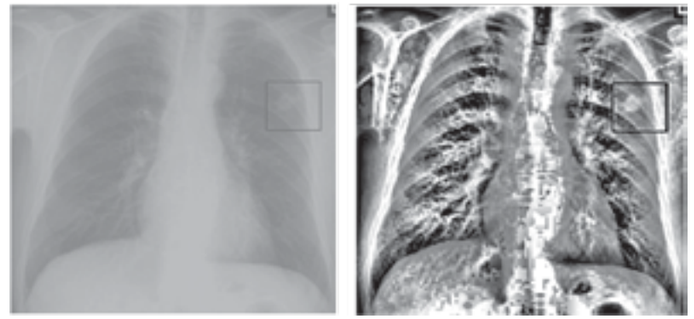
Rysunek 21.
Historyczne zdjęcie ręki żony Röntgena [po lewej – źródło http://upload.wikimedia.org/wikipedia/en/6/6e/Anna_Berthe_Roentgen.gif] oraz współczesna fotografia rentgenowska dłoni polepszona za pomocą komputera [po prawej – źródło http://science.hq.nasa.gov/kids/imagers/ems/hand.gif]

Inny przykład pokazano na rysunku 22. Na nieprzetworzonym komputerowo obrazie (po lewej stronie) nie widać szeregu szczegółów badanego obiektu, które ujawniają się dopiero po komputerowej obróbce rentgenogramu. Po prawej stronie rysunku na zdjęciu komputerowo przetworzonym widać wyraźnie raka w płucach pacjenta (w obszarze zaznaczonym kwadratem), podczas gdy ta sama zmiana nowotworowa na oryginalnym zdjęciu rentgenowskim (po lewej stronie) była prawie niewidoczna i mogłaby być przez lekarza niezauważona. Rola techniki komputerowej we wczesnej diagnostyce raka jest tu ewidentna.

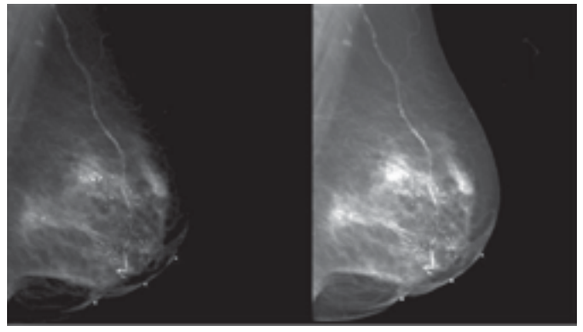
Jeszcze jeden przykład z tej samej serii przedstawia rysunek 23. Jest tam pokazany obraz mammograficzny w postaci oryginalnej oraz komputerowo poprawionej. Pokazano po lewej stronie obraz oryginalny, uzyskany przy użyciu samej tylko techniki rentgenowskiej, a po prawej ten sam obraz po zastosowaniu obróbki komputerowej. Widać, że komputer pomaga lepiej widzieć obrazowane szczegóły anatomiczne.

Badanie mammograficzne prowadzone jest w celu wczesnego wykrycia zmian nowotworowych w piersi kobiety, więc dokładna ocena struktur widocznych wewnątrz piersi jest niesłychanie ważne. Jak bardzo ważna jest jakość zobrazowania medycznego pokazuje przykład (zaczerpnięty z prezentacji udostępnionej przez prof. Artura Przelaskowskiego z Politechniki Warszawskiej) na rysunku 24. Na tym rysunku przedstawiono po lewej stronie obraz mammograficzny uzyskany podczas badania pewnej kobiety w 1996 roku. Obraz był doskonalony komputerowo,

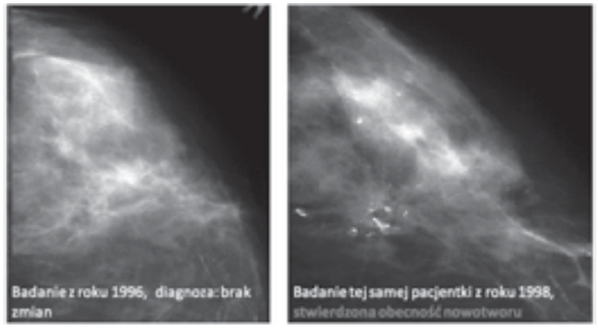
ale technika ta w latach 90. była jeszcze bardzo niedoskonała. Diagnoza (postawiona przez doświadczonego lekarza!) brzmiała: wszystko w porządku. Niestety był to błąd. Kolejne badanie wykonane w 1998 roku, którego wynik widoczny jest po prawej stronie, nie pozostawiało żadnych wątpliwości: w piersi był groźny rak. Niestety wykryto go zbyt późno i mimo natychmiast podjętego leczenia kobieta zmarła.



Rysunek 22.
Wpływ komputerowej obróbki obrazu rentgenowskiego na możliwość analizy szczegółów. Po lewej obraz w takiej formie, w jakiej go zarejestrował aparat rentgenowski, po prawej ten sam obraz po obróbce komputerowej

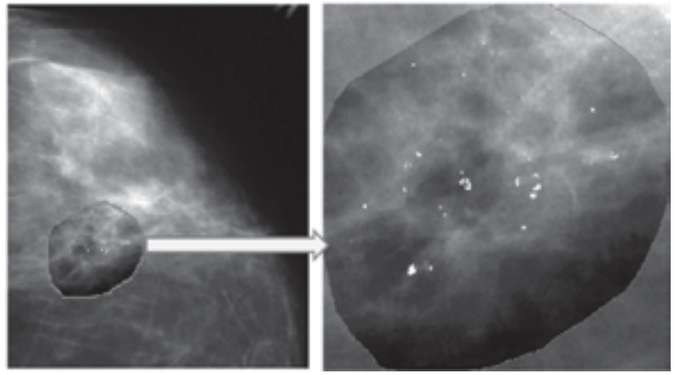


Rysunek 23.
Polepszanie jakości obrazu rentgenowskiego (mammograficznego) przy użyciu technik komputerowych [źródło: http://www.eradimaging.com/cffm/custom/V1_2/Figure-3.jpg]



Rysunek 24.
Przykład pomyłki diagnostycznej spowodowanej niską jakością obrazu

Gdyby obraz z 1996 roku analizowały komputery tak oprogramowane, jak to jest dzisiaj stosowane, na obrazie pojawiłoby się ostrzeżenie (rys. 25), że komputer widzi tu nieprawidłowości. Powiększenie rejonu z zaznaczeniem podejrzanych zmian (po prawej stronie rysunku) spowodowałoby wykrycie raka na etapie, kiedy jego usunięcie było łatwe i bezpieczne. Kobieta pewnie by żyła do dziś...



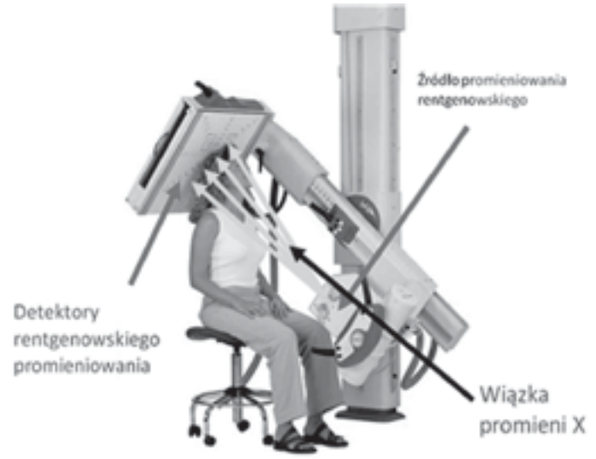
Rysunek 25.
Nowoczesny system komputerowy nie tylko polepsza jakość obrazu medycznego, ale dodatkowo także ostrzega lekarza o automatycznie wykrywanych zmianach patologicznych

4 AUTOMATYCZNE ROZPOZNAWANIE OBRAZÓW MEDYCZNYCH

W kontekście ostatniego omawianego tu przykładu interesująca jest możliwość automatycznego rozpoznawania obrazów medycznych przez komputer. Jest to przedmiot prowadzonych od ponad 20 lat badań naukowych autora wykładu. Otwarty jest także problem, jaką rolę taki system automatycznego rozpoznawania powinien pełnić: czy ma zastępować lekarza, czy być tylko jego inteligentnym doradcą? O tym wszystkim można poczytać między innymi w popularnej, przystępnej dla uczniów szkół średnich książce [3]. Rysunek 26 przedstawia etapy pozyskiwania, przetwarzania, analizy i diagnostycznego wykorzystania obrazów medycznych.

Rysunek 26.
Etapy pozyskiwania, przetwarzania, analizy i diagnostycznego wykorzystania obrazów medycznych. Strzałka wskazuje obszar, którym się obecnie zajmujemy

Angażowanie komputerów także do automatycznej **interpretacji** obrazów medycznych, a nie tylko ich ulepszonej prezentacji (w stosunku do oryginalnego obrazu uzyskanego za pomocą przenikających ciało pacjenta promieni X) jest uzasadnione jeszcze jedną okolicznością. Otóż komputer może widzieć na obrazie medycznym znacznie więcej niż człowiek!

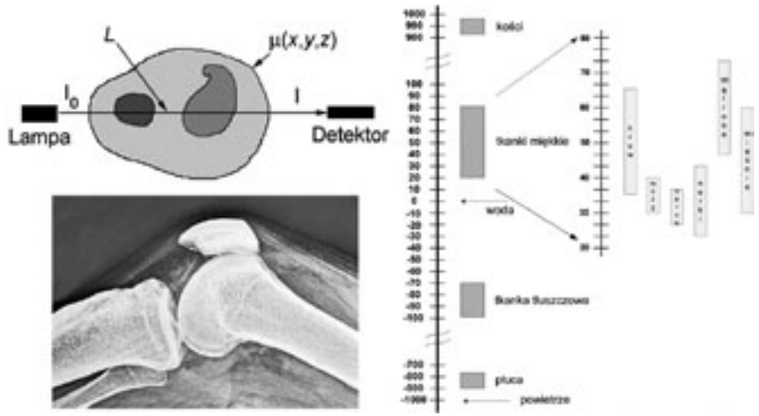


Rysunek 27.
Budowa aparatury rentgenowskiej [źródło: 1]

Żeby to wyjaśnić, przypomnijmy, jak działa aparat rentgenowski (rys. 27). Promieniowanie rentgenowskie wytwarzane jest w specjalnej lampie. Budowy lampy nie będziemy tu omawiać, natomiast warto poczytać o niej np. w Internecie, bo jest ciekawym osiągnięciem sztuki inżynierskiej, które naukowe odkrycie Röntgena zamieniła na technologię użyteczną w codziennej praktyce medycznej. Promieniowanie lampy przechodzi przez ciało pacjenta i jest po drugiej stronie mierzone przez detektory (rys. 28). Detektorów jest dużo, bo każdy z nich tworzy jeden piksel cyfrowego obrazu rentgenowskiego. Detektory te zastępują kliszę rentgenowską, z którą można się jeszcze spotkać w starszych aparatach rentgenowskich. Zdjęć rentgenowskich robionych na kliszach nie można komputerowo doskonalić, a ponadto trudno je gromadzić, wyszukiwać, przesyłać, zdalnie konsultować itd., więc wychodzą one z użycia.

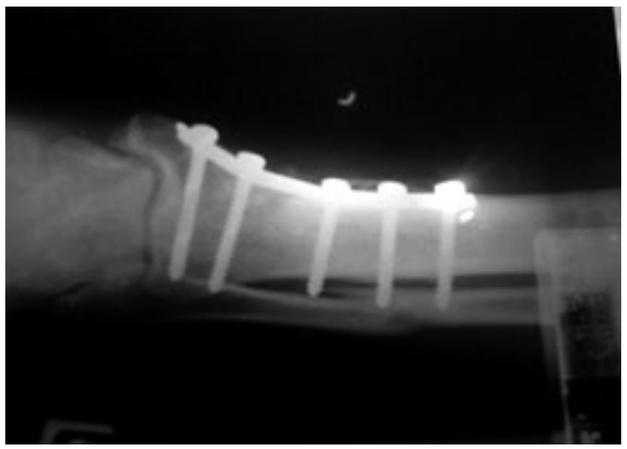
Rysunek 28.
Za pomocą takich matryc detektorów CCD rejestruje się obecnie zdjęcia rentgenowskie [źródło: http://suncityhospital.com/images/naomi-tech.png]

Promieniowanie lampy przechodzi przez ciało pacjenta i jest po drugiej stronie mierzone przez detektory. Obserwowane zmniejszenie natężenia promieniowania zależy od stopnia jego pochłaniania w tkankach. Dla uzyskania obrazu rentgenowskiego istotne jest, że różne tkanki pochłaniają promieniowanie w różnym stopniu. Przedstawiając w formie obrazu natężenie promieniowania, które przeszło przez ciało pacjenta w różnych miejscach, otrzymujemy **cienie** narządów wewnętrznych, szczególnie widoczne wtedy, gdy narządy te bardzo silnie pochłaniają promieniowanie (kości).



Rysunek 29.
Na zdjęciu rentgenowskim widać cienie narządów, które pochłaniają promienie X [źródło: 5]

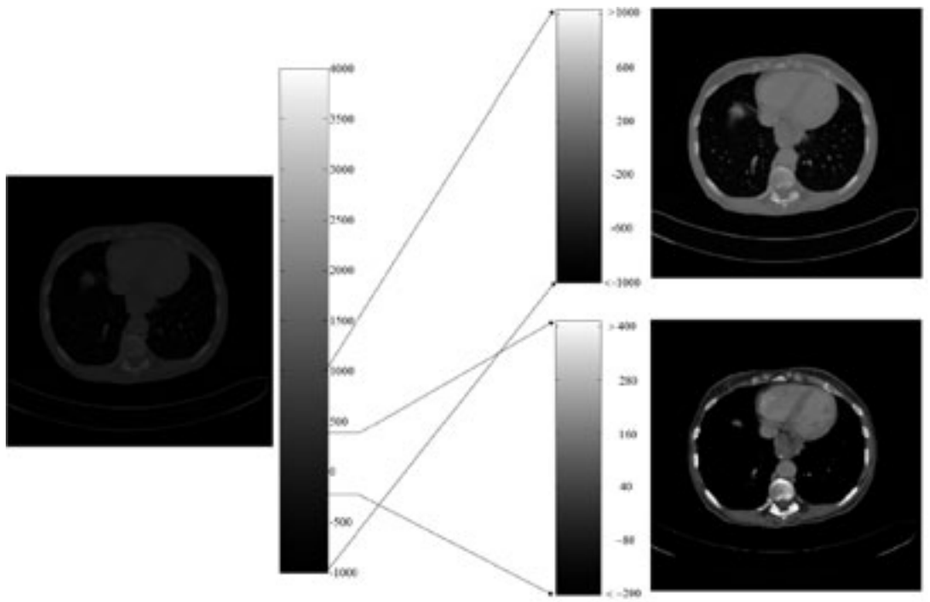
Stopień pochłaniania promieniowania rentgenowskiego określa tzw. **skala Hounsfielda** pokazana na rysunku 29. Warto zauważyć, że skala ta rozciąga się od wartości –1000 jednostek Hounsfielda (dla powietrza) do +1000 jednostek Hounsfielda (dla kości). Na zdjęciach rentgenowskich skala jest jeszcze szersza, bo bywają na nim ujawniane także obiekty metalowe, których zdolność pochłaniania sięga nawet 4000 jednostek Hounsfielda (rys. 30).



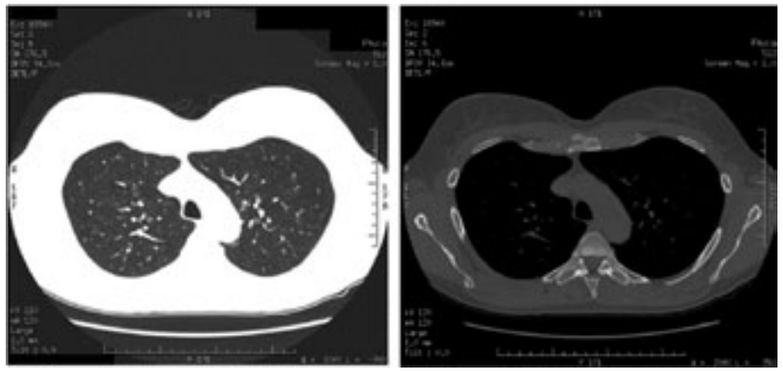
Rysunek 30.
Zdjęcie rentgenowskie uwidaczniające śruby zespalające kości po skomplikowanym złamaniu [źródło: http://images46.fotosik.pl/72/419f7b19982d40d0m.jpg]

Zakładając, że detektory mierzą promieniowanie w sposób zapewniający określenie nawet pojedynczych jednostek Hounsfielda (a są aparaty zdolne do mierzenia ułamkowych wartości w tej skali!), na zdjęciu rentgenowskim możliwe jest wyróżnienie 5000 różnych poziomów jasności. Tymczasem człowiek rozróżnia wzrokiem najwyżej 60 poziomów szarości! W związku z tym dobrze zrobione zdjęcie rentgenowskie zawiera o wiele więcej informacji, a człowiekowi (lekarzowi) pokazuje się obrazy w tzw. oknach – wybierając pewien fragment skali Hounsfielda, który (po odpowiednim skalowaniu) jest odwzorowywany w postaci rozróżnialnych ludzkim okiem szarości (rys. 31). Cała reszta informacji zawartej w jednostkach Hounsfielda poniżej i powyżej okna – jest przy tym tracona. Taki sposób postępowania sprawia, że możliwa jest całkiem odmienna wizualizacja tego samego obrazu rentgenowskiego w zależności od dobranych parametrów okna (rys. 32). W efekcie lekarz zawsze widzi tylko jeden, wybrany aspekt badanego zjawiska, podczas gdy komputer może widzieć wszystko, bez żadnych ograniczeń. Stąd automatyczna (komputerowa) interpretacja obrazów medycznych może być głębsza i dokładniejsza, niż prowadzona przez ludzi.

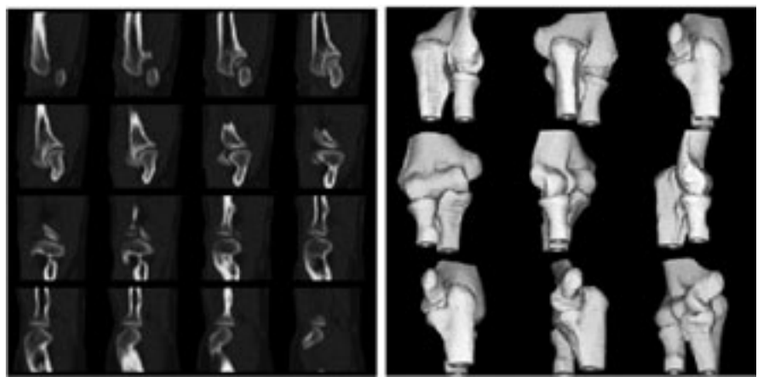
Jednak nawet pozostawiając decyzje w rękach lekarzy stwierdzić trzeba, że komputery mogą się posunąć znacznie dalej w doskonaleniu obrazów medycznych, niż to było pokazane na rysunkach 21–25. Najnowsze systemy tego typu łączą w sobie elementy analizy obrazu i grafiki komputerowej (rys. 33). Seria zdjęć rentgenowskich (pokazana po lewej stronie rysunku), ujawniająca budowę badanego narządu (w pokazanym przypadku jest to łokieć), dostarcza informacji o rozmiarach, proporcjach i wzajemnym ułożeniu elementów anatomicznych u konkretnego pacjenta. Sam narząd jest potem modelowany komputerowo, a efekt tego modelowania jest wizualizowany za pomocą metod grafiki komputerowej, co umożliwia oglądanie obrazu czytelniejszego niż źródłowe zdjęcia rentgenowskie, a ponadto możliwy do oglądania w dowolnym położeniu i pod dowolnym kątem.



Rysunek 31.
Prezentacja obrazu rentgenowskiego z pomocą wybieranych okien w skali jednostek Hounsfielda przedstawianych w dostępnej dla człowieka skali szarości [źródło: 1]



Rysunek 32.
Wizualizacja tego samego obrazu rentgenowskiego w zależności od dobranych parametrów okna [źródło: 1]



Rysunek 33.
Doskonalenie obrazu rentgenowskiego przed jego prezentacją z wykorzystaniem modelowania komputerowego i grafiki komputerowej [źródło: 1]

ZAKOŃCZENIE

W tym wykładzie skupiono uwagę na jednej tylko metodzie obrazowania medycznego (klasycznej radiologii rentgenowskiej), pokazując, że także w przypadku tej ponad sto lat liczącej techniki obrazowania medycznego komputery znacząco polepszają możliwości penetracji wnętrza ciała człowieka dla potrzeb nowoczesnej diagnostyki i terapii.

Nieporównywalnie bogatsze i ciekawsze (ale i trudniejsze) są zastosowania komputerów w kontekście innych metod obrazowania medycznego, które tyko wymienimy:

- tomografia komputerowa rentgenowska (CT)
- wizualizacja za pomocą magnetycznego rezonansu jądrowego (NMR, MRI)
- tomografia emisji pozytronowej (PET)

- tomografia emisyjna pojedynczych fotonów (SPECT)
- obrazowanie radioizotopowe
- termowizja medyczna
- ultrasonografia

Omówienie tych metod badawczych wraz ze wskazaniem roli komputera w każdym z tych badań musi być jednak przedmiotem oddzielnych wykładów.

LITERATURA

1. Tadeusiewicz R., Śmietański J., *Pozyskiwanie obrazów medycznych oraz ich przetwarzanie, analiza, automatyczne rozpoznawanie i diagnostyczna interpretacja*, Wydawnictwo STN, Kraków 2011
2. Tadeusiewicz R., Augustyniak P. (red.), *Podstawy inżynierii biomedycznej – Tom 1 i 2.*, Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH, Kraków 2009
3. Tadeusiewicz R. (red.), *Inżynieria Biomedyczna – Księga współczesnej wiedzy tajemnej w wersji przystępnej i przyjemnej*, UWND AGH, Kraków 2008
4. Tadeusiewicz R., Korohoda P., *Komputerowa analiza i przetwarzanie obrazów*, Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków 1997; http://winntbg.bg.agh.edu.pl/skrypty2/0098/
5. Tadeusiewicz R., *Problemy biocybernetyki*, PWN, Warszawa 1993, (II wydanie poprawione); http://winntbg.bg.agh.edu.pl/skrypty2/0262/

Naśladowanie żywego mózgu w komputerze

Ryszard Tadeusiewicz
Katedra Automatyki, Akademia Górniczo-Hutnicza im. S. Staszica
Kraków
rtad@agh.edu.pl

Streszczenie

Wykład stanowi wprowadzenie do nowego typu narzędzi informatycznych, znanych pod nazwą **sieci neuronowe**. Narzędzia te, jeszcze niedawno traktowane nieufnie przez informatyków, dziś są szeroko stosowane ze względu na liczne zalety. Naśladując w komputerze ludzki mózg staramy się bowiem połączyć zalety komputera (łatwa dostępność i szybkość działania) z zaletami mózgu (zdolność uczenia się). Wykład zawiera wyłącznie teoretyczne elementy wiedzy na temat sieci neuronowych, w dodatku przedstawione w bardzo dużym skrócie. Można jednak swoją wiedzę poszerzyć korzystając ze wskazanych w treści wykładu zasobów internetowych. Z tego samego źródła można skorzystać w celu uzupełnienia wiedzy teoretycznej elementami praktyki posługując się dostępnymi darmowymi programami, pozwalającymi na samodzielne eksperymentowanie ze sztucznymi neuronami i z sieciami neuronowymi na domowym komputerze lub na poręcznym laptopie. Tekst wykładu dostarcza wiedzy potrzebnej do tego żeby taką samodzielną zabawę z sieciami neuronowymi sensownie zacząć.

Spis treści

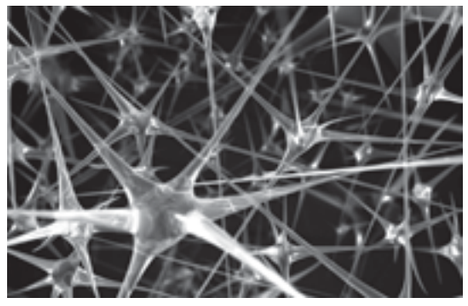
1 GENEZA SIECI NEURONOWYCH

Wykład stanowi wprowadzenie do nowego typu narzędzi informatycznych, znanych pod nazwą **sieci neuronowe**. Narzędzia te, jeszcze niedawno traktowane nieufnie przez informatyków, dziś są szeroko stosowane ze względu na liczne zalety.

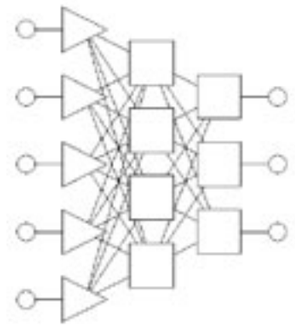
Pokazane na rysunku 1 metaforyczne połączenie elementów technicznych (układy scalone i inne elementy techniczne, z jakich zwykle składa się komputery) z kształtem ludzkiej głowy ma na celu uformowanie pierwszego skojarzenia, wskazującego na to, że omawiane dalej sieci neuronowe dziedziczą pewne właściwości po biologicznym pierwowzorze (mózgu), ale są w istocie specjalnego typu komputerami [1].

Rysunek 1.
Symboliczne wyobrażenie naśladowania mózgu w komputerze [źródło: https://cs.byu.edu/files/images/neural_networking.jpg, dostęp lipiec 2011]

Badacze zbudowali wiele systemów technicznych, naśladujących w komputerze ludzki mózg. Najbardziej pożyteczne okazały się **sieci neuronowe**. Badane przez biologów wnętrze mózgu ujawnia sieć powiązanych ze sobą elementów, tak zwanych **neuronów** (rys. 2). Mają one dosyć skomplikowaną budowę i funkcje, poznano je jednak na tyle dobrze, że potrafimy budować ich techniczne odpowiedniki – **sztuczne neurony**. Gdy się takie sztuczne neurony odpowiednio połączy, to uzyskuje się sztuczną sieć neuronową. Na rysunku 3 pokazano schemat przykładowej sieci neuronowej. Sztuczne neurony zaznaczono w postaci prostokątnych bloków, a połączenie miedzy neuronami symbolizują kreski łączące bloki. Trójkątne symbole oznaczają tzw. neurony wejściowe, służące do tego, żeby odebrać od użytkowników dane charakteryzujące zadanie do wykonania i rozesłać odpowiednie informacje w głąb sieci. Rozwiązanie zadania pojawia się w postaci sygnałów na liniach zakończonych kółkami (rys. 3).

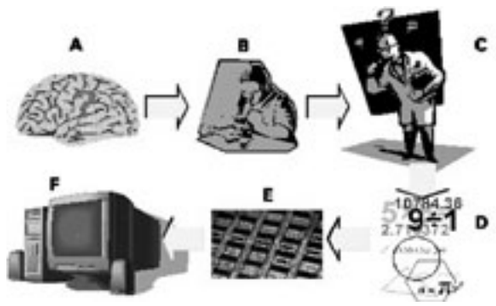


Rysunek 2.
Badanie wnętrza mózgu ujawniło w nim sieć powiązanych ze sobą elementów [źródło: http://encefalus.com/wp-content/uploads/2010/07/neural_network.jpg, dostęp lipiec 2011]



Rysunek 3.
Sztuczna sieć neuronowa powstaje, gdy się połączy ze sobą sztuczne elementy naśladujące rzeczywiste biologiczne neurony [źródło: 3]

Jak doszło do zbudowania pierwszych sieci neuronowych? Był to długi proces, przedstawiony schematycznie na rysunku 4. Punktem początkowym był mózg człowieka (A), a dokładniej – zafascynowanie ludzi jego skomplikowaną budową i tajemniczymi funkcjami. Badania naukowe biologów (B) dostarczyły wielu szczegółowych wyników, które stały się podstawą analiz, prowadzonych przez biocybernetyków (C), którzy na tej podstawie zbudowali wiele modeli matematycznych, opisujących mózg i jego działanie (D). Opierając się na tych modelach matematycznych neurocybernetycy zbudowali z kolei specjalizowane układy elektroniczne (tzw. **neurokomputery** E), które w swoim działaniu naśladują wybrane funkcje mózgu. Jednak w zastosowaniach praktycznych bardziej użyteczne okazały się sieci neuronowe symulowane za pomocą odpowiednich programów w strukturach zwykłych komputerów (F).

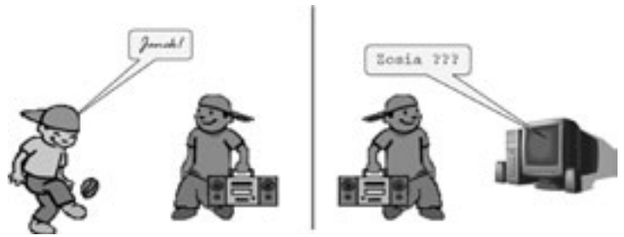


Rysunek 4.
Droga powstania sieci neuronowych: od badań naukowych do technicznych zastosowań [źródło: 2]

Co chcemy osiągnąć naśladując w komputerze ludzki mózg? Przecież komputery są dziś tak sprawne i tak dobrze zaspokajają wszelkie nasze potrzeby? Otóż naśladowanie mózgu w komputerze jest nam potrzebne między innymi do tego, żeby rozwiązać zadania, dla których nie potrafimy podać gotowych algorytmów ich rozwiązywania. Przypomnijmy sobie, że komputer potrafi zrobić tylko to, co mu narzucił programista, a programista opisuje to, co komputer powinien zrobić, tworząc odpowiedni algorytm. Jeśli więc potrafię rozwiązać jakieś zadanie i umiem podać opis postępowania prowadzącego do sukcesu (czyli właśnie algorytm), to komputer też potrafi rozwiązać zadnie posługując się tym algorytmem. Istnieją jednak liczne zadania, dla których nikt nie potrafi podać algorytmu.

Na pierwszy rzut oka wydaje się, że takie problemy, tak skomplikowane, że nikt na świecie nie potrafił podać algorytmu ich rozwiązywania, zdarzają się rzadko i dotyczą jakichś bardzo nietypowych sytuacji. Nieprawda! Przykładem zadania, które jest pozornie bardzo łatwe, a dla którego nikt nie umie podać algorytmu, jest automatyczne rozpoznawanie ludzi (rys. 5).

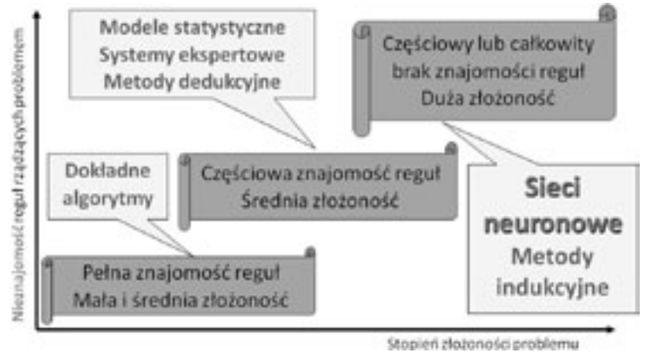
Obraz człowieka uzyskany za pomocą dowolnego cyfrowego aparatu (nawet zwykłej komórki) można łatwo wprowadzić do komputera, który może go zapamiętać, pokazać na ekranie, wydrukować, przesłać, a nawet przetworzyć, na przykład zmieniając rozmiary czy proporcje. Ale nikt nie potrafi napisać programu, dzięki któremu komputer sam zamieni miliony pikseli składających się na obraz na pewną i trafną decyzję: *To jest Janek, to na pewno on i nikt inny*!



Rysunek 5.
Zadanie rozpoznawania jako przykład zadania łatwego dla ludzi, dla którego jednak nie można stworzyć skutecznego algorytmu. Po lewej łatwe rozwiązywanie tego problemu przez człowieka, po prawej nieskuteczne rozwiązywanie tego problemu przez komputer [źródło: 2]

Uwzględniając to, co powiedziano wyżej, można teraz odpowiedzieć na pytanie: Kiedy sieci neuronowe są lepsze od innych metod informatycznych?

Dla dokładnego wskazania tego obszaru narysujemy prosty wykres (rys. 6), służący do scharakteryzowania i opisania różnych problemów informatycznych. Na osi poziomej tego wykresu zaznaczamy, jak bardzo rozważany problem jest **trudny.** Łatwiejsze do rozwiązania zadania umieścimy po lewej stronie, a im trudniejsze zadanie będziemy chcieli rozwiązać, tym bardziej na prawo będziemy je umieszczać na pokazanej „mapie". Tę miarę (że coś jest łatwiejsze albo trudniejsze do rozwiązania) nazwiemy **złożonością.** Zadania łatwe cechuje mała złożoność. Zadania trudne wiążą się z dużą złożonością. Na osi pionowej zaznaczymy z kolei stopień naszej **nieumiejętności** znalezienia algorytmu, za pomocą którego moglibyśmy problem rozwiązać. Jeśli dokładnie wiemy, jak rozwiązać zadanie, to umieścimy je nisko. Ale im więcej zagadek i trudności spodziewamy się napotkać, tym wyżej będziemy lokować nasze zadanie na rysowanej mapie.

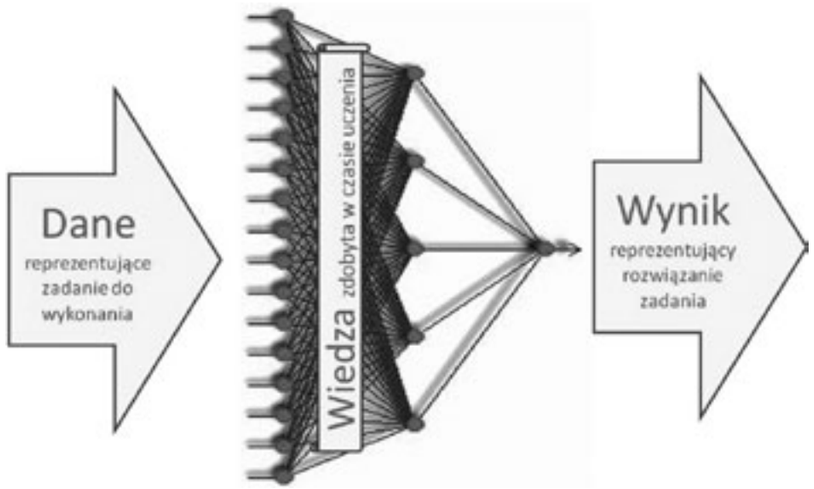


Rysunek 6.
Diagram ilustrujący, kiedy sieci neuronowe są lepsze od innych metod informatycznych [źródło: 2]

Tytułem przykładu na rysunku 6 zaznaczono trzy zadania.

W lewym dolnym rogu jest zadanie łatwe i w dodatku takie, dla którego znamy sposób rozwiązywania. Naturalnym sposobem wykonania takiego zadania jest wykorzystanie algorytmu, który w tym przypadku jest możliwy do opracowania. Pośrodku zaznaczono zadanie, w którym znajomość reguł jest tylko częściowa, co utrudnia zastosowanie algorytmu. W takim przypadku zamiast na dokładnym algorytmie możemy rozwiązanie oprzeć na opinii eksperta, którego zdroworozsądkowe rady mogą być zgromadzone w specjalnej bazie wiedzy, albo możemy skorzystać z namiastki wiedzy, jaką daje statystyka. Na przykład nie znamy reguł opisujących zachowanie w sklepie każdego konkretnego klienta i nie wiemy, czym się kieruje wybierając towary, ale **statystyka** pozwala przewidzieć, że jak się pojawi w telewizji reklama jakiegoś towaru, to więcej klientów go kupi. Taka metoda rozumowania, w której ogólna reguła (*reklama zwiększa sprzedaż*) pozwala na odgadywanie konkretnych zachowań, nazywa się **dedukcją**. Właśnie znajomość ogólnych reguł opisujących zachowania ludzi (a zwłaszcza kryminalistów) umożliwiała ujawnianie przestępców najsłynniejszemu detektywowi, jakim był niewątpliwie Sherlock Holmes.

Dedukcji nie da się zastosować, gdy rozważany problem jest całkowicie pozbawiony jakichkolwiek reguł dostępnych dla osoby, która próbuje go rozwiązać (jest to obszar zaznaczony na rysunku 6 w prawym górnym rogu). Brak reguł nie uniemożliwia jednak rozwiązania problemu, ponieważ ludzie potrafią w takich przypadkach stosować wnioskowanie przez analogię. Jeśli znamy kilka przykładów poprawnie rozwiązanych zadań, to potrafimy często odgadnąć prawidłowy wynik kolejnego zadania tego samego typu. Taki typ rozumowania, oparty na przykładach, a nie na regułach, nazywa się **indukcją.** Otóż sieci neuronowe są najlepszym znanym obecnie narzędziem informatycznym zdolnym do automatycznego prowadzenia rozumowania indukcyjnego. Odpowiedź na pytanie, dlaczego tak jest, stanie się jasna, gdy poznamy sposób rozwiązywania problemów przy użyciu sieci neuronowej. Wprawdzie o budowie sieci nie było jeszcze mowy, ale spojrzymy na razie, jak taka sieć działa, traktowana chwilowo jako całość (rys. 7). Na rysunku widzimy sieć jako pewną strukturę połączonych ze sobą elementów. Domyślamy się, że te elementy to sztuczne neurony, ale do budowy sieci jeszcze wrócimy i poznamy ją znacznie dokładniej. Po lewej stronie jest strzałka reprezentująca dane wejściowe. Tutaj także nie pokazano jeszcze żadnych konkretów, ale widzimy w jaki sposób stawia się sieci zadanie do rozwiązania – trzeba wszystkie jego istotne elementy przedstawić w postaci sygnałów możliwych



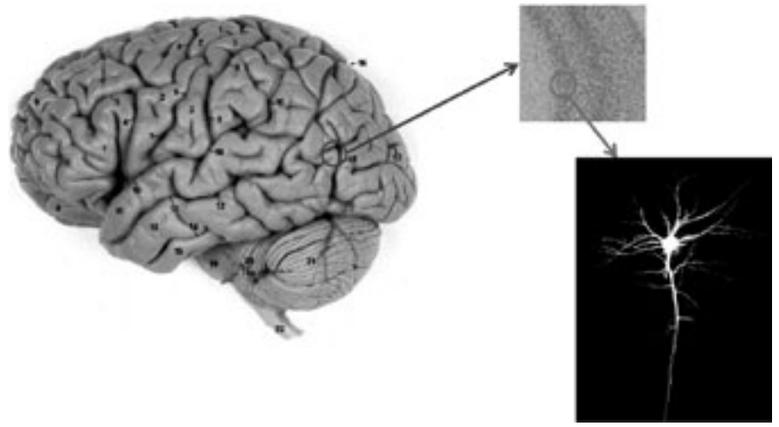
Rysunek 7.
Ilustracja działania sieci neuronowej [źródło: 2]

do wprowadzenia do neuronów. Wymaga to czasem dokładnego przemyślenia tego, co w danej kwestii jest ważne i jak te ważne informacje podać, żeby mogły być przyjęte przez sieć, ale to się opłaca. Również rozwiązanie zadania ma postać sygnału pojawiającego się na wyjściu neuronu, który jest ostatnim elementem sieci. Trzeba zatem tak sformułować zadanie, żeby sygnał z wyjściowego neuronu (zwykle mający formę liczby przyjmującej wartości z przedziału od 0 do 1) można było zinterpretować (zrozumieć) jako odpowiedź na pytanie, które zostało postawione w zadaniu. Czasami jest więcej niż jedno wyjście z sieci, bo chcemy uzyskać odpowiedzi na kilka pytań równocześnie, ale tym zagadnieniem zajmiemy się nieco dalej.

Sieć potrafi wypracować taką odpowiedź na podstawie danych wejściowych, ponieważ ma pewną wiedzę. Rysunek sugeruje, gdzie się ta wiedza mieści: w połączeniach pomiędzy neuronami. Odpowiedni napis na rysunku wskazuje też, skąd się ta wiedza bierze – powstaje automatycznie w trakcie procesu uczenia, który niebawem omówimy.

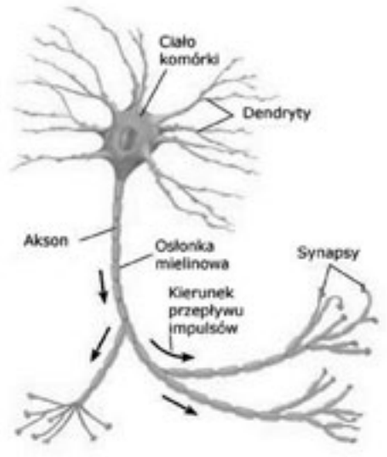
2 BUDOWA SIECI NEURONOWYCH

Naśladując w komputerze ludzki mózg staramy się wzorować na jego budowie. Analizując tę budowę coraz dokładniej najpierw widzimy cały mózg, potem wycinek jego kory, a na końcu – pojedynczą komórkę nerwową będącą głównym budulcem tej kory (patrz rysunek 8). Na rysunku należy zwrócić uwagę na zmienną skalę poszczególnych jego części: cały mózg ma rozmiar kilkunastu centymetrów, pokazany fragment kory mózgowej ma szerokość i wysokość 1 mm, a uwidoczniony pojedynczy neuron ma średnicę 20 mikrometrów.



Rysunek 8.
Etapy poznawania budowy i elementów składowych mózgu [źródło: 1]

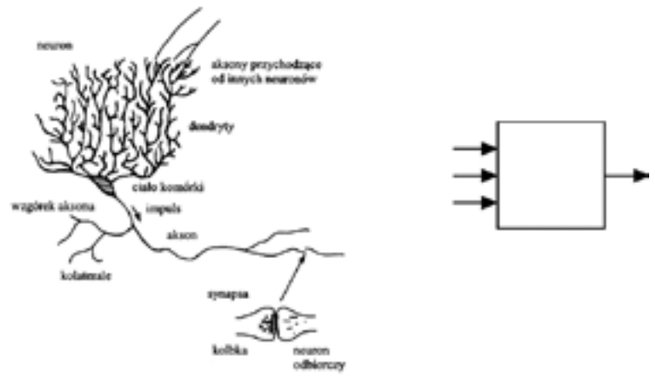
Tworząc modele mózgu nadające się do umieszczenia ich w komputerze zaczynamy od podstawowego elementu składowego, jakim jest **sztuczny neuron**. Stanowi on bardzo uproszczony model rzeczywistego biologicznego neuronu. Rzeczywista komórka nerwowa, wchodząca w skład mózgu, ma dosyć skomplikowaną budowę, pokazaną na rysunku 9. Elementy składowe komórki mają następujące funkcje: **dendryty** zbierają sygnały wejściowe i doprowadzają je do **ciała komórki**, które jest biologicznym procesorem analizującym te sygnały. Wynik tej analizy zbierany jest przez **akson**, który jest izolowany jak kabel elektryczny za pomocą **osłonki mielinowej**. Impulsy wyjściowe neuronu są przekazywane do następnych neuronów za pomocą **synaps**, w których lokują się procesy uczenia.



Rysunek 9.
Budowa biologicznej komórki nerwowej [źródło: 3]

Przy opisie pojedynczego rzeczywistego neuronu (biologicznej komórki nerwowej będącej częścią mózgu) warto odnotować jeszcze jedną ciekawostkę, widoczną na rysunku 8. Otóż ciała komórek są pozbawione osłonki mielinowej, więc tam, gdzie są one nagromadzone w dużych ilościach (głównie w korze mózgowej, ale także i w innych strukturach anatomicznych, na przykład we wzgórzu albo w jądrach pnia mózgu), tkanka nerwowa ma charakterystyczną szarą barwę, bo taki jest naturalny kolor tych komórek. Natomiast tam, gdzie biegną aksony łączące neurony w sieci, kolor (biały z perłowym połyskiem) całej tkance nadają osłonki mielinowe przypisane do tych biologicznych „kabli połączeniowych". Stąd już starożytni badacze mózgu wyróżniali w nim „substancję szarą" oraz „substancję białą", zaś w wielu kontekstach mówi się o „szarych komórkach" mając na myśli siedlisko naszej inteligencji.

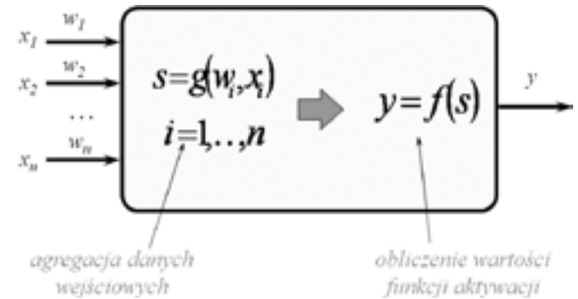
Budując sztuczne neurony, z których są tworzone sieci neuronowe, staramy się w nich odwzorować jedynie najpotrzebniejsze cechy biologicznych neuronów. Z komórki, która ma skomplikowaną budowę, pozostaje więc tylko blok przetwarzający informacje (odpowiednik ciała komórki), duża liczba kanałów wejściowych, odpowiadających dendrytom i jeden kanał wyjściowy, będący odpowiednikiem aksonu (rys. 10).



Rysunek 10.
Przejście pomiędzy biologicznym neuronem i jego sztucznym modelem [źródło: 3]

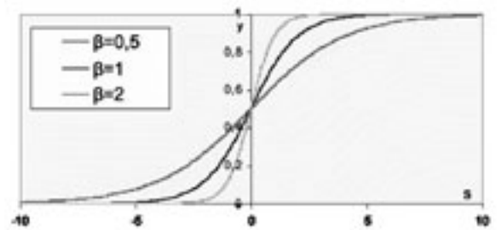
Sztuczny neuron musi się zachowywać w sposób maksymalnie podobny do zachowania biologicznego neuronu, a jednocześnie nie może być zbyt skomplikowany, bo jego realizacja techniczna będzie wtedy zbyt kosztowna. Trzeba bowiem pamiętać, że dla zbudowania sieci neuronowej, która będzie służyć do rozwiązywania jakiegoś praktycznego zadania, będziemy potrzebowali na ogół kilkudziesięciu lub nawet kilkuset neuronów. Dlatego przy tworzeniu sztucznych neuronów stosujemy wyłącznie trzy operacje, przedstawione schematycznie na rysunku 11:

- **Agregacja danych wejściowych**. Skoro neuron ma wiele wejść, a na tych wejściach wiele sygnałów x_1, x_2, ..., x_n, zaś wyjście jest tylko jedno – to jest oczywiste, że te wejściowe sygnały trzeba zintegrować, czyli jakoś połączyć razem, aby wyprodukować ten jeden sygnał wyjściowy. W najprostszym przypadku można sygnały wejściowe zsumować, ale bywają też bardziej wyrafinowane metody agregacji. Sygnał po agregacji jest oznaczany przez *s*.
- Sygnały wejściowe trzeba **zróżnicować** (pod względem skutków ich oddziaływania na neuron). Do tego celu służą współczynniki nazywane **wagami.** Z każdym wejściem o numerze *i*, do którego dociera z zewnątrz sygnał x_i, jest związany współczynnik wagi w_i. Znaczenie tego współczynnika będzie za chwilę omówione nieco dokładniej.
- Gdy już sygnały wejściowe zostały zróżnicowane przez wagi oraz zagregowane przez specjalną funkcję $g(w_i,x_i)$ (gdzie *i* zmienia się od 1 do *n*) – trzeba ustalić, jaki sygnał *y* wyśle neuron na swoim wyjściu (odpowiadającym aksonowi). Wiąże się to z zastosowaniem **funkcji aktywacji** $f(s)$.



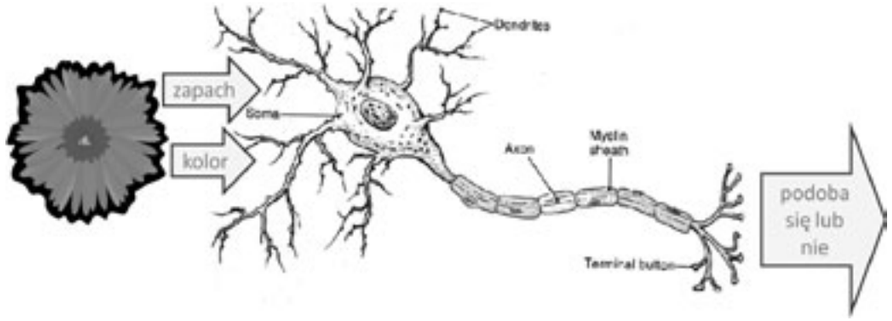
Rysunek 11.
Operacje uwzględnione w sztucznym neuronie

Agregacja sygnałów wejściowych jest raczej łatwa do przeprowadzenia. Jak wspomniano wyżej, jest to zwykle po prostu sumowanie. Funkcja aktywacji też jest łatwa do zrozumienia, więc ograniczymy się do pokazania jej przykładowego kształtu (rys. 12). Na tym wykresie zmienna *s* (sumaryczne pobudzenie neuronu) jest odkładana wzdłuż osi poziomej, a sygnał wyjściowy *y,* wysyłany poprzez akson neuronu, jest odczytywany z osi pionowej. Parametr β służy do doboru kształtu funkcji zależnie od potrzeb.



Rysunek 12.
Jedna z najpopularniejszych funkcji aktywacji neuronu, tak zwana **sigmoida**

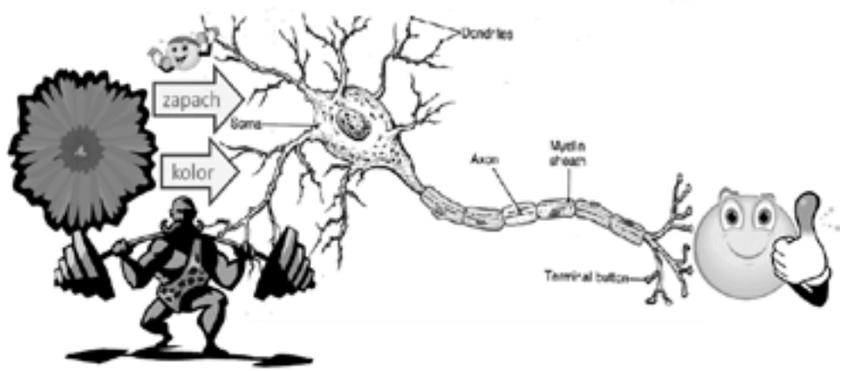
Zagadnieniem, któremu warto się przyjrzeć uważniej, jest kwestia **wag** różnicujących wejścia do neuronu. Wejścia te mogą być związane z różnymi cechami obserwowanego przez neuron obiektu. Na przykład na rysunku 13 pokazano neuron, do którego docierają dwa sygnały opisujące obserwowany kwiat. Jeden z tych sygnałów niesie informację o zapachu, a drugi o kolorze tego obiektu. Neuron może wysłać na wyjściu sygnał, że dany kwiat mu się podoba albo nie.



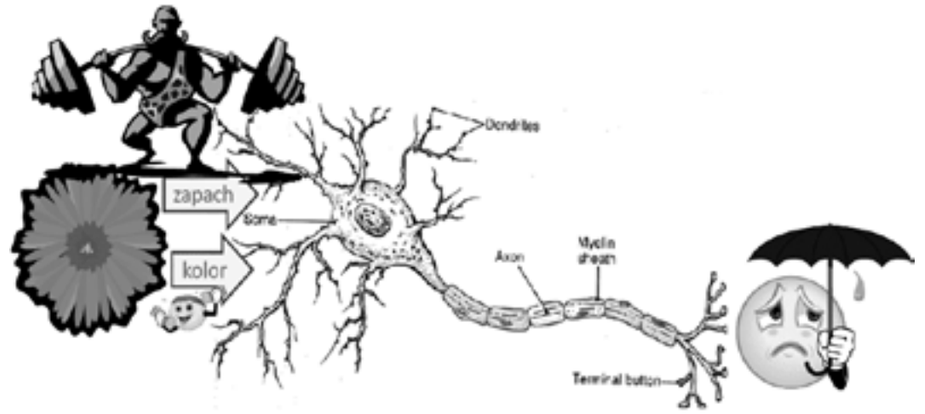
Rysunek 13.
Znaczenie wag na wejściach neuronu [źródło: 2]

Załóżmy, że oceniany kwiat ma ładny kolor, ale brzydki zapach. Jeżeli do sygnału wejściowego „zapach" przypiszemy małą wagę, a do sygnału „kolor" wagę dużą, to neuron wyśle na wyjściu sygnał, że kwiat mu się podoba (rys. 14). Natomiast przy przeciwnym ustawieniu wag, ocena wystawiona przez neuron będzie negatywna (rys. 15). Przy tym samym zestawie sygnałów wejściowych mamy dwie całkiem różne reakcje neuronu.

Wagi przypisywane wejściom neuronów mogą być nie tylko duże i małe, ale także dodatnie oraz ujemne, co oznacza że pewne cechy ocenianych obiektów, chociaż obiektywnie pozytywne, będą negatywnie oceniane przez neuron. Na przykład można by było sobie wyobrazić, że neuron będzie negatywnie reagował na ładnie pachnące kwiaty (po prostu taki będzie miał gust). W takim przypadku na pierwszym wejściu będzie można ustawić ujemną wartość wagi. Albo można sobie wyobrazić, że waga związana z kolorem będzie wynosiła zero. Wówczas kolor nie będzie wcale wpływał na ocenę podawaną przez neuron, co może odpowiadać sytuacji daltonisty. Przykładów można by było mnożyć bez liku.

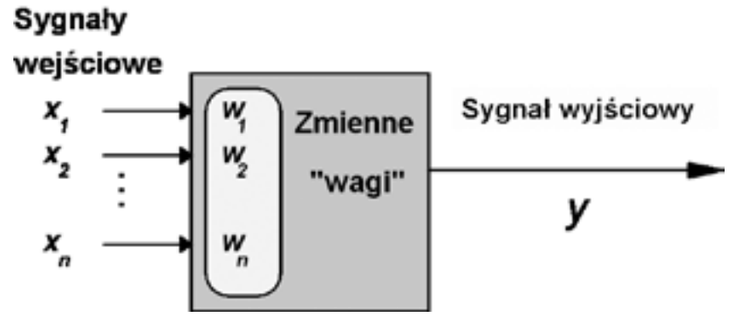


Rysunek 14.
Określone ustawienie wag neuronu daje pozytywną odpowiedź neuronu [źródło: 2]



Rysunek 15.
Odwrotne ustawienie wag daje odwrotny efekt [źródło: 2]

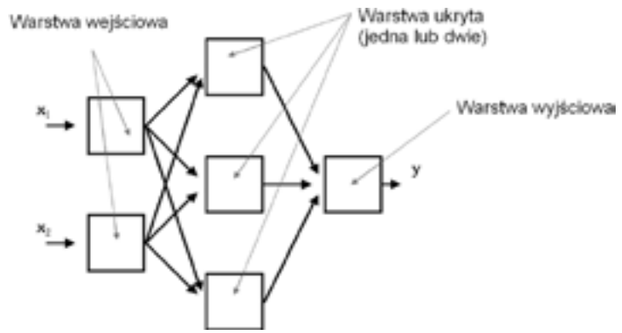
O sztucznym neuronie można by jeszcze długo opowiadać, ale poprzestaniemy na tym, że pokażemy jego zbiorczy schemat (rys. 16), uwzględniający wszystkie do tej pory ustalone fakty, bowiem najwyższa już pora zacząć budować sieci. Model z rysunku 16 będzie służył dalej jako podstawowa „cegiełka" przy budowie sieci neuronowych.



Rysunek 16.
Schemat właściwości sztucznego neuronu [źródło: 3]

Pytanie, które sobie teraz trzeba postawić, brzmi: Jak połączyć sztuczne neurony, żeby powstała użyteczna sieć? Współczesne sieci neuronowe buduje się z reguły z neuronów układanych w warstwy (rys. 17). Wyjaśnimy teraz kolejno, jaką rolę odgrywają poszczególne warstwy sieci:

- **Warstwa wejściowa** służy do wprowadzenia danych, które są potrzebne do rozwiązania postawionego zadania. Warstwa ta nie uczestniczy z reguły w procesie uczenia.
- **Warstwa ukryta** analizuje dane i przygotowuje przesłanki do rozwiązania zadania. To tutaj mieści się głównie inteligencja sieci.
- **Warstwa wyjściowa** wyznacza ostateczne rozwiązanie zadania i podaje je do wykorzystania.



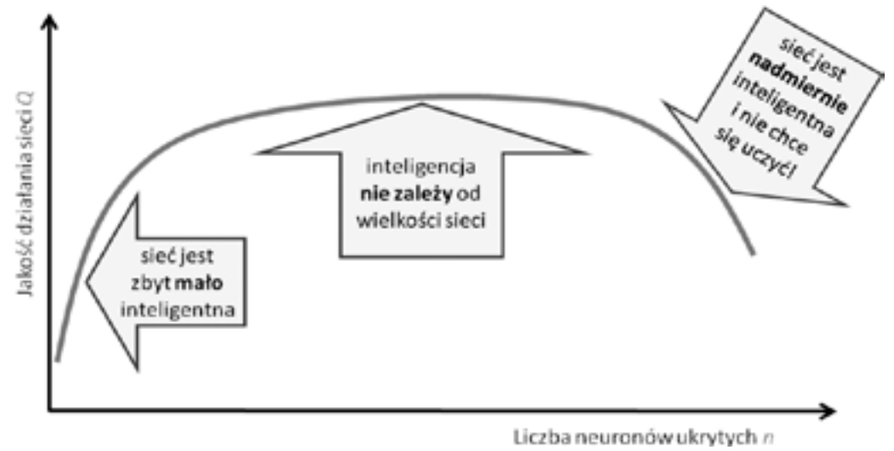
Rysunek 17.
Przykładowa sieć z jej wszystkimi ważnymi elementami [źródło: 3]

Sieci neuronowe o budowie warstwowej najczęściej służą do rozwiązywania konkretnych zadań, ponieważ takie sieci w prosty sposób się buduje, łatwo się ich nauczyć, wygodnie się ich używa. Nie są one tak całkiem pozbawione merytorycznego uzasadnienia. Na rysunku 18 pokazano przykładową sieć neuronową i obraz mikroskopowy fragmentu kory mózgowej, odpowiedzialnego za analizę wrażeń wzrokowych. Wnioski nasuwają się same.

Rysunek 18.
Budowa typowej sztucznej sieci neuronowej i przekrój kory mózgowej [źródło: 2]

3 INTELIGENCJA SIECI NEURONOWYCH

Teoretycznie twórca sieci może dowolnie wybrać wszystkie jej elementy. W rzeczywistości jednak swoboda twórcy sieci jest ograniczona, bo liczba neuronów w warstwie wejściowej zależy od liczby posiadanych danych, a wielkość warstwy wyjściowej zależy od tego, jakie chcemy dostać wyniki. Twórca sieci neuronowej może więc głównie mieć wpływ na liczbę neuronów ukrytych *n*. Decyduje ona o jakości działania sieci *Q*, co pokazano na rysunku 19. Oglądając ten rysunek warto zwrócić uwagę na interpretację miary jakości *Q* (oznaczenie to pochodzi od angielskiego słowa *Quality* oznaczającego właśnie *jakość*). Oczywiście będzie ona różnie definiowana dla różnych zadań wykonywanych przez sieć, ale na tym etapie rozważań możemy założyć, że sieć powinna rozpoznawać jakieś obiekty (ludzkie twarze, ręcznie pisane litery, odciski palców itp.). W takim przypadku *Q* może po prostu pokazywać procent poprawnie rozpoznanych obiektów.



Rysunek 19.
Zależność jakości działania sieci *Q* od liczby neuronów ukrytych *n*

Kształt zależności jakości działania sieci *Q* od liczby neuronów ukrytych *n* wykazuje istnienie trzech wyraźnie odmiennych obszarów, wskazanych na rysunku 19 przez odpowiednio opisane strzałki.

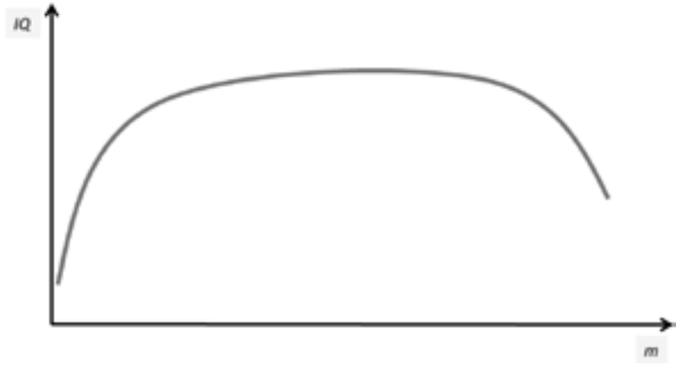
Pierwszy obszar odpowiadający zbyt małej liczbie neuronów ukrytych to obszar, w którym sieć jest zbyt mało inteligentna, żeby nauczyć się rozwiązywać postawione zadanie.

Drugi obszar cechuje się tym, że praktycznie nie ma związku między wielkością sieci, a jakością jej działania. Taką samą jakość *Q* mogą wykazywać (po procesie uczenia) sieci o różnej liczbie neuronów ukrytych *n*.

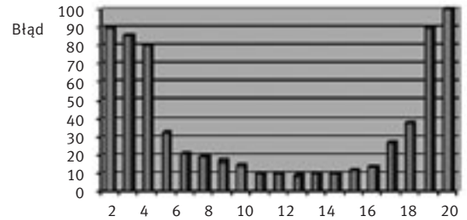
Najbardziej zagadkowy jest trzeci obszar, w którym sieć pogarsza swoje działanie w miarę jak przybywa jej elementów. W tym obszarze sieć jest już tak inteligentna, że potrafi skutecznie wykręcić się od uczenia i nie można nad nią zapanować. Fenomen ten będzie dokładniej omówiony przy dyskusji metod uczenia sieci. Zapamiętajmy jednak: zbyt duża inteligencja sieci jest szkodliwa!

W zasadzie można by było na tym poprzestać, popatrzmy jednak, co się stanie, kiedy na wykresie z rysunku 19 zamienimy *n* na *m* oraz *Q* na *IQ* (rys. 20). Wykres ten przedstawia teraz znaną z psychologii zależność miary inteligencji człowieka (IQ to tzw. **iloraz inteligencji**) od masy jego mózgu *m*. Oglądając ten rysunek należy zwrócić uwagę na to, że lewa część wykresu odpowiada przypadkom tzw. niedorozwoju umysłowego (*oligocefalii*), gdzie w zależności od stopnia niedoboru tkanki mózgowej mamy do czynienia z takimi przypadkami jak debilizm, imbecylizm, kretynizm itd. Środkowa część wykresu pokazuje, że u większości ludzi poziom inteligencji nie zależy od wielkości mózgu. Na przykład znany fakt biologiczny, że mózg typowej kobiety jest mniejszy niż mózg mężczyzny w żaden sposób nie przekłada się na różnice sprawności intelektualnej. Ciekawe jest też to, że osoby mające zbyt wielki mózg z reguły nie są w stanie prawidłowo funkcjonować w społeczeństwie i ich miara inteligencji bywa ponownie dramatycznie niska.

Przedstawione wyżej (zwłaszcza na rysunku 19) ogólne reguły wskazujące na niekorzystne skutki używania zarówno za małej, jak i za dużej sieci, potwierdzają wyniki przykładowego eksperymentu, przytoczone na rysunku 21. W eksperymencie tym oceniano błąd popełniany przez sieci neuronowe mające różną liczbę neuronów ukrytych. Widać, że tam, gdzie ogólny model przewidywał niską jakość działania sieci (mierzoną wskaźnikiem *Q*) konkretnie badana tu sieć popełniała dużo błędów. Dotyczyło to zarówno sieci o zbyt małej liczbie neuronów ukrytych, jak i sieci o liczbie neuronów zbyt dużej.



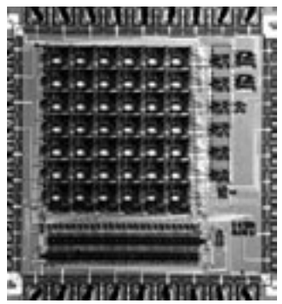
Rysunek 20.
Wykres z rysunku 19 po zmianie znaczenia zmiennych odkładanych na osiach nabiera nowego sensu, także informującego o czymś ciekawym



Rysunek 21.
Wyniki eksperymentu potwierdzającego zależność sprawności działania sieci neuronowej od liczby neuronów ukrytych w tej sieci [źródło: 2]

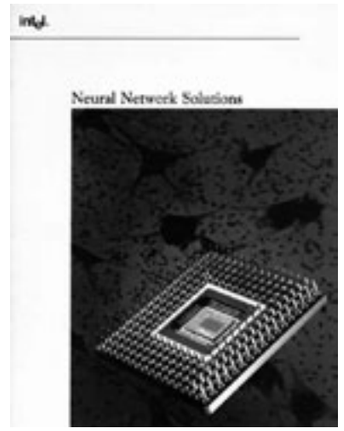
4 REALIZACJA SIECI NEURONOWYCH

Mając zaprojektowaną sieć neuronową, to znaczy wiedząc, z jakich elementów jest ona zbudowana (sztuczne neurony), ile tych elementów trzeba zastosować i jak te elementy są łączone pomiędzy sobą – można się zastanowić, jak tę sieć zrealizować. W najwcześniejszych pracach dotyczących budowy sieci neuronowych chętnie stosowano urządzenia elektroniczne, które modelowały sieć. Przykład tego typu układu przedstawiono na rysunku 22.



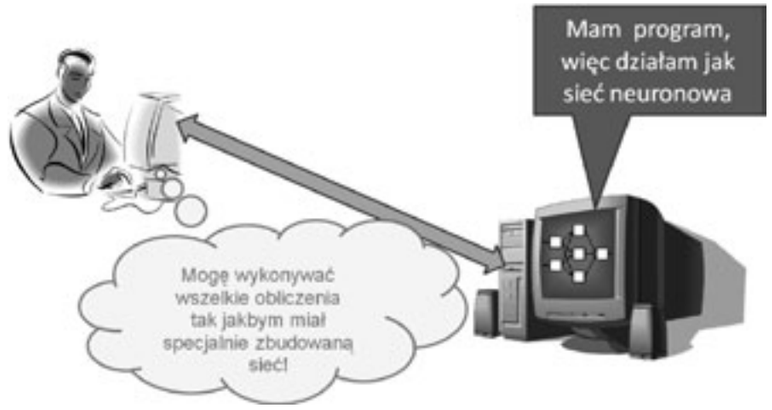
Rysunek 22.
Sieć neuronowa zbudowana z indywidualnych elementów elektronicznych [źródło: http://www.isn.ucsd.edu/papers/asic96/img50.gif, dostęp wrzesień 2011]

Obecnie urządzenia elektroniczne służące do budowy sieci neuronowych (rzadko jednak budowanych) mają postać specjalizowanych układów scalonych (rys. 23).



Rysunek 23.
Elektronicznych układ scalony służący do modelowania sieci neuronowych wyprodukowany przez Intel [źródło: http://www.warthman.com/images/intel%2080170%20B.jpg]

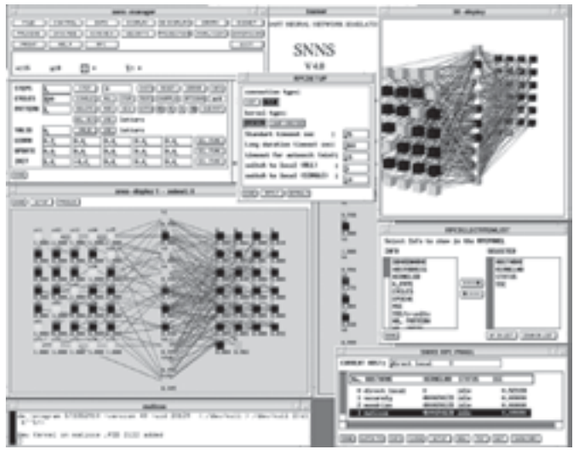
Najczęściej jednak do budowy sieci neuronowych wykorzystuje się program komputerowy, który w zwykłym komputerze modeluje sieć (rys. 24). Komputery potrafią modelować różne obiekty (na przykład statek kosmiczny albo pogodę), więc mogą także modelować sieć neuronową, jeśli jest ona dobrze opisana.



Rysunek 24.
Modelowana komputerowo sieć neuronowa jest szczególnie wygodna i łatwa w użyciu

Nazwy i loga przykładowych programów modelujących sieci neuronowe są podane na rysunku 25, a przykładowy wygląd ekranu komputera modelującego sieć neuronową pokazano na rysunku 26. Przedstawione programy bardzo dobrze modelują sieci neuronowe, mające jednak jedną poważną wadę: są bardzo drogie, więc kupują je tylko instytucje, które stosują sieci neuronowe.

Rysunek 25.
Nazwy i loga przykładowych programów modelujących sieci neuronowe [źródło: 2]

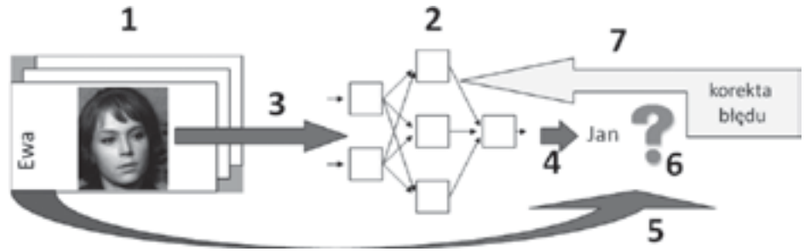


Rysunek 26.
Przykładowy wygląd ekranu komputera modelującego sieć neuronową [źródło: http://cortex.cs.nuim.ie/tools/spikeNNS/images/snns-picture.gif, dostęp wrzesień 2011]

5 PROCES UCZENIA SIECI NEURONOWEJ

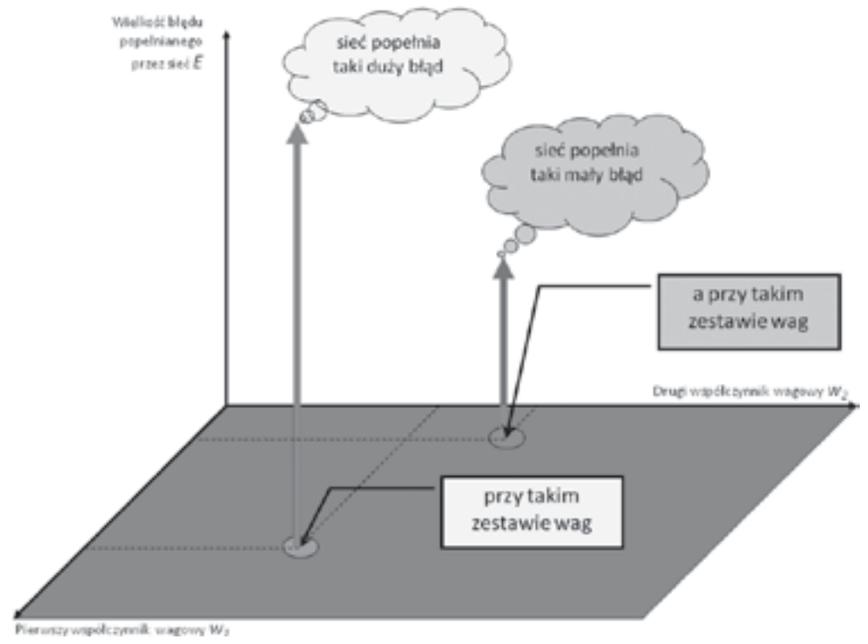
Przystąpimy teraz do omówienia procesu **uczenia** sieci neuronowych. Będziemy odwoływać się przy tym do numerów zaznaczonych na rysunku 27, który przedstawia maksymalnie uproszczony schemat tego procesu. Jego podstawą jest zbiór przykładowych danych wraz z rozwiązaniami, nazywany **zbiorem uczącym** (1). W przykładzie zbiór ten zawiera wizerunki osób, których sieć ma się nauczyć rozpoznawać oraz informacje, jakie jest poprawne rozwiązanie (tzn. kim jest osoba na zdjęciu). Uczenie polega na pokazywaniu sieci (2) kolejnych zadań – wizerunków osób (3), które sieć próbuje rozpoznać podając własne rozwiązania (4). W zbiorze uczącym są informacje o tym, jak naprawdę

nazywa się osoba na zdjęciu (5). Porównanie odpowiedzi sieci z prawidłowym rozwiązaniem umożliwia wyznaczenie błędu sieci (6). Uczenie prowadzone jest tak, żeby zminimalizować wartość błędu (7).



Rysunek 27.
Uczenie polega na takim poprawianiu parametrów sieci (wartości wag we wszystkich neuronach całej sieci), żeby krok po kroku, zadanie po zadaniu, zmniejszać błąd popełniany przez sieć. Kolejne numery oznaczają kolejne czynności [źródło: 2]

Pytanie, które się nasuwa, jest następujące: Skąd wiemy, w jaki sposób zmieniać parametry sieci (wartości wag), żeby uzyskać efekt zmniejszania błędu?



Rysunek 28.
Zależność błędu popełnianego przez sieć od współczynników wagowych

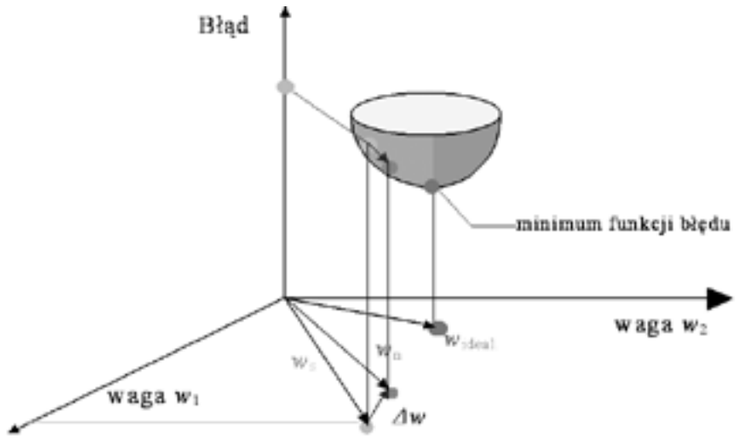
Zachowanie sieci jest wypadkową zachowania wszystkich jej neuronów, zaś zachowanie poszczególnych neuronów można uzależnić od wartości wag występujących w tych neuronach (por. rys. 14 i 15). Jeśli więc ustalimy wszystkie wagi we wszystkich neuronach całej sieci, a potem pokażemy sieci wszystkie zadania ze

zbioru uczącego, to wyznaczymy łączny błąd, popełniany przez sieć dla tych zadań. Dla różnych zestawów wag otrzymamy różne wartości łącznego błędu (rys. 28). Gdybyśmy takie strzałki wystawiali we wszystkich punktach ciemnej płaszczyzny, to powstałaby powierzchnia nazywana **powierzchnią błędu**, której za chwilę użyjemy do wyznaczenia sposobu uczenia. Najpierw jednak wyjaśnimy pewne wątpliwości, jakie się mogą nasuwać przy analizowaniu rysunku 28.

Po pierwsze w sieci zawierającej dużo neuronów (co jest regułą!) istnieje bardzo dużo współczynników wagowych, które trzeba ustalić w toku uczenia – a na rysunku pokazano tylko dwa. Czy to jest prawidłowe? Nie, to nie jest prawidłowe, ale takie uproszczenie trzeba przyjąć, aby umożliwić odzwierciedlenie tego na rysunku. Rysunek jest więc intuicyjną metaforą, a nie dokładną ilustracją rzeczywistych procesów zachodzących w prawdziwej sieci neuronowej.

Po drugie jak wyznaczyć łączny błąd sieci (dla danego zestawu wag), skoro w zbiorze uczącym jest wiele zadań, a w każdym z tych zadań sieć popełnia inny błąd? Otóż błędy ustalone dla poszczególnych zadań trzeba do siebie dodać, żeby błędy ujemne nie zmniejszały błędów dodatnich – wszystkie błędy przed sumowaniem podnosimy do kwadratu. Dzięki temu błędy są tylko dodatnie, a w dodatku dla dużych błędów jest większa „kara” niż dla małych, bo po podniesieniu do kwadratu trzy razy większy błąd oznacza dziewięć razy większą karę.

Na rysunku 29 pokazano przykładową powierzchnię błędu (szara) oraz uczenie sieci jako poszukiwanie minimum funkcji błędu. Istota uczenia polega na szukaniu miejsca (zestawu wag określonego jako w_{ideal}), w którym błąd jest minimalny. Na rysunku zaznaczony jest stary zestaw wag w_s (przed wykonaniem jednego kroku procesu uczenia), któremu odpowiada duża wartość błędu (oznaczona kropką na powierzchni błędu). Metoda uczenia potrafi na powierzchni błędu wyznaczyć kierunek najszybszego malenia błędu (krótka strzałka na płaszczyźnie). Na tej podstawie stary zestaw wag w_s zostaje zmodyfikowany o wartość Δw i powstaje nowy zestaw wag w_n, który jest bliższy zestawowi idealnemu w_{ideal}.



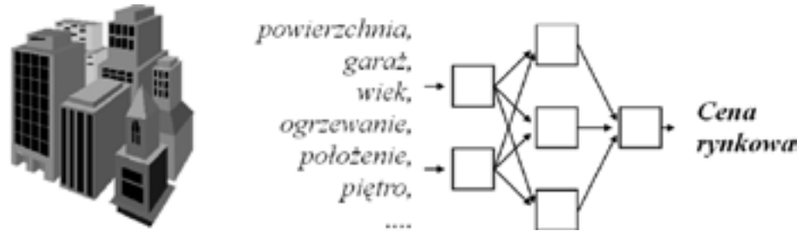
Rysunek 29.
Uczenie sieci jako poszukiwanie minimum funkcji błędu

Szczegółowe algorytmy uczenia wbudowane są zwykle w programy symulujące sieci neuronowe na komputerze, dlatego użytkownik sieci nie musi się tym osobiście zajmować. I dobrze, bo przecież opisana wyżej ogólna idea sposobu uczenia jest dość prosta i może być łatwo zrozumiana to jednak szczegóły praktycznej realizacji tej metody są dość skomplikowane i wymagają sporego wysiłku. Na szczęście nie naszego wysiłku tylko komputera.

6 ZASTOSOWANIA SIECI NEURONOWYCH

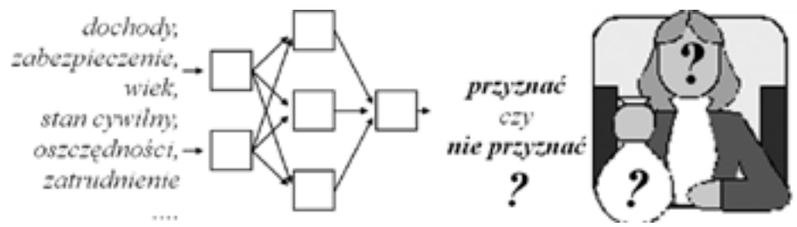
Sieci neuronowe mają wiele zastosowań. Przedstawimy krótko tylko dwa przykłady.

Tworzenie modelu procesu. Wyobraźmy sobie, że chcemy przewidzieć, jaką cenę osiągnie na wolnym rynku określone mieszkanie (rys. 30). Nie mamy gotowych reguł (bo nikt ich nie zna), ale możemy użyć jako zbioru uczącego opisu wcześniejszych transakcji kupna-sprzedaży. Na wejściu sieci są dane dotyczące mieszkania, a sieć ma podać jego cenę. Rozważając ten przykład warto zwrócić uwagę, że na wejściu sieci mogą się pojawić zarówno informacje dające się wyrazić ilościowo (na przykład powierzchnia mieszkania), jak i takie dane, które są opisowe (na przykład położenie, czyli informacja, w jakiej dzielnicy jest rozważane mieszkanie). Sieci neuronowe radzą sobie z danymi wszelkich typów.



Rysunek 30.
Zastosowanie sieci neuronowej do tworzenia modelu procesu

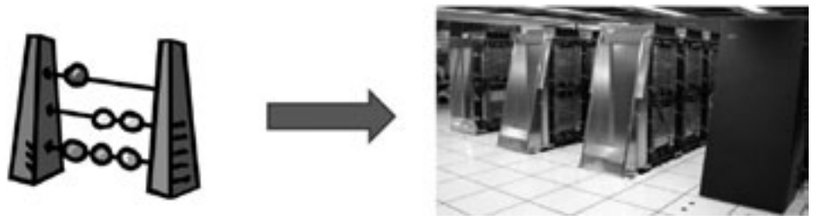
Podejmowanie decyzji. W banku trzeba zdecydować, czy przyznać konkretnemu klientowi pożyczkę, czy lepiej nie (rys. 31). Jak się nie pożyczy pieniędzy uczciwemu klientowi, to bank nie zarobi. Ale jak się pożyczy nieuczciwemu, to bank poniesie stratę. Nie wiadomo, po czym poznać nierzetelnego klienta, ale można dać sieci jako zbiór uczący informacje o wszystkich udzielonych pożyczkach, tych udanych i nie. Sieć się sama nauczy rozpoznawać nieuczciwych klientów i może nam radzić. Rozważając ten przykład warto podkreślić, że wielokrotnie wzmiankowane wcześniej zagadnienia rozpoznawania (na przykład ludzkich twarzy) są przykładem podejmowania decyzji przy pomocy sieci neuronowej.



Rysunek 31.
Zastosowanie sieci neuronowej do wspomagania procesu podejmowania decyzji

ZAKOŃCZENIE

Sieci neuronowe powstały w wyniku procesu twórczego, przeciwnego do tego, który doprowadził do powstania typowych komputerów. Komputery powstały bowiem w taki sposób, że stosunkowo proste (początkowo) urządzenia przeznaczone do mechanizacji obliczeń: liczydła, suwaki, kalkulatory itd. poddano procesowi intensywnego doskonalenia, dzięki czemu powstały znane nam obecnie systemy informatyczne, o ogromnych możliwościach, ale też niezwykle skomplikowane (rys. 32).



Rysunek 32.
Ewolucja informatyki – od liczydła do superkomputera

Z sieciami neuronowymi było przeciwnie: Za punkt wyjścia przyjęto niesłychanie skomplikowany twór, jakim jest mózg, i podjęto próbę modelowania jego struktury i właściwości za pomocą opisów, które w miarę ich doskonalenia stawały się coraz prostsze. Obecnie używane sieci neuronowe są tak bardzo uproszczone, że każdy może zrozumieć ich budowę i działanie, a jednocześnie zachowały one tyle właściwości oryginalnego mózgu, że potrafią się bardzo inteligentnie zachowywać (rys. 33).

Rysunek 33.
Ewolucja neurocybernetyki – od mózgu do sieci neuronowej

Zasadniczą cechą użytkową, odróżniającą sieci neuronowe od typowych, ogólnie znanych i powszechnie stosowanych komputerów, jest ich zdolność do samodzielnego nabywania wiedzy w procesie uczenia się. Komputery mogą bardzo szybko i dokładnie wykonywać rozmaite czynności, czasem bardzo skomplikowane i ogromnie pożyteczne, ale robią to tylko wtedy, gdy człowiek wcześniej dokładnie określi, co i jak mają robić. To człowiek zasila komputer wiedzą, tworząc algorytm i pisząc na jego podstawie program. Natomiast sieci neuronowe nie wymagają programowania. Wystarczy, że pokażemy sieci trochę przykładów poprawnie rozwiązanych zadań, a sieć sama zgromadzi potrzebną wiedzę i potrafi potem rozwiązywać podobne zadania. Jest to bardzo wygodne, a ponadto umożliwia rozwiązywanie także takich zadań, dla których nikt nie potrafi napisać algorytmu! Dlatego sieci neuronowe są dziś bardzo chętnie stosowane i dlatego warto je poznać jako fascynujące narzędzia nowoczesnej informatyki.

Jednak nie tylko sprawność działania i wygoda stosowania powoduje, że sieciami neuronowymi zajmuje się coraz większe grono badaczy i praktyków na całym świecie. Dodatkowy powód jest taki, że mimo ogromnych uproszczeń sieci te zachowały wiele elementów podobieństwa do naszego mózgu (od którego badania zaczęła się droga, która doprowadziła do powstania tych sieci). Dlatego używając sieci i obserwując procesy w nich zachodzące poznajemy także jedną z najbardziej fascynujących tajemnic Natury: zagadkę ludzkiego intelektu...

LITERATURA

1. Tadeusiewicz R. (red.), *Neurocybernetyka teoretyczna*, Wydawnictwo Uniwersytetu Warszawskiego, Warszawa 2009
2. Tadeusiewicz R., Gąciarz T., Borowik B., Leper B., *Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#*, Wydawnictwo Polskiej Akademii Umiejętności, Kraków 2007; http://home.agh.edu.pl/~tad//
3. Tadeusiewicz R., *Elementarne wprowadzenie do sieci neuronowych z przykładowymi programami*, Akademicka Oficyna Wydawnicza, Warszawa 1998; http://winntbg.bg.agh.edu.pl/skrypty2/0263/
4. Tadeusiewicz R., *Sieci neuronowe*, Akademicka Oficyna Wydawnicza, Warszawa 1993; (wyd. I i II wydanie poprawione); http://winntbg.bg.agh.edu.pl/skrypty/0001/

Od złamania Enigmy do współczesnej kryptologii

Jerzy Gawinecki

Instytut Matematyki i Kryptologii, Wydział Cybernetyki
Wojskowa Akademia Techniczna
jgawinecki@wat.edu.pl

Streszczenie

Wykład składa się z trzech części, z których pierwsza jest poświęcona ukazaniu roli kryptologii w wielu historycznych wydarzeniach począwszy od czasów starożytnych po złamanie Enigmy. Druga część przedstawia dokonania polskich kryptologów: Mariana Rejewskiego, Jerzego Różyckiego i Henryka Zygalskiego w złamaniu Enigmy. Ma na celu wykazanie, że to Polacy dokonali przełomu w kryptoanalizie, stosując metody matematyczne oparte na teorii permutacji. Od tej pory datuje się przełom w kryptoanalizie. Zostanie podkreślone również znaczenie złamania Enigmy dla skrócenia czasu trwania II wojny światowej. Część trzecia wykładu natomiast jest poświęcona współczesnej kryptologii od złamania Enigmy do współczesnych dokonań. Podkreślone zostaną zagrożenia, jakie niesie ze sobą wykorzystanie komputerów w przesyłaniu informacji, ze szczególnym uwzględnieniem cyberterroryzmu.

Spis treści

Tam sięgaj, gdzie wzrok nie sięga.
Łam, czego rozum nie złamie.
Adam Mickiewicz

1 ZANIM ZŁAMANO ENIGMĘ

Ogromnym przełomem w kryptologii było użycie w kryptoanalizie, a później także w kryptografii, metod matematycznych. Uczynili to po raz pierwszy przed II wojną światową, polscy matematycy. Zastosowanie przez nich ***teorii permutacji*** umożliwiło kryptoanalitykom złamanie niemieckiego szyfru maszyny Enigma, co z kolei miało wpływ, według słów premiera Wielkiej Brytanii Winstona Churchilla, na skrócenie wojny o dwa, trzy lata.

Ale sukcesy polskiej kryptoanalizy w walce z Enigmą nie były pierwszymi. Najnowsze prace historyczne odtajniły niedawno fakt łamania przez Polskę radzieckich szyfrów podczas wojny w 1920 roku. Oznacza to, że polska szkoła kryptoanalizy nie zaczęła się od zatrudnienia matematyków do łamania Enigmy, ale była wynikiem prac rozpoczętych wraz z odzyskaniem niepodległości.

Najbardziej znany specjalista od historii kryptologii, profesor Uniwersytetu w Oksfordzie David Kahn, autor książki *Łamacze kodów* twierdzi, że dobrzy specjaliści z kryptologii pochodzą z krajów wielojęzycznych i wielokulturowych. I jak mówi „muszą być tam uniwersytety z matematyką na wysokim poziomie i coś jeszcze – improwizacja: musi być tam komponowana muzyka”[3]. Bo matematyka to logiczne myślenie, a muzyka to kwestia wyobraźni.

Jeszcze przed sukcesem złamania szyfrów Enigmy, w latach dwudziestych w łamaniu sowieckich szyfrów brali udział zatrudnieni po raz pierwszy w historii kryptologii wybitni polscy profesorowie matematyki: Stefan Mazurkiewicz, Wacław Sierpiński i Stanisław Leśniewski.

Wybitny polski matematyk, profesor Wacław Sierpiński, twórca polskiej szkoły matematycznej, autor prac z dziedziny teorii funkcji rzeczywistych, teorii liczb i teorii mnogości wraz z profesorem Stefanem Mazurkiewiczem, jednym z założycieli słynnej na cały świat warszawskiej szkoły logicznej, i profesorem Stanisławem Leśniewskim – specjalistą od logiki matematycznej z Wydziału Matematyki i Nauk Przyrodniczych Uniwersytetu Warszawskiego, wspólnymi siłami złamali około setki rosyjskich szyfrów podstawieniowych, którymi posługiwali się dowódcy wojsk rosyjskich. Pozwoliło to w oparciu o systematycznie prowadzone nasłuchy radiowe stacji bolszewickich na przechwycenie i rozszyfrowanie kilku tysięcy rosyjskich radiodepesz, a tylko w samym sierpniu 1920 roku, czyli w okresie przesilenia wojny, blisko pięciuset.

Rysunek 1.
Odszyfrowany szyfr sowiecki MARS [źródło: 7, s. 361]

Złamanie rosyjskich szyfrów dostarczyło Sztabowi Generalnemu szerokiej i pełnej wiedzy o przeciwniku i dawało polskiej stronie znaczną przewagę w trzech wymiarach: taktycznym, operacyjnym i strategicznym. Można powiedzieć, że marszałek Józef Piłsudski miał tak dokładne informacje, jakich do jego czasów nie miał żaden dowódca w żadnej wojnie. W konsekwencji doprowadziło to do Cudu nad Wisłą i zatrzymania nawałnicy bolszewickiej, co jak wiemy zmieniło historię Polski i Europy.

Do rangi symbolu urasta bolszewicki szyfr Rewolucja. Złamali go wspólnie w sierpniu 1920 roku dwaj polscy kryptoanalitycy – bardzo zasłużony, pochodzący z Łodzi chemik, talent matematyczny i lingwistyczny, porucznik Jan Kowalewski, odznaczony za zasługi w roku 1921 przez szefa Sztabu Generalnego gen. Władysława Sikorskiego najwyższym odznaczeniem państwowym Krzyżem Virtuti Militarii, oraz profesor Stefan Mazurkiewicz, o którym już wcześniej wspomniano.

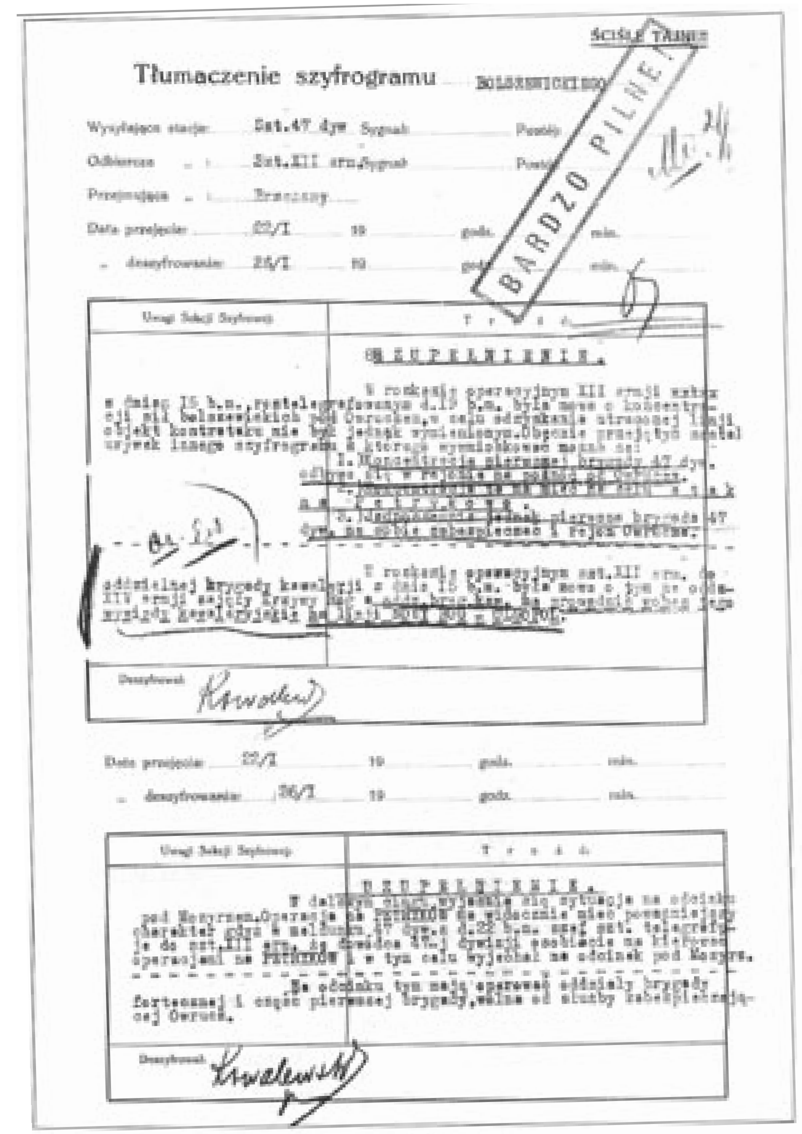
Rysunek 2.
Mjr Jan Kowalewski [źródło: 7, s. 247]

Złamanie szyfru Rewolucja i dziesiątków innych szyfrów rosyjskich przez Polaków, profesorów matematyki, profesorów logiki, studentów i oficerów, zaowocowało później złamaniem niemieckiego szyfru maszynowego Enigma.

W czasach współczesnych ujawniono skrywane przez dziesięciolecia tajemnice funkcjonowania polskiego Biura Szyfrów i jego niezwykłe sukcesy odniesione podczas wojny z bolszewicką Rosją w latach 1918-1920. Systematyczne łamanie kluczy szyfrowych nieprzyjaciela umożliwiło odczytanie kilku tysięcy bolszewickich szyfrogramów i miało wielki wpływ na zwycięstwo odniesione w tej wojnie. Te sukcesy sprawiły, że polski kontrwywiad przygotowywał się przed II wojną światową także do walki kryptologicznej z Niemcami.

2 TŁO HISTORYCZNE

Można powiedzieć, że walka między kryptografami a kryptoanalitykami przed II wojną światową weszła w nową fazę. Po raz pierwszy zaczęto stosować do szyfrowania urządzenia mechaniczno-elektryczne. Przykładem takiej maszyny była Enigma. Zastosowanie tej maszyny przez Niemców zmusiło początkowo do „kapitulacji” angielskich kryptoanalityków, którzy stwierdzili, że tego szyfru nie można złamać, natomiast Francuzom udało się uzyskać cenne informacje o Enigmie, opłacając niemieckiego szpiega Hansa Szmidta o pseudonimie Asche. Sami Niemcy do końca wojny byli przekonani o nienaruszalnej sile Enigmy. Jedynie Polacy, nauczeni sukcesem wojny bolszewickiej, nie powiedzieli „nie” i przystąpili do działań w kierunku złamania tego szyfru. Wykorzystano potencjał polskich matematyków tym razem z Poznania. O sukcesie Polaków zadecydowały trzy czynniki: strach, matematyka i szpiegostwo. Gdyby nie strach przed inwazją, Polacy zapewne uważaliby, że złamanie szyfru jest niemożliwe (bo wszystkich konfiguracji ustawień w maszynie Enigma jest czterysta sekstylionów, czyli 4×10^{26}


ŚCISLE TAJNE
Tłumaczenie szyfrogramu
BARDZO PILNE!
Wysyłająca stacja: Szt.47 dyw Sygnał: Postój:
Odbiorca: Szt.XII arm. Sygnał: Postój:
Data przejęcia: 22/I 19 godz. min.
„ deszyfrowania: 26/I 19 godz. min.
Uwagi Sekcji Szyfrowej | Treść
UZUPEŁNIENIE.
Deszyfrował: Kowalewski

Rysunek 3.
Tłumaczenie odszyfrowanego meldunku [źródło: 7, s. 498]

– więcej niż liczba sekund, jakie upłynęły od początku świata zakładając, że świat istnieje od pięciu miliardów lat) – liczba kluczy, według których można kodować tekst za pomocą Enigmy, to 10 bilionów, czyli 1×10^{10}.

Bez matematyki Marian Rejewski nie byłby w stanie zanalizować szyfrogramów, a bez dokumentów, dostarczonych przez Schmidta, Polacy nie znaliby wewnętrznych połączeń w bębenkach i nie mogliby rozpocząć kryptoanalizy. Szybko zorientowano się, że Niemcy wykorzystują zmodyfikowaną wersję handlowej maszyny Enigma. Tu dopomógł przypadek. Na przełomie 1932 i 1933 roku w Biurze Szyfrów w Pyrach odtworzono wewnętrzne połączenia Enigmy i w konsekwencji zbudowano jej działającą replikę. Atak Mariana Rejewskiego na Enigmę (razem z Jerzym Różyckim i Henrykiem Zygalskim) był jednym z największych osiągnięć w historii kryptoanalizy i miał znaczący wpływ na dalsze losy wojny.

3 ZŁAMANIE ENIGMY

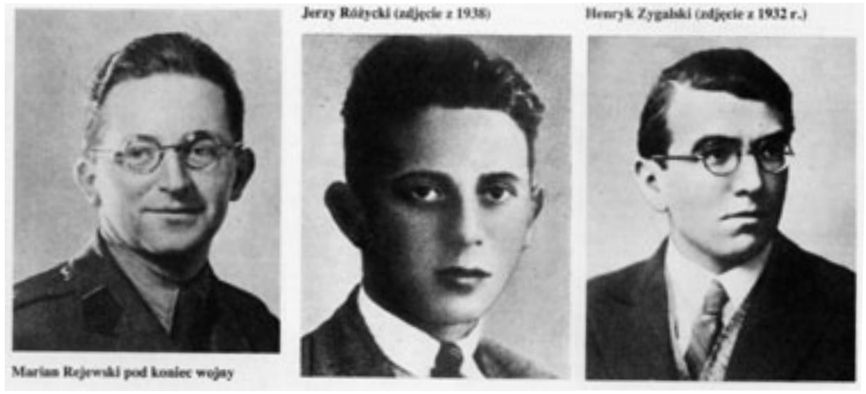
Według słów premiera Wielkiej Brytanii Winstona Churchila – gdyby nie złamanie Enigmy, dzięki której między innymi zniszczono dużą liczbę niemieckich łodzi podwodnych na Atlantyku, wojna skończyłaby się znacznie później, bo w 1948 roku zrzuceniem bomby atomowej na Niemcy.

„Natomiast Bill Clinton, jako prezydent USA powiedział w polskim sejmie 7 lipca 1994 roku: „To polscy matematycy z laboratoriów Poznania złamali sekrety kodu Enigmy – co Winston Churchill nazwał najważniejszą bronią przeciwko Hitlerowi i jego armiom. To Oni byli tymi łamaczami kodów, którzy sprawili, że stało się możliwe wielkie lądowanie Aliantów w Normandii, kiedy to amerykańskie, angielskie, francuskie, kanadyjskie, a także wolne polskie siły połączone razem wyzwoliły ten kontynent niszcząc tym samym straszną tyranię, która okryła cieniem nasze stulecie”[12].

A teraz trochę historii. Enigma z łaciny to zagadka. Ta maszyna powstała w Niemczech w 1918 roku, a jej wynalazcą był Hugo Koch, który sprzedał jej patent inżynierowi Arturowi Scherbiusowi. Planowanymi użytkownikami tej maszyny miały być korporacje, wielkie firmy chcące chronić swoją korespondencję, poczty, a także inne instytucje państwowe. Początkowo armia niemiecka nie była zainteresowana wprowadzeniem maszyn szyfrujących na miejsce powszechnego w tym czasie kodu ręcznego, jednakże plany remilitaryzacji Republiki Weimarskiej, a także odkrycie, iż służby Królestwa Brytyjskiego czytały depesze niemieckie w czasie I wojny światowej spowodowały, że dowództwo niemieckie zdecydowało się na wprowadzenie kodu maszynowego, stanowiącego gwarancję zachowania bezpieczeństwa przekazywania informacji. Ulepszona wersja Enigmy pojawiła się po raz pierwszy na wyposażeniu niemieckiej armii już w 1926 roku, najpierw w marynarce wojennej, a dwa lata później w siłach lądowych.

W 1929 roku w Instytucie Matematyki Uniwersytetu Poznańskiego na zlecenie Sztabu Głównego Wojska Polskiego zorganizowano kurs kryptologii dla studentów matematyki. Wyłonieni w trakcie tego kursu Marian Rejewski, Jerzy Różycki i Henryk Zygalski podjęli pracę nad niemieckimi szyframi w Biurze Szyfrów Sztabu Głównego Wojska Polskiego w Warszawie. W tym czasie mocarstwa zachodnie były przekonane, że złamanie algorytmu szyfrującego Enigmy jest niemożliwe i nie podejmowały jakichkolwiek prób.

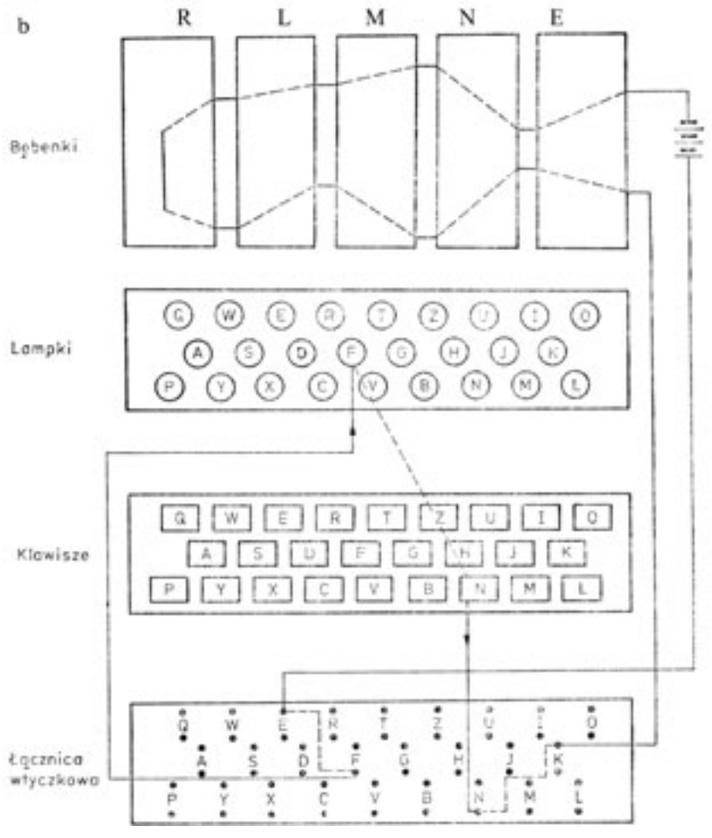
Rejewski zaczął pracować nad deszyfracją maszyny w 1932 roku. W tym czasie do szyfrowania zaczęto po raz pierwszy używać maszyn kryptograficznych. Po wieloletnich doświadczeniach walki między kryptografami i kryptoanalitykami zauważono, że szyfrowanie ręczne nie daje wystarczającego bezpieczeństwa. Bo to co człowiek zaszyfruje, człowiek potrafi złamać.



Rysunek 4.
Polscy pogromcy Enigmy [źródło: 5]

[12] Tłumaczenie autora, za: http://www.presidency.ucsb.edu/ws/index.php?pid=50451.

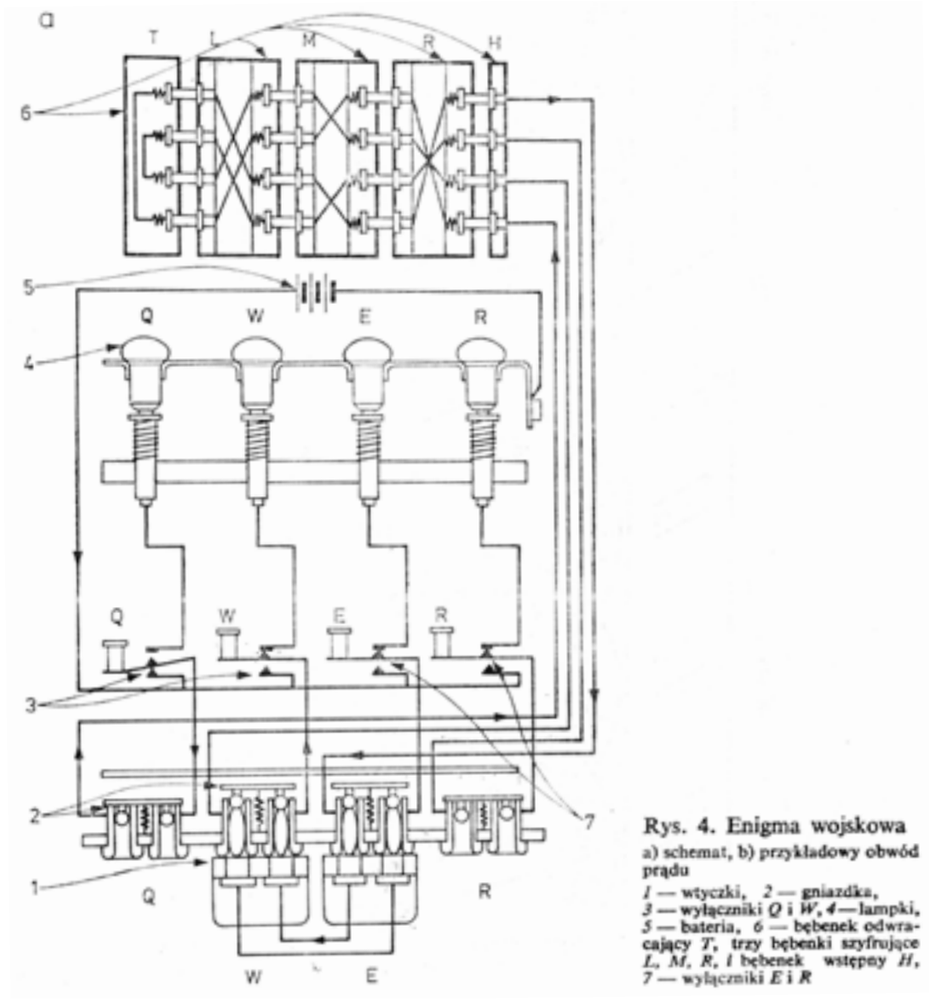
Rysunek 5.
Pomnik Mariana Rejewskiego w Bydgoszczy [źródło: Sesja naukowa w Wojskowej Akademii Technicznej poświęconej 100. rocznicy urodzin Mariana Rejewskiego, „Przeglad Historyczno-Wojskowy” 2005, nr 5/210]



Rysunek 6.
Schemat działania maszyny Enigma [źródło: 5]

Enigma jest maszyną mechaniczno-elektryczną, ma wymiary i wygląd przenośnej maszyny do pisania. Głównymi częściami składowymi maszyny są:

- klawiatura;
- zestaw lampek oświetlających;
- bębenki tworzące zasadniczą cześć maszyny tzw. mieszacz (ang. *scramble-unit*);
- łącznica wtyczkowa;
- bateria zasilająca;
- mechanizm obrotowy;
- bęben odwracający (R);
- trzy bębenki szyfrujące (L, M, N);
- bębenek wstępny (E).



Rys. 4. Enigma wojskowa
a) schemat, b) przykładowy obwód prądu
1 — wtyczki, *2* — gniazdka, *3* — wyłączniki *Q* i *W*, *4* — lampki, *5* — bateria, *6* — bębenek odwracający *T*, trzy bębenki szyfrujące *L*, *M*, *R*, *I* bębenek wstępny *H*, *7* — wyłączniki *E* i *R*

Rysunek 7.
Schemat działania Enigmy wojskowej [źródło: 5]

$\frac{1}{26}$

Naciśnięcie dowolnego klawisza na klawiaturze powoduje zamknięcie obwodu z prądem, który płynie przez:

- łącznicę wtyczkową;
- bębenek wstępny (E);
- bębenki szyfrujące (N, M, L);
- bębenek odwracający (R);
- ponownie przez trzy bębenki szyfrujące (L, M, N);
- bębenek wstępny (E) i łącznicę;
- do określonej lampki, która się zapala.

Mechanizm obrotowy Enigmy jest oparty na zasadzie licznika. Po naciśnięciu klawisza prawy bębenek zawsze obraca się o $\frac{1}{26}$ kąta pełnego, bębenki środkowy i lewy zwykle pozostają nieruchome, każdy następny bębenek obraca się po pełnym obrocie bębenka, poprzedniego. W ten sposób każda następna litera jest szyfrowana przy innych położeniach bębenka, co sprawia, że naciskając kilka razy ten sam klawisz pod rząd uzyskuje się zapalenie coraz to innych lampek. Fakt ten implikuje m.in. charakterystyczną dla szyfru maszynowego cechę polegającą na tym, że zaszyfrowane teksty dowolnej długości cechują prawie idealnie równe częstości występowania poszczególnych liter. Równie ważną cechą wynikającą wprost ze schematu połączeń elektrycznych Enigmy jest fakt, że procesy szyfrowania i deszyfrowania są identyczne.

Rozszyfrowywanie tekstu polega na wystukaniu utajnionego tekstu na Enigmę przy takich samych połączeniach łącznicy oraz kolejnych ustawieniach wirników.

Kluczem maszyny było ustawienie kolejności i pozycji początkowych trzech wymiennych bębenków, a także połączenia w łącznicy. W sumie liczba możliwych kluczy była ogromna nawet na dzisiejsze warunki, wynosiła bowiem około 10^{22}, co oznacza, że Enigma była porównywalna ze współczesnymi szyframi o długości kluczy na poziomie 75 bitów. Oprócz zmienianych codziennie kluczy Polacy nie znali budowy samej Enigmy, a zwłaszcza połączeń wewnętrznych w poszczególnych bębenkach.

Genialnym pomysłem Mariana Rejewskiego było zastosowanie do opisu budowy i działania Enigmy aparatu matematycznego, a konkretnie teorii permutacji. Proces przebiegu prądu przez łącznicę i bębenki szyfrujące zapisał on za pomocą równań permutacyjnych o nieznanych permutacjach *N*, *M*, *L*, *R*.

$$A = SPNP^{-1}MLRL^{-1}M^{-1}PN^{-1}P^{-1}S^{-1}$$
$$B = SP^{2}NP^{-2}MLRL^{-1}M^{-1}P^{2}N^{-1}P^{-2}S^{-1}$$
$$\ldots$$
$$E = SP^{5}NP^{-5}MLRL^{-1}M^{-1}P^{5}N^{-1}P^{-5}S^{-1}$$
$$F = SP^{6}NP^{-6}MLRL^{-1}M^{-1}P^{6}N^{-1}P^{-6}S^{-1}$$

Korzystając między innymi z błędów popełnianych przez niemieckich szyfrantów, wyznaczył znane iloczyny permutacji:

$$AD = SPNP^{-1}MLRL^{-1}M^{-1}PN^{-1}P^{3}NP^{-4}MLRL^{-1}M^{-1}P^{4}N^{-1}P^{-4}S^{-1}$$
$$BE = SP^{2}NP^{-2}MLRL^{-1}M^{-1}P^{2}N^{-1}P^{3}NP^{-5}MLRL^{-1}M^{-1}P^{5}N^{-1}P^{-5}S^{-1}$$
$$CF = SP^{3}NP^{-3}MLRL^{-1}M^{-1}P^{3}N^{-1}P^{3}NP^{-6}MLRL^{-1}M^{-1}P^{6}N^{-1}P^{-6}S^{-1}$$

W ten sposób skonstruował układ równań permutacyjnych z ogromną liczbą niewiadomych. W celu rozwiązania tego układu udowodnił następujące twierdzenie:

Twierdzenie. Jeżeli znane są permutacje *A* oraz *T*, to permutacja

$$B = T^{-1}AT$$

i permutacja *A* mają rozkład na iloczyn cykli rozłącznych o odpowiadających sobie cyklach o tej samej długości.

Z uwagi na wkład tego twierdzenia w złamanie Enigmy, a co za tym idzie na przebieg II wojny światowej, twierdzenie to nazwano: *Twierdzeniem, które wygrało II wojnę światową*. Na jego bazie Rejewski rozwiązał układ równań permutacyjnych. Odtworzył wszystkie nieznane permutacje, czyli zrekonstruował wewnętrzne połączenia bębenków Enigmy. Odtworzona permutacja *N*:

$$N = \begin{pmatrix} abcdefghijklmnopqrstuvwxyz \\ azfpotjyexnsiwkrhdmvclugbq \end{pmatrix}$$

Na tej podstawie zbudowano kopię Enigmy i można było codziennie znajdywać jej klucze, a w konsekwencji odszyfrowywać niemieckie meldunki.

Rejewski jest uważany za twórcę nowoczesnej kryptologii ze względu na głębokie i skuteczne zastosowanie matematyki do kryptoanalizy.

Dopuszczeni tuż przed wybuchem wojny, a dokładnie 25 lipca 1939 roku, do tej tajemnicy przekazanej przez Polaków, najpierw Francuzi, a później Anglicy umiejętnie wykorzystali i rozwinęli polskie koncepcje, tworząc w Bletchley Park pod Londynem sprawnie działający ośrodek dekryptażu. To właśnie w Bletchley Park zbudowano pierwszy komputer lampowy Colossus, który służył do łamania szyfrów maszyn Enigma i Lorenza, używanej przez niemiecki Sztab Generalny do komunikacji między Hitlerem i jego generałami.

Z okazji 100. rocznicy urodzin Mariana Rejewskiego w Wojskowej Akademii Technicznej odbyła się międzynarodowa konferencja naukowa. Obecny na niej prezes Międzynarodowego Stowarzyszenia Badań Kryptologicznych (IACR) Andy Clark podkreślił, że polscy kryptolodzy złamali szyfry niemieckie i sowieckie. W konferencji uczestniczyli współcześni najwybitniejsi kryptolodzy świata m.in.: Eli Biham, Andrew J. Clark, Peter Landrock, Nicolaus Courtois. Konferencja zakończyła się złożeniem wieńców na grobie Mariana Rejewskiego.

4 OD ENIGMY DO WSPÓŁCZESNEJ KRYPTOLOGII

W czasie II wojny światowej, słynny szpieg radziecki pochodzenia niemieckiego Richard Sorge, wnuk sekretarza Karola Marksa, jako attache prasowy ambasady niemieckiej w Tokio od 1933 do 1941 roku kierował siatką agentów o nazwie Czerwona Orkiestra. To on przekazywał zdobyte bezcenne informacje radzieckiemu wywiadowi wojskowemu, między innymi o pakcie berlińskim, ataku na Pearl Harbor i o tym, że Japonia nie zaatakuje Związku Radzieckiego, podał również dokładną datę rozpoczęcia planu Barbarossa. Moskwa podziękowała za te informację, ale nie podjęła żadnych działań. Co ciekawe, Sorge w swoich przekazach wykorzystywał stary szyfr podstawieniowy, gdzie właściwe szyfrowanie oparte było na ciągach liczb losowych, którymi były dane z niemieckich roczników statystycznych. Szyfr ten nie został złamany przez żadne służby specjalne.

Jednym z ważniejszych osiągnięć kryptologicznych, które zmieniło bieg wydarzeń II wojny światowej na Pacyfiku, było rozszyfrowanie japońskiego szyfru maszynowego Purple, czyli purpurowego, przez wojskowego kryptologa Williama Friedmana. Amerykański wywiad mógł gromadzić informacje na temat ruchu japońskich okrętów i samolotów. Amerykanie dzięki temu wygrali bitwę o Midway. Na skutek tych informacji, udało się także

zestrzelić samolot admirała Yamamoto, znakomitego dowódcy japońskiego, który jako jeden z niewielu wśród najwyższych rangą dowódców cesarskiej floty, posiadał ogromną wiedzę na temat planowania strategicznego. Mówiono o nim, że był „mózgiem" armii japońskiej i postrachem armii amerykańskiej. Podobno prezydent USA Franklin D. Roosvelt wiedział o wcześniejszym ataku na Pearl Harbor, ale nie poinformował o tym społeczeństwa, gdyż chciał, aby Senat USA wyraził zgodę na przystąpienie Stanów Zjednoczonych do wojny.

W latach 1948-1955 w ramach projektu Venona, przeprowadzonego przez Amerykanów, udało się im rozszyfrować część depesz radzieckich. Rosjanie zrobili błąd używając czasami jako klucza do różnych depesz tego samego ciągu losowego.

20 czerwca 1953 roku w nowojorskim więzieniu Sing-Sing miała miejsce egzekucja małżeństwa Ethel i Juliusa Rosenbergów. Dwa lata wcześniej skazano ich za zdradę tajemnic związanych z budową bomby atomowej. Szpiegowali na rzecz ZSRR, zdemaskowano ich dzięki „wypadkowi przy pracy". Rosjanie wykorzystywali kilkakrotnie ten sam klucz. Również oficer odpowiedzialny za całą akcję przypłacił tę pomyłkę życiem. Służby specjalne byłego ZSRR posługiwały się przypadkowo skonstruowanymi kluczami, a mówiąc dokładniej – ciągami liczb losowych.

Kiedy w czerwcu 1957 roku w jednym z nowojorskich hoteli został ujęty radziecki szpieg Rudolf Abel, agenci FBI znaleźli blok, którego strony wielkości znaczków pocztowych były pokryte długimi ciągami liczb. Były to klucze numeryczne.

W latach siedemdziesiątych również innym szpiegom sowieckim nie udało się zniszczyć posiadanych kluczy przed ich schwytaniem. Amerykanie szybko odkryli, że nie są to w żadnym wypadku prawdziwie liczby losowe. Można sądzić, że właściwości rosyjskich kluczy z liczb przypadkowych wpłynęły na historię świata w sierpniu 1991 roku. Wówczas to podczas puczu przeciwko Michaiłowi Gorbaczowowi dwaj spiskowcy – szef KGB Władimir Kriuczkow i minister obrony narodowej Dymitrij Jazow wymieniali zaszyfrowane informacje. Dzięki regularnościom w kluczu, Amerykanie zdołali odczytać te wiadomości. Prezydent George Bush przekazał je następnie Borysowi Jelcynowi.

Można powiedzieć, że I wojna światowa była wojną chemików – ponieważ wówczas po raz pierwszy zastosowano chlor i gaz musztardowy jako gazy bojowe. Natomiast II wojna światowa była wojną fizyków, którzy skonstruowali bombę atomową. Zapewne III wojna światowa będzie wojną matematyków, bo to właśnie oni będą kontrolować następną ważną broń – informację. To matematycy stworzyli szyfry stosowane obecnie do zabezpieczania informacji wojskowych i oczywiście również oni przewodzą w próbach ich złamania.

LITERATURA

1. Bertrand G., *Enigma ou la plus grande énigme de la guerre 1939–1945* (*Enigma: the Greatest Enigma of the War of 1939-1945*), Plon, Paris 1973
2. Gannon J., *Stealing Secrets, Telling Lies: How Spies and Codebreakers Helped Shape the Twentieth Century*, Brassey's, Washington D.C. 2001
3. Kahn D., *Łamacze kodów*, WNT, Warszawa 2004
4. Kippenhahn R., *Tajemne przekazy, szyfry, Enigma i karty chipowe*, Prószyński i S-ka, Warszawa 2000
5. Kozaczuk W., *Enigma: How the German Machine Cipher Was Broken, and How It Was Read by the Allies in World War II*, edited and translated by Christopher Kasparek, University Publications of America, Frederick MD, 1984
6. Levy S., *Rewolucja w kryptografii*, WNT, Warszawa 2002
7. Nowik G., *Zanim złamano Enigmę. Polski Radiowy wywiad podczas wojny z bolszewicką Rosją 1918-1920*, Rytm, Warszawa 2005
8. Singh S., *Księga szyfrów. Od starożytnego Egiptu do kryptografii kwantowej*, Albatros, Warszawa 2001

Od abaków do maszyny ENIAC i Internetu

Piotr Sienkiewicz

Warszawska Wyższa Szkoła Informatyki

sienkiewicz@wwsi.edu.pl, p.sienkiewicz@aon.edu.pl

Streszczenie

Wykład stanowi wprowadzenie do historii informatyki i technologii informacyjno-komunikacyjnych (ICT). Przedstawione są punkty zwrotne w rozwoju ICT z określeniem ich roli w postępie cywilizacyjnym. Ponadto warto pokazać, jak główne ścieżki rozwojowe w różnych obszarach ludzkiej działalności intelektualnej i praktycznej „przecinały się" i co z tego wynikało dla rozwoju informatyki oraz ICT w rozmaitych dziedzinach ludzkiej aktywności.

Umownie wyróżniono: etap prehistoryczny – od abaków do maszyny ENIAC, etap historyczny – od maszyny ENIAC do Internetu i etap współczesny – po Internecie. W pierwszym etapie zwraca się uwagę na pierwsze próby liczenia przy pomocy prostych urządzeń (np. abaków), powstanie liczydeł i mechanicznych arytmometrów, ale również na pomysły i koncepcje, o tak różnym stopniu zaawansowania technologicznego, jak np. projekty Schickarda i Pascala, Leibniza i Babbage'a, wreszcie Holleritha, Zusego i twórców maszyny ENIAC. Nie można ponadto pominąć roli postaci tej wielkości, jak Boole, Turing i von Neumann. W drugim etapie uwaga jest skupiona przede wszystkim na zmianach generacyjnych oraz wpływie technologii elektronicznych na architekturę i efektywność systemów komputerowych. Także scharakteryzowany jest rozwój oprogramowania – powstanie i rozwój systemów operacyjnych oraz generacje języków programowania. Należy zwrócić uwagę na powstanie masowych zastosowań w związku z użyciem PC. Etap historii informatyki wieńczy powstanie Internetu, co jest rezultatem zbieżności i konwergencji trzech megatrendów: technologicznego (elektronicznego), informatycznego i telekomunikacyjnego. Stanowi to punkt wyjścia dla refleksji na temat tendencji rozwojowych, możliwych i prawdopodobnych w perspektywie najbliższej dekady.

Wykład jest bogato ilustrowany, często unikatowymi zdjęciami i fragmentami filmów poświęconych historii komputerów.

Spis treści

1 WPROWADZENIE

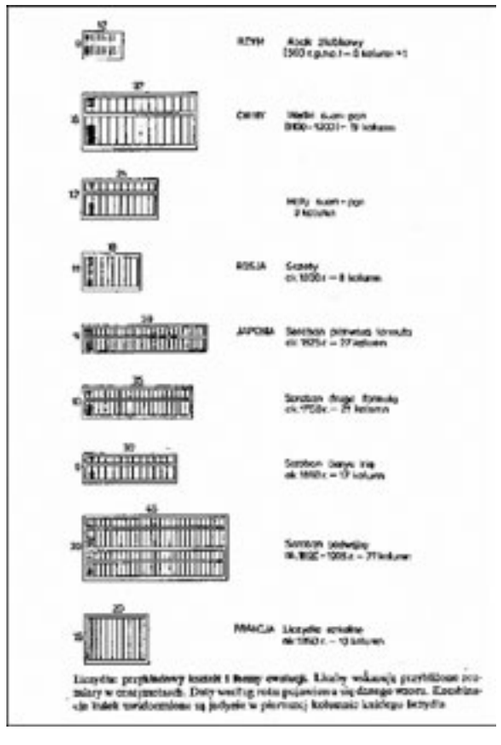
Historia cywilizacji pisana dziejami narzędzi tworzonych przez człowieka w celu wspomagania siły mięśni, zdolności przemieszczania się w przestrzeni i czasie, a także możliwości intelektu, jest równie fascynująca jak dzieje przemian społecznych, politycznych i gospodarczych, czy dzieje wojen. Trudno rozdzielać poszczególne zjawiska, gdyż wspólnie tworzą zmieniający się obraz cywilizacji.

W pięknej książce Roberta Ligonniere'a [2] poświęconej historii komputerów czytamy: „Komputer, symbol XX w., wywodzi się mimo wszystko z dalekiej, a mało znanej przeszłości. Od antycznych abaków po pałeczki obliczeniowe, od maszyn Leibniza lub Pascala po mechanizmy Babbage'a i Holleritha, od logiki binarnej YiKing po koncepcje Boole'a przeplatają się metamorfozy wielkiej chimerycznej idei i natchnione poszukiwania upartych wynalazców".

Po rewolucji agrarnej późnego neolitu i rewolucji przemysłowej ostatnich dwóch stuleci, ludzkość stanęła u progu kolejnego przełomu – rewolucji informacyjnej. Chodzi o zjawisko społeczne, które dla Alvina Toffflera jest *Trzecią Falą*, dla innych zaś społeczeństwem informacyjnym. Czy można wyobrazić sobie powstanie i rozwój społeczeństwa informacyjnego bez wpływu postępu naukowo-technicznego w takich dziedzinach, jak: fizyka ciała stałego i mikroelektronika, telekomunikacja i informatyka?

2 PREHISTORIA

U podstaw społeczeństwa informacyjnego należy widzieć jeden z najbardziej fascynujących wynalazków wszechczasów: maszynę do przetwarzania informacji – komputer. Pomysł – praktycznie zrealizowany ponad



Rysunek 1.
Liczydła: przykładowy kształt i formy ewolucji [źródło: 3, s. 13]

pół wieku temu – dojrzewał powoli, od starożytności począwszy. Przez stulecia kumulowała się wiedza, jedne pomysły wypierały inne. Najpierw musiały powstać cyfry, aby następnie powstawały mechanizmy zdolne do operowania nimi, wykonywania coraz bardziej złożonych obliczeń. Do nich należą: abaki i liczydła, które przez wieki były jedynymi urządzeniami ułatwiającymi czynności intelektualne, jakimi niewątpliwie są obliczenia. Abak zrodził się gdzieś między Mezopotamią a Indiami.

Z kolei liczydła – powstały przypuszczalnie na Bliskim Wschodzie, a od V wieku p.n.e. zadomowiły się w Rzymie. Różne ich odmiany znajdujemy w różnych krajach: w Chinach – suan-pan, Japonii – soroban, Rosji – sczoty, aż wreszcie – od XVII wieku pałeczki Nepera wykorzystane w maszynie Schickarda.

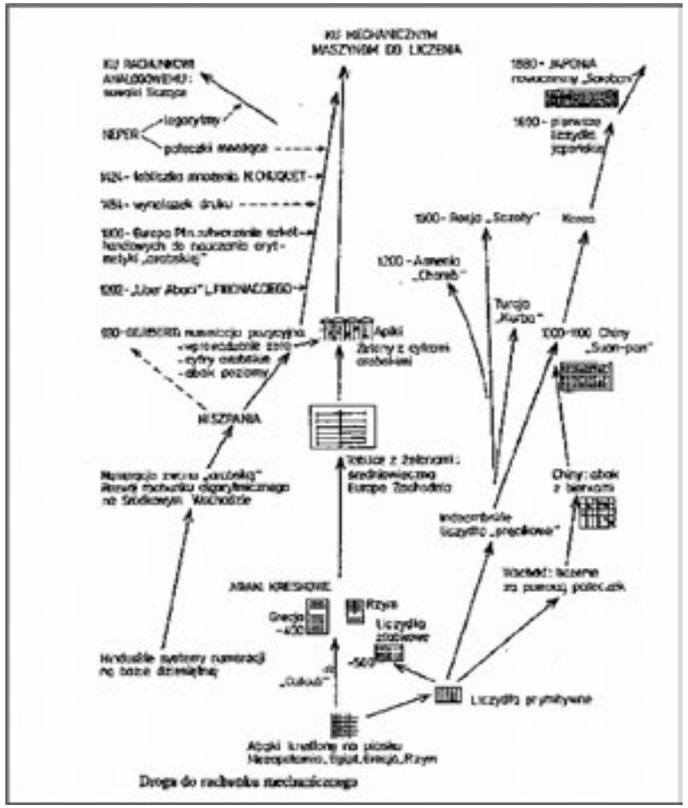
Abakiem była każda plansza obliczeniowa, liniowana pionowo lub poziomo, której zarys ułatwiał przemieszczanie kamyków, patyków, żetonów lub innych znaków, natomiast **liczydło** było już kompletnym, samodzielnym, przenośnym przyrządem, złożonym z pręcików lub wyżłobień, umożliwiających wyrażanie różnych rzędów wielkości oraz z określonej liczby znaków, które użytkownik przemieszczał, gdy istniała potrzeba obliczeń. Warto prześledzić chociaż wybrane próby ludzkiego radzenia sobie z obliczeniami przy pomocy historycznych narzędzi. Ludy żyjące na Bliskim Wschodzie w II tysiącleciu p.n.e. przy obliczeniach posługiwały się stożkami, kulkami, pałeczkami i innymi glinianymi przedmiotami (calculi), przedstawiającymi jednostki rozmaitych rzędów wielkości w używanych przez nich systemach liczenia. Sumerowie w okresie 2700-2300 p.n.e. przestali używać do obliczeń dawnych calculi i stworzyli abakus – tablicę, na której nakreślone uprzednio kolumny rozgraniczają kolejne rzędy wielkości systemu sześćdziesiątkowego, a działania arytmetyczne wykonywali za pomocą przemyślnego przekładania kulek lub pałeczek.

W II wieku p.n.e. Plutarch wspomina o posługiwaniu się przez Greków i Persów abakusem pyłowym, równolegle z abakusem z żetonami. Z tego okresu pochodzą pierwsze znane świadectwa stosowania chińskiego abakusa i „rachunku za pomocą patyczków" (suan zi). W I wieku p.n.e Horacy wspominał o posługiwaniu się przez Rzymian abakusem woskowym, obok abakusa z żetonami, jako prawdziwym „przenośnym kalkulatorem", który można była przewiesić przez ramię. W tym czasie korzystano z rzymskiego abakusa, przypominającego liczydła, do dziś używanego na Dalekim Wschodzie.

3 HISTORIA DO XX WIEKU

W wiekach X-XII europejscy rachmistrze wykonywali działania arytmetyczne na abakusie z kolumnami pochodzenia rzymskiego, udoskonalonym przez Gerberta z Aurillac. Na okres XII-XVI w. przypadły spory między **abacystami** (zwolennikami rachunku za pomocą żetonów na abakusie) a **algorystami** (wielbicielami rachunku pisemnego za pomocą zera i dziewięciu cyfr pochodzenia indyjskiego). Z XIII wieku pochodzą pierwsze znane świadectwa posługiwania się chińskimi liczydłami (suan pan) w postaci używanej do dziś. Podobnych liczydeł używano do niedawna w Rosji (sczoty), w Iranie i Afganistanie (czoreb), w Armenii i Turcji (kulba). Jednym z ważniejszych wydarzeń było sprowadzenie z Hiszpanii abakusa przez wspomnianego Gerberta z Aurillac, nauczyciela szkoły katedralnej w Reims, późniejszego (999 r.) papieża Sylwestra II. Ten fascynujący przyrząd był drewnianą tablicą podzieloną na 30 kolumn zawierających poziome pręty, na których przesuwało się koraliki. Umożliwiał dodawanie, odejmowanie, a nawet mnożenie, dzięki zastosowaniu dziesiętnego systemu jednostek, dziesiątek, setek itd., co nie było jednak wcale łatwe. Z korespondencji między papieżem i cesarzem, z czasów gdy Gerbert przybył po raz pierwszy do Rzymu, można się domyślić, że biegłość w posługiwaniu się abakusem ceniono wysoko. Gdy papież napisał do cesarza „Mam tu dobrego matematyka", ten odpisał mu: „Nie wypuszczaj go z miasta!" [3, s. 14].

W historii wynalazku szczególne znaczenie ma XVI stulecie, kiedy to mają miejsce dwa niezależne od siebie wynalazki. Dla Francuzów wynalazcą pierwszej maszyny liczącej jest Blaise Pascal, który mając zaledwie 18 lat obmyślił maszynę arytmetyczną, zbudowaną następnie w blisko 50 różnych egzemplarzach i różnych wariantach (np. maszyny zwykłe sześcio- lub ośmiocyfrowe, maszyny typu „monetarnego" i maszyny dla geometrów). Nie wszystkim wiadomo, zaś Francuzi niechętnie przyjmują to do wiadomości, że wielkiego filozofa, autora myśli o człowieku jako „trzcinie myślącej", uprzedził w zmaganiu o realizację idei „maszyny myślącej"



Rysunek 2.
Droga do rachunku mechanicznego [źródło: 3, s. 15]

(oczywiście, w sensie – liczącej) Niemiec – Wilhelm Schickard. W 1623 roku pisał on do Johannesa Keplera: „...mechanicznie spróbowałem zrobić to, co ty wykonujesz ręcznie, zbudowałem maszynę, która natychmiast, automatycznie przelicza zadane liczby, dodaje, odejmuje, mnoży i dzieli... Skakać będziesz pewnie z radości, gdy zobaczysz, jak przenosi ona liczbę dziesiątek i setek lub też ujmuje ją przy odejmowaniu" [3, s. 25].

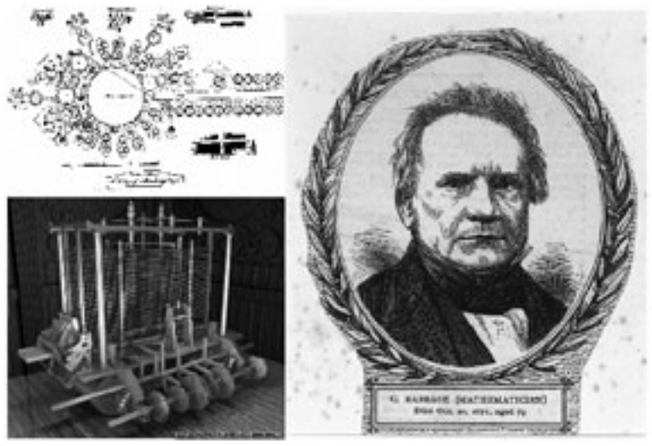
Rysunek. 3
Maszyna Wilhelma Schickarda i maszyna Blaise'a Pascala [źródło: 7, s. 10]

Z Niemiec i Francji idea maszyn liczących wiedzie do Anglii, gdzie Samuel Morland, po latach niebezpiecznych gier politycznych (trzeba pamiętać, że są to czasy Olivera Cromwella i Karola Stuarta, kiedy to głowę stracić można szczególnie łatwo), skonstruował kalkulator... kieszonkowy. Ten pionier miniaturyzacji maszyn liczących nie zawsze, jak to z pionierami bywa, spotykał się ze zrozumieniem współczesnych, bo choć Samuel Pepys zapisał w swym dzienniku: „Bardzo ładne, ale mało użyteczne", to już taki Robert Hooke był bardziej jednoznaczny w komentarzu: „Widziałem maszynkę arytmetyczną Sir Samuela Morlanda. Idiotyzm" [2].

Na wiek XVII przypada również żywot jednego z najprzedniejszych uczonych wszystkich czasów – Gottfrieda Leibniza. Zasługi Leibniza dla rozwoju filozofii (monady), matematyzacji logiki oraz rachunku różniczkowego i całkowego są powszechnie znane. Mniej natomiast znane są prace nad konstrukcją maszyn liczących, w związku z którymi w 1671 roku Leibniz tak pisał: „Nie godzi się wybitnym ludziom trwonić czas na niewolniczą pracę, na obliczenia, które z zastosowaniem maszyn mógłby wykonać ktokolwiek" [2]. Dla realizacji takiej maszyny poświęcił część majątku osobistego, a sława o nim dotarła nawet do Chin. Schyłek życia wielkiego uczonego był smutny, gdyż bardzo samotny, a pewien kronikarz hanowerski pisał w związku ze śmiercią Leibniza: „pochowany został niby złodziej, nie zaś jak ktoś, kto był chlubą swej epoki".

Rysunek 4.
Gotfried Leibniz

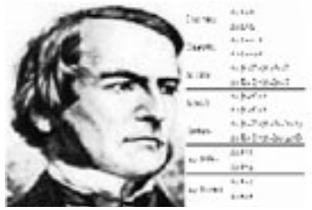
W 1822 roku Charles Babbage przesłał prezesowi Akademii Nauk memoriał zawierający opis projektu maszyny zdolnej do wykorzystania wszelkiego rodzaju tablic matematycznych przy użyciu li tylko metody różnic oraz propozycję sfinansowania budowy jego **maszyny różnicowej**... ze środków państwowych. I te środki na projekt Babbage'a zostały przyznane. Gdyby ta maszyna różnicowa została zrealizowana, byłaby konstrukcją o wysokości 3 m, szerokości ok. 1,6 m i głębokości ponad 1 m. Gdyby genialny konstruktor nie poniósł porażki, bowiem, jak się dziś sądzi, projekt Babbage'a przerósł możliwości technologiczne epoki. A o zwyczajnym pechu prześladującym konstruktora i krążących plotkach (o przywłaszczeniu sobie środków społecznych, rzecz jasna) nawet nie warto wspominać. W każdym razie dziś Anglicy nie mają raczej wątpliwości, że faktycznym wynalazcą maszyny cyfrowej był Charles Babbage.



Rysunek 5.
Charles Babbage i projekt maszyny różnicowej

W 1833 roku na pewnym przyjęciu Babbage'a poznała pewna osiemnastolatka, z której to późniejszego artykułu poznano opis działania **maszyny analitycznej** i jej programowania. Była nią Ada Lovelace, córka wielkiego poety Lorda Byrona, którego wszak nie miała nigdy poznać. Uważa się, że Ada – młodsza o 23 lata od Babbage'a – łącząca młodość, pasję, inteligencję i sobie właściwy tylko urok, stała się dla niego czymś w rodzaju podpory moralnej. Gorzka była starość Babbage'a, na co nie bez wpływu była przedwczesna i w opłakanej sytuacji materialnej śmierć Ady (1852).

Rysunek 6.
Ada Lovelace i fragment jej artykułu



Rysunek 7.
George Boole i fragment publikacji

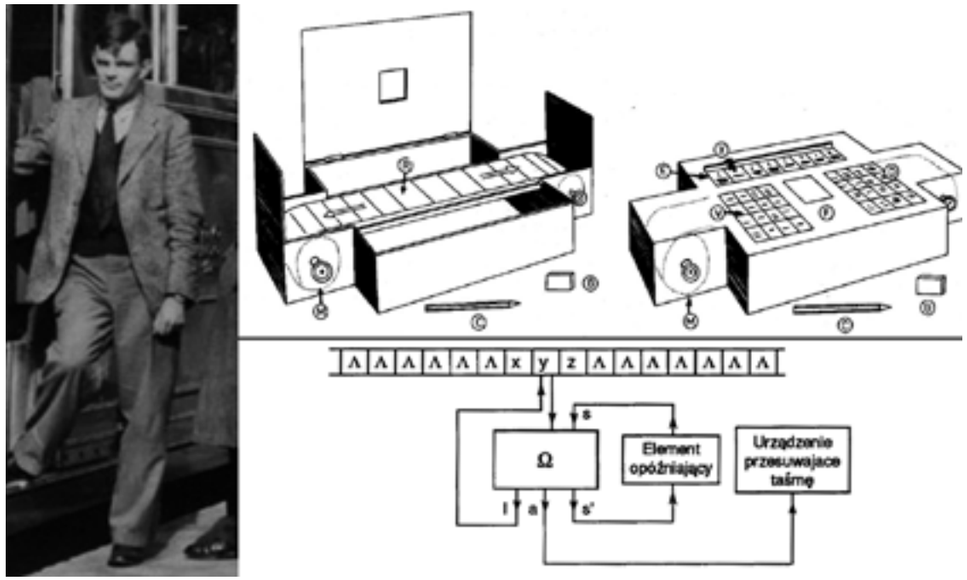
Dziś w Muzeum Nauk w Londynie jest przechowywany prototyp maszyny analitycznej, zaś jeden z bardziej znanych języków programowania nosi imię Ady. Ale w wieku XIX pracują także matematycy, o których historia komputerów nie może milczeć: Augustus De Morgan i George Boole, którym zawdzięczamy podstawy logiki maszyn liczących.

Blisko sto lat czekały prace Boole'a na temat logiki dwuwartościowej (algebra Boole'a), by stać się teoretycznym narzędziem informatyki i telekomunikacji. Prace Boole'a zaowocowały powstaniem rachunku zdań i rachunku predykatów, a na ich gruncie – języków programowania. W ramach logiki matematycznej powstała teoria dowodu i teoria obliczalności. W 1936 roku Claude E. Shannon jako pierwszy zastosował algebrę Boole'a do analizy i syntezy układów logicznych.

4 HISTORIA DO POWSTANIA MASZYNY ENIAC

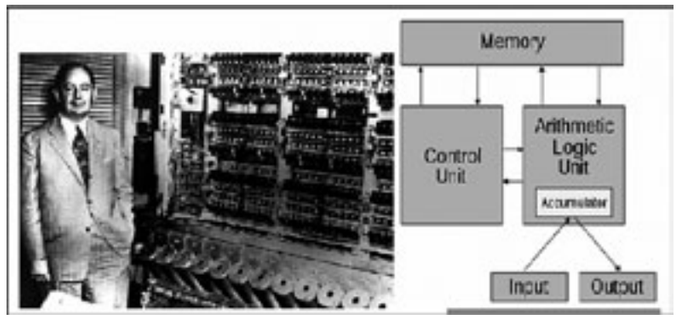
W 1890 roku prasa amerykańska ogłosiła rozpoczęcie nowej ery: „Po raz pierwszy w historii świata spis wielkiego narodu dokonany został za pomocą elektryczności". Stało się to możliwe dzięki **systemom tabulacyjnym** Hermana Holleritha, wykorzystującym m.in. karty perforowane. Należałoby jeszcze choć wspomnieć o patentach Norwega Frederica Bulla i maszynie Williama C. Burroughsa, telegrafie elektrycznym Samuela Morse'a, czy wszechstronnego wynalazcę, pioniera elektromagnetycznego liczenia Hiszpana Leonardo Torresy Quevedo. Tego ostatniego uważa się obecnie za pierwszego teoretyka „totalnej" automatyzacji („Esej o automatyce", 1914).

O latach trzydziestych XX wieku mówiono jako o „czasach wielkich teoretyków", mając na uwadze przede wszystkim osiągnięcia całej plejady fizyków i matematyków. Jednym z nich był Anglik Alan Turing, który w latach 1935-1938 wymyślił „maszynę logiczno-matematyczną, czysto abstrakcyjną i teoretycznie uniwersalną, przy której po raz pierwszy pojawił się pomysł automatu algorytmicznego". Amerykanin Claude E. Shannon przedstawił w 1937 roku błyskotliwą syntezę technologii elektromechanicznej, algebry Boole'a i systemu binarnego (10 lat później zaprezentował fundamentalną, matematyczną teorię komunikacji). Matematyk, Austriak Kurt Gödel w 1931 roku wykazał, że nie wszystko da się udowodnić, a po nim Turing, że nie wszystko da się policzyć. W tym okresie w Princeton pracował Albert Einstein, ale również Alonzo Church, Kurt Gödel i Alan Turing, a przede wszystkim inny geniusz John von Neumann (zajmował się on teorią funkcji rzeczywistych, logiką matematyczną, teorią miary, geometrią i topologią, rachunkiem operatorów i probabi-



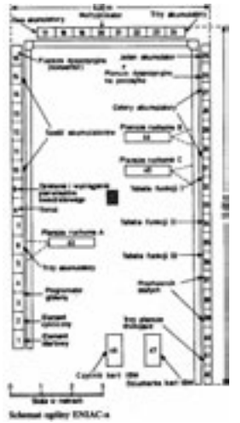
Rysunek 8.
Alan Turing i jego maszyna [źródło: 3, s. 208]

listyką, był jednym z twórców teorii gier i zapoczątkował prace nad matematycznymi modelami gospodarki, a poza tym wniósł wkład do powstania tak praktycznych wynalazków, jak komputer i... bomba atomowa). Do niedawna, i niemal powszechnie, za „ojca komputerów" uważano właśnie J. von Neumanna. Jego bowiem koncepcja maszyny cyfrowej, opartej na binarnym układzie arytmetycznym, rozdziale programu i danych w pamięci itp., legła u podstaw prac prowadzonych w ramach wojskowego Projektu X, a zmierzających do skonstruowania kalkulatora elektronicznego, mającego przyspieszyć obliczenia balistyczne, z atomistyki itp. Uwieńczenie tych prac nastąpiło w dniu św. Walentego, w 1946 roku, gdy gen. Gladeon Barnes uruchomił pierwszą maszynę cyfrową.



Rysunek 9.
John von Neumann i koncepcja komputera [źródło: 7]

15 lutego 1946 roku na Uniwersytecie Pensylwanii w Filadelfii uruchomiono pierwszą elektroniczną maszynę cyfrową nazwaną przez jej konstruktorów: Johna H. Mauchly'ego i J. Prespera Eckerta – ENIAC (ang. *Electronic Numerical Integrator and Computer*). Zainstalowany na parterze jednego z budynków Szkoły Moore'a, ENIAC ważył 30 ton, zajmował 72 m^2 powierzchni (miał kształt litery U w prostokącie 12x6 m), a pobór przez niego mocy wynosił 140 kWh, składał się m.in. z 18 000 lamp elektronowych szesnastu rodzajów, 6000 komutatorów, 10 000 kondensatorów, 50 000 oporników, 1500 przekaźników. Ulegał częstym uszkodzeniom (średnio każdą lampę należało wymieniać co 2 dni), ale dobrze służył użytkownikom, aż do 2 października 1955 roku, kiedy to o godz. 23:45 został wycofany z eksploatacji, a rząd postanowił go sprzedać na złom.

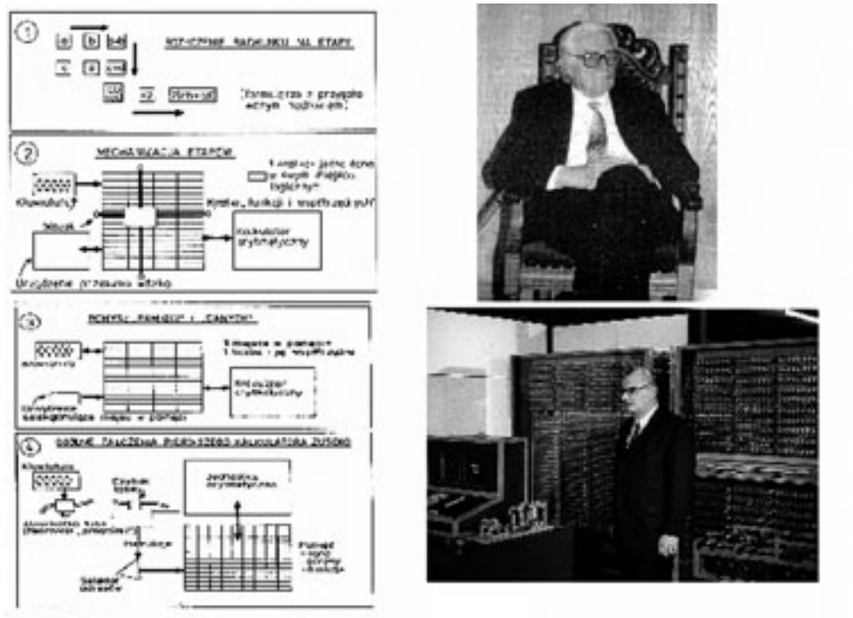


Rysunek 10.
ENIAC i jego twórcy [źródło: 7]

Gdy latem 1946 roku zniesiono tajemnicę wojskową, jaka otaczała maszynę ENIAC, stało się jasne, że był to punkt przełomowy w historii komputerów. Lato owego roku, jak pisze Ligonniere [2], zamyka bardzo długi, bogaty i zróżnicowany okres dojrzewania technologicznego i intelektualnego, otwiera przyszłość, której znaczenia i zasięgu nikt jeszcze nie podejrzewał, rewolucję, której nazwa brzmi – eksplozja informatyki.

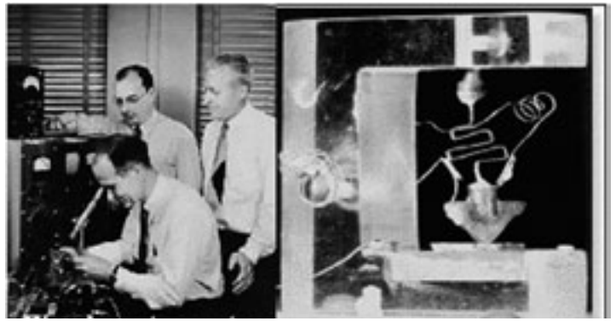
Wszystko zaczęło się od maszyny ENIAC, co do tego nie było wątpliwości, zwłaszcza podczas 50-lecia jego jubileuszu, gdy wiceprezydent Al Gore uruchomił na chwilę jego replikę. Z tej okazji zademonstrowano mikroprocesor będący kopią maszyny ENIAC o rozmiarach: 7,44 mm x 5,29 mm, liczący 174 569 tranzystorów.

Wiosną 1993 roku w Uniwersytecie Szczecińskim nadano tytuł Profesora Honorowego Instytutu Cybernetyki Ekonomicznej i Informatyki gościowi z Niemiec – Konradowi Zuse. Zuse urodził się w 1910 roku w Berlinie, a po studiach na tamtejszej politechnice, poświęcił się konstruowaniu maszyn liczących. W 1938 roku skonstruował pierwszą mechaniczną maszynę liczącą Z1, która – co należy podkreślić – pracowała w oparciu o binarny system liczenia, zmienny przecinek i sterowana była przy pomocy taśmy dziurkowanej, z której dane mogły być wczytane do 16 komórek pamięci o długości 24 bitów każda. Zbudowana trzy lata później przy wykorzystaniu techniki mechaniczno-elektrycznej, kolejna maszyna licząca Z3 była pierwszym zadowalająco działającym komputerem na świecie. Był on wyposażony w: 6000 przekaźników w układzie liczącym, 1800 przekaźników w pamięci, binarny system liczenia, zmienny przecinek, pojemność pamięci 64 słowa o długości 22 bitów, podstawowe operacje arytmetyczne, wprowadzanie danych z klawiatury w postaci 4 liczb, dziesiętnych z możliwością ustawienia przecinka w obszarze 20 miejsc dziesiętnych, wprowadzanie danych liczbowych poprzez lampy z wyświetleniem przecinka, sterowanie przez sekwencyjny program na taśmie perforowanej. Dziś replikę Z3 można oglądać w Deutschen Museum w Monachium. Konrad Zuse – twórca komputera, jeden z pionierów informatyki – zmarł w grudniu 1995 roku.



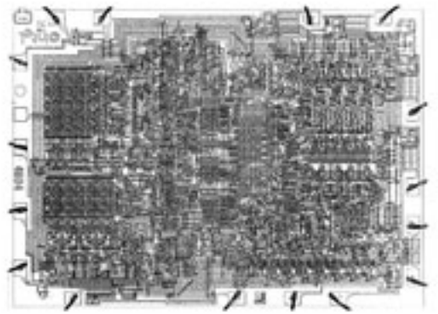
Rysunek 11.
Konrad Zuse i jego koncepcja komputera [źródło: 3, s. 244]

Zanim maszynę ENIAC przeznaczono na złom miały miejsce inne ważne wydarzenia: w 1951 roku po raz pierwszy zastosowano maszyny liczące w dziedzinie innej niż obliczenia naukowo-techniczne, a mianowicie w przetwarzaniu danych, najpierw w logistyce wojskowej, potem dla potrzeb biznesu. Faktycznie początki przetwarzania danych należy łączyć z zastosowaniami systemu tabulacyjnego Holleritha. Komputery z uniwersytetów wkroczyły do banków i wojskowych systemów dowodzenia. W 1946 roku Delmar S. Harder z fabryki Forda wprowadził pojęcie „automatyzacja", a w 1950 roku pojawił się inny termin – „automatyzacja pracy biurowej", zaś dwa lata później John Diebold publikuje pracę pt. *Automation and the Advent of the Automated Factory* (1952). W 1955 roku znana amerykańska firma Texas Instruments tworzy pierwsze „centrum przetwarzania danych". J. Diebold stworzył serię wydawniczą, której kolejne pozycje tłumaczone na język polski stanowiły w latach 60. i 70. ubiegłego wieku ważne źródło wiedzy o systemach przetwarzania danych.

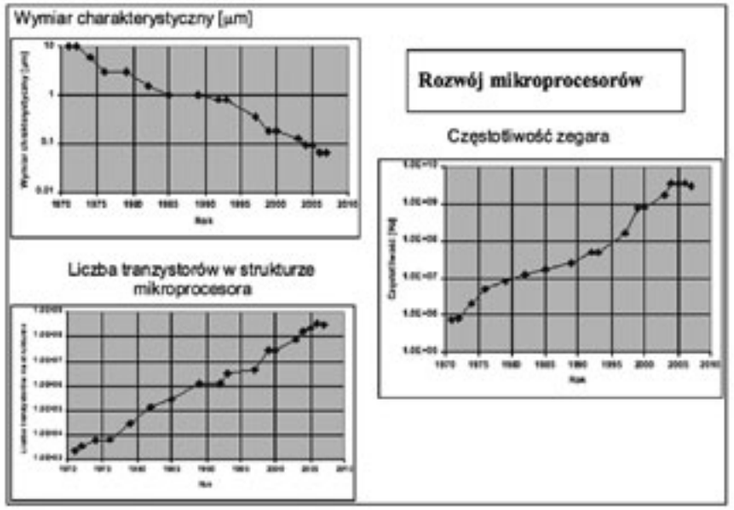


Rysunek 12.
Wynalazcy tranzystora i ich dzieło

Układ scalony (1958)



Mikroprocesor (1971)



Rysunek 13.
Rozwój technologii elektronicznej od układu scalonego do mikroprocesora [źródło: 7]

Od wynalezienia w 1906 roku przez Lee de Foresta wzmacniającej lampy elektronowej – **triody** – do zastosowania jej w pierwszym komputerze upłynęło 40 lat (ale po 10 latach została wykorzystana do skonstruowania układu przerzutnika, który stał się podstawowym układem cyfrowym). W 1947 roku trzech amerykańskich uczonych: William Shockley, John Bardeen i Walter Brattain dokonało odkrycia nowego półprzewodnikowego elementu elektronicznego – **tranzystora bipolarnego**. On to, po upływie kolejnych 10 lat, stał się podstawowym elementem układów komputerowych (II generacja komputerów). W 1958 roku Jack Kilby i Robert Noyce w laboratoriach firmy Texas Instruments skonstruował pierwszy układ scalony, umieszczając na jednym krysztale półprzewodnika więcej niż jeden ze współpracujących z sobą elementów. Wytwarzany od 1961 roku na skalę przemysłową układ scalony był przerzutnikiem

1957 FORTRAN, 1958 ALGOL,
1960 LISP, 1960 COBOL, 1962 APL ,
1962 SIMULA, 1964 BASIC ,
1964 PL/I , 1970 Prolog , 1972 C ,
1972 Smalltalk, 1975 Pascal ,
1975 Scheme , 1979 MODULA-2,
1980 dBASE II, 1983 Smalltalk-80,
1983 Ada , 1984 Standard ML ,
1986 C++, 1988 Mathematica,
1989 HTML, 1990 Haskell,
1995 Delphi, 1995 Java,
1997 PHP 2.0, 2000 C#

Ewolucja języków programowania

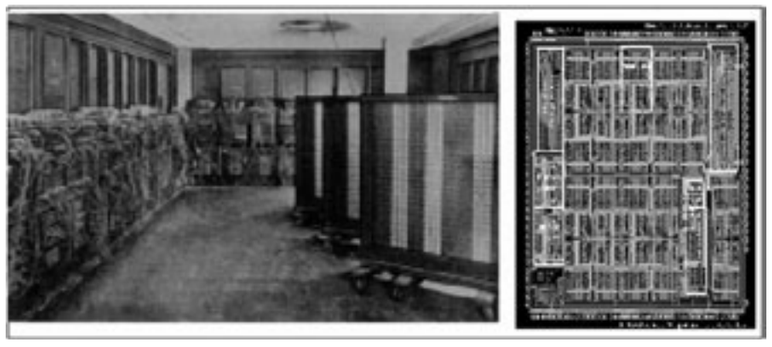
i składał się z czterech tranzystorów bipolarnych i dwóch rezystorów. Rozwój technologiczny przynosił stały wzrost skali integracji układów integracji: od małej (SSI) do bardzo wielkiej (VHLSI). I znów po 10 latach, bo u schyłku lat 60. XX wieku układy scalone zastosowano w konstrukcji układów komputerowych (komputery III generacji). A potem stosowanie układów coraz większej skali integracji **przynosiło** komputery, nie tylko mniejsze i lżejsze, ale przede wszystkim szybsze, tańsze i bardziej niezawodne. Bez tych przemian technologicznych z pewnością nie dokonałyby się zmiany organizacji procesów przetwarzania danych w systemach komputerowych.

Pierwszym **językiem programowania** był **Plankalküll** Konrada Zuse. Obecnie istnieje ponad 2500 różnych języków programowania. Jeszcze na początku lat 60. komputer mógł być wykorzystywany przez tylko jednego użytkownika z tylko jednym programem napisanym, jeśli nie w języku wewnętrznym maszyny, to w tzw. **assemblerze**. Pod koniec szóstej dekady XX wieku komputery wyposażono już w kompilatory języka symbolicznego, co znacznie zwiększało efektywność programowania i użytkowania systemów liczących. Te zaś dzięki powstaniu i rozwojowi systemów operacyjnych zyskały właściwości wieloprogramowości (użytkownik mógł już uruchamiać cały „wsad" programów nie troszcząc się o to, jak będzie organizowany ich proces realizacji w komputerze) i wielodostępności (z zasobów komputera może korzystać wielu użytkowników i to bez troski o to, jak ich żądania będą przez komputer realizowane).

W omawianym okresie rozwój techniczny komputerów, a także środków telekomunikacji, uczynił możliwym połączenie odległych od siebie komputerów w celu bezpośredniego przesyłania danych między nimi.

W lipcu 1969 roku Neil Armstrong postawił nogę na Księżycu, co – dzięki bezpośredniej transmisji telewizyjnej – oglądało jednocześnie ponad pół miliarda ludzi. A trzy miesiące później uruchomiono eksperymentalną, pierwszą sieć komputerową. Rok ów był zatem szczególny – to zapewne jedna z najważniejszych dat w rozwoju „wieku informacji".

W 1971 roku w firmie Intel został zaprojektowany i wykonany pierwszy **mikroprocesor**. Składał się z czterech bloków funkcjonalnych (sterowania, jednostki arytmetyczno-logicznej, rejestrów, wewnętrznych szyn przesyłowych). Od pierwszego mikroprocesora **Intel 4004** o architekturze czterobitowej rozpoczął się trwający do dziś proces nieustannego rozwoju mikroprocesorów; w 1980 roku powstał pierwszy mikroprocesor trzydziestodwubitowy. Stanowił on zapowiedź istnej eksplozji informatycznej: w ciągu 30 lat objętość całego pokoju pełnego lamp elektronowych i innych elementów zmalała do rozmiarów płatka owsianego! Już w schyłku lat 70. stwierdzono, że gdyby w ciągu ostatnich 30 lat w przemyśle samochodowym dokonał się taki postęp jak w elektronice, to samochód Rolls-Royce'a można byłoby kupić za 2,5 dolara i przejechać nim dwa miliony mil zużywając na to galon benzyny.



Rysunek 14.
ENIAC i jego replika (1996) [źródło: 7]

Ale „złoty okres" informatyki miał dopiero nastąpić. Obliczenia wykonywane w 1946 roku na maszynie ENIAC, w 1982 roku wykonywał już mikrokomputer zbudowany z jednego lub kilku układów scalonych i mieszczący się bez trudu w szufladzie biurka. Rok później mikrokomputer IBM PC, od którego zaczął się „boom PC" został przez tygodnik „Time" wybrany „człowiekiem roku". Zanim to nastąpiło, IBM przede wszystkim dzięki modelom 360 i 370 wyznaczał główne kierunki rozwoju na świecie.

Rysunek 15.
IBM 360 i IBM 370 [źródło: 7]

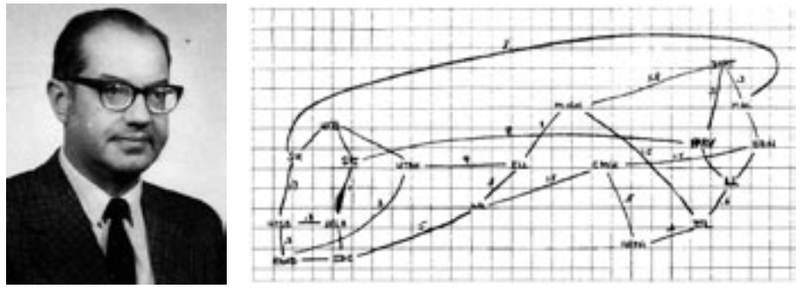
Rysunek 16.
Superkomputer CRAY [źródło: 7]

Obecnie na całym świecie wykorzystywane są miliony (miliardy?) komputerów osobistych: od desktopów (czyli na biurko), poprzez laptopy (czyli, do torby), notebooki i subnotebooki, po palmtopy (czyli do ręki). Rzecz jasna, oprócz PC funkcjonują komputery o większej mocy obliczeniowej: **stacje robocze** (ang. *workstation*), **minikomputery**, komputery (ang. *mainframe*) i **superkomputery** (np. CRAY). O dekadzie lat 80. XX wieku powiedziano, że była dekadą PC-tów, natomiast dekada lat 90. była dekadą sieci komputerowych, bowiem „Dopiero sieć to komputer" (hasło wymyślone przez firmę Sun Microsystems).

5 HISTORIA DO INTERNETU

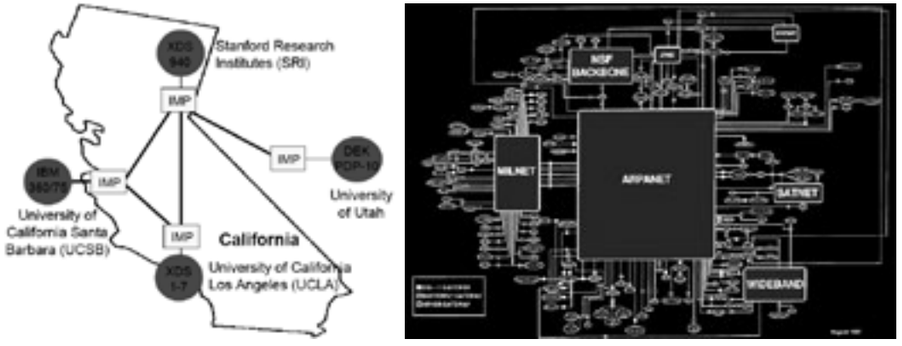
Siecią komputerową jest system, który tworzą wzajemnie połączone komputery zdolne do wymiany informacji między sobą. Połączenia w sieci mogą być realizowane za pomocą łączy przewodowych, radiowych, radioliniowych, mikrofalowych, światłowodowych i satelitarnych. Sieci komputerowe budowane są w celu: zapewnienia użytkownikom dostępu do wszystkich programów, danych i innych zasobów liczeniowych niezależnie od przestrzennej lokalizacji użytkowników i tych zasobów, a także dla łatwości aktualizacji informacji w odległych bazach danych i uzyskania wysokiej niezawodności przez stworzenie alternatywnych dróg sięgania do zasobów komputerowych. Ze względu na zasięg terytorialny przyjmuje się podział sieci teleinformatycznych na: lokalne (LAN – do kilku kilometrów), miejskie (MAN – do kilkudziesięciu kilometrów) i rozlegle (WAN – rozwinięte na dowolnym obszarze), a także globalne.

Niewątpliwie na całej kuli ziemskiej obecnie trwa „boom sieciowy": budowane są sieci zarówno ograniczone do użytkowników określonej organizacji, jak i sieci o powszechnym dostępie, a tempo sprzedaży technologii sieciowych wzrasta z roku na rok. Rosną także wymagania stawiane sieciom dotyczące funkcjonalności i niezawodności, ochrony zasobów i bezpieczeństwa sieci, a przede wszystkim zakresu oferowanych usług informacyjnych. Wzmaga się zainteresowanie sieciami multimedialnymi integrującymi, w celu efektywnego oddziaływania na odbiorcę, wszystkie typy informacji: video (pełny ruch) – audio (głos, dźwięk) – data (dane, teks, grafika).



Rysunek 17.
Paul Baran i jego koncepcja sieci

Reakcją na umieszczenie przez Rosjan w 1957 roku pierwszego sztucznego satelity w Kosmosie było powołanie przez Departament Obrony USA agencji DARPA, która zleciła korporacji RAND opracowanie bezpiecznego, rozległego systemu komputerowego spełniającego wymogi bezpieczeństwa narodowego. Koncepcję takiego systemu opracował w 1962 roku Paul Baran, emigrant z Polski już wykształcony w USA. Pierwszą próbę uruchomienia systemu przeprowadzono 23 września 1969 roku i tę datę można przyjąć za początek rozwoju sieci komputerowych – powstała bowiem sieć ARPANET.

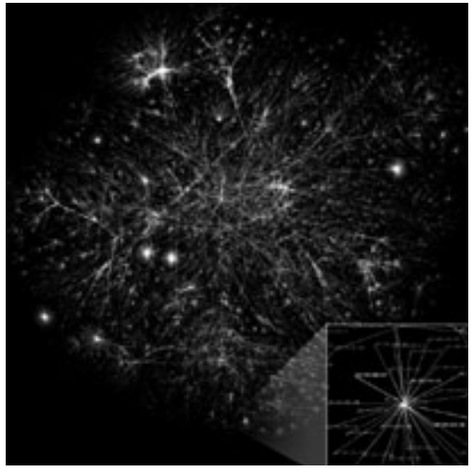


Rysunek 18.
ARPANET (grudzień 1969) [źródło: withfriendship/user/Athiv/arpanet.php]

ARPANET w latach 70. XX wieku rozwijała się także w kierunku zastosowań niemilitarnych i to w takim stopniu, że pod koniec dekady było niezbędne opracowanie nowych standardów komunikacyjnych (w 1982 r. powstał protokół TCP/IP). Około roku 1980 połączono ją z innymi sieciami (Usenet i BITNET) i w ten sposób utworzono mieszankę wielu sieci. W 1990 roku powstał **Internet** – największe wydarzenie ostatniej dekady XX wieku. Wynalazca **WWW** (ang. *World Wide Web*) Tim Berners-Lee znalazł się na liście dwudziestu najważniejszych uczonych XX wieku. Czym jest dziś Internet? Na pewno ogromną siecią, oplatającą centra komputerowe niemal na całym świecie, a natura tej „splątanej pajęczyny sieci komputerowych" uniemożliwia jakąkolwiek ocenę jej rozmiarów. Ścisła definicja Internetu opisuje to zjawisko obecnie jako „sieć łączącą wiele innych sieci korzystających z protokołu TCP/IP połączonych za pośrednictwem bram i korzystających ze wspólnej przestrzeni adresowej". Internet dzisiaj to sieć sieci, zasoby i usługi dostępne w tej sieci oraz społeczność Internetu.

Nie jest to definicja zadowalająca, bowiem niełatwo jest określić usługi dostępne w Internecie, zaś ich zakres wzrasta niemal z miesiąca na miesiąc, chociaż do najważniejszych z pewnością należą trzy: poczta elektroniczna, przesyłanie plików i interakcyjna praca na odległych komputerach. To o Internecie powiedział Stanisław Lem, że stanowi odpowiedź na pytanie, które nie zostało jeszcze postawione.

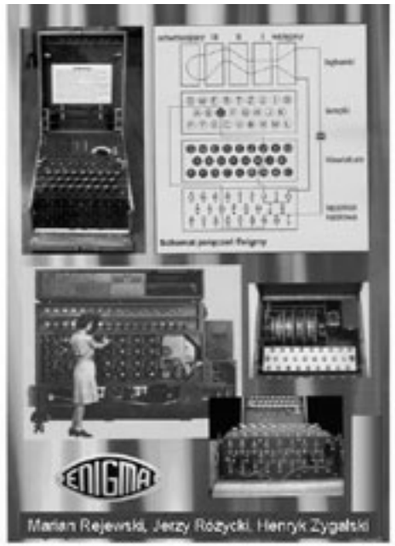
Rysunek 19.
Tim Berners-Lee



Rysunek 20.
„Galaktyka" Internetu

6 HISTORIA KOMPUTERÓW W POLSCE

Pierwszym znanym konstruktorem maszyn liczących w Polsce był żyjący w latach 1769-1842 w Hrubieszowie Abraham Stern. W 1812 roku zbudował on czterodziałaniową maszynę rachunkową, zaś 4 lata później urządzenie do obliczania pierwiastków kwadratowych, które w 1817 roku połączył tworząc pięciodziałaniowe urządzenie do liczenia. Warto dodać, że Stern zawdzięcza wykształcenie i opiekę Stanisławowi Staszicowi. W tygodniku „Wędrowiec" z 24 czerwca 1882 roku opisano polską maszynę rachunkową konstrukcji zegarmistrza Izraela Abrahama Staffela – można ją obejrzeć w Muzeum Techniki w Warszawie.



Rysunek 21.
Enigma

Zimą 1932/1933 trzech matematyków Marian Rejewski, Jerzy Różycki i Henryk Zygalski odniosło zwycięstwo nad niemieckim systemem szyfrowania Enigma. Skonstruowana w 1918 roku Enigma była urządzeniem elektromechanicznym zasilanym prądem z bateryjki 4V, przypominającym maszynę do pisania. Złamanie kodu Enigmy przyniosło podczas II wojny światowej możliwość rozszyfrowania depesz niemieckich, co długo było tajną bronią aliantów.

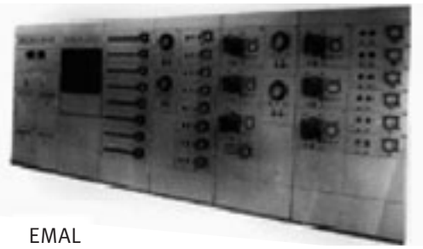
Dzieje informatyki w Polsce zasługują na uwagę i kompetentne opisanie, chociażby z dwóch powodów: 1) już dwa lata po uruchomieniu maszyny ENIAC podjęto prace nad maszynami liczącymi, a działo się to mimo zimnowojennego klimatu; 2) Polska, pomimo dotkliwej luki technologicznej w stosunku do krajów wysokorozwiniętych, stworzyła przemysł komputerowy (ZE Elwro, MERA), stając się eksporterem urządzeń informatycznych.

Jak zwykle początki były bardzo skromne: 23 grudnia 1948 roku powstała Grupa Aparatów Matematycznych (GAM), przy tworzonym wówczas Państwowym Instytucie Matematycznym, organizowanym przez prof. Kazimierza Kuratowskiego. Zadanie, jakie stało przed zespołem było prawie nierealne – wspominał po latach jeden z uczestników GAM i późniejszy jego kierownik Leon Łukaszewicz – albowiem ENIAC, wzór dany do naśladowania, był gigantem, jednym ze szczytowych osiągnięć ówczesnej technologii amerykańskiej. Od jesieni 1950 roku w Instytucie Matematycznym trwały prace nad Analizatorem Równań Algebraicznych (ARL), Analizatorem Równań Różniczkowych (ARR) i Elektroniczną Maszyną Automatycznie Liczącą (EMAL). Jesienią 1958 roku siłami Zakładu Aparatów Matematycznych (ZAM) uruchomiono pierwszą polską poprawnie funkcjonującą maszynę cyfrową XYZ, której architektura była uproszczeniem architektury IBM 701. Udoskonalona

maszyna XYZ została wyprodukowana jako ZAM 2, zaś niewątpliwym jej atutem było oprogramowanie – System Automatycznego Kodowania (SAKO) określany jako „polski Fortran".

Kazimierz Kuratowski

Janusz Groszkowski



EMAL

Romuald Marczyński

Leon Łukaszewicz

XYZ

Rysunek 22.
Członkowie grupy aparatów matematycznych i ich pierwsze komputery

W 1963 roku wrocławskie zakłady Elwro podjęły przemysłową produkcję komputerów UMC-1, zaprojektowanych przez Zdzisława Pawlaka. Tamże, od roku 1964 produkowano komputery z serii ODRA. W Wojskowej Akademii Technicznej opracowano cyfrowy analizator różnicowy JAGA 63 oraz pierwszy komputer analogowy ELWAT. W 1968 roku rozpoczęto międzynarodowe prace zmierzające do skonstruowania rodziny komputerów Jednolitego Systemu (RIAD), a cztery lata później zmontowano w Elwro komputer R-30. W 1975 roku w Zakładach Wytwórczych Przyrządów Pomiarowych ERA rozpoczęto produkcję minikomputera MERA 300 oraz MERA-400, w Instytucie Badań Jądrowych uruchomiono system abonencki CYFRONET, a na Politechnice Wrocławskiej WASC (Wielodostępny Abonencki System Cyfrowy). Wymieniając głównych producentów sprzętu komputerowego należy zauważyć, że zgodnie z ówczesnymi rozwiązaniami organizacyjnymi w przemyśle produkcja ulokowana była w Zjednoczeniu Przemysłu Automatyki i Aparatury Pomiarowej MERA. Oprócz fabryk WZE Elwro i ZSM Era duże znaczenie miały takie przedsiębiorstwa jak Meramat-Warszawa (pamięci taśmowe PT-3M i PT-305/310, systemy wprowadzania danych MERA-9150), ZPM Mera-Błonie (drukarki wierszowe do komputerów Odra i Riad, drukarki mozaikowe DZM-180 na licencji francuskiej firmy Logabax), Mera-KFAP Kraków (w latach 80. XX wieku uruchomiono produkcję komputerów 8-bitowych PSPD-90).

Przytoczone wyżej wydarzenia to jedynie wybrane przykłady istotnych osiągnięć charakteryzujących początki budowy komputerów w Polsce. Swoistym ich uwieńczeniem było wprowadzenie Internetu do Polski i początki transformacji – wyłaniania się polskiego społeczeństwa informacyjnego.

Za symboliczną datę wprowadzenia Internetu do naszego kraju uważa się 17 sierpnia 1991 roku, kiedy to Ryszard Pietrak, fizyk z UW, nawiązał łączność komputerową w oparciu o protokół IP z Janem Sorensenem z Uniwersytetu w Kopenhadze.

ZAKOŃCZENIE

Legendarny szef imperium Microsoftu William H. Gates nie kryje, że najbliższa przyszłość należy do globalnej **infostrady** opartej na rozwoju Internetu, która stanie się podstawą globalnej wioski. Dzięki ludziom takim jak Gates i, rzecz jasna, wielu jego poprzednikom, których wspominaliśmy wcześniej, ziściła się przepowiednia Marshalla McLuhana z lat 60. minionego wieku o naszej planecie jako globalnej wiosce, czyli globalnym społeczeństwie informacyjnym (lub społeczeństwie globalnej informacji).

Trudno pisać jakieś zakończenie tej fascynującej przygody ludzi, której ostatni etap obejmuje lata „od maszyny ENIAC do Internetu". Może łatwiej zastanowić się nad tym, czym jest obecnie informatyka. To nie tylko *computer science*, ani nawet *computer engineering*, ale złożona dziedzina naukowej wiedzy multi- i interdyscyplinamej, która nie jest wolna (nie może być) od refleksji humanistycznej i uwzględniania społecznego kontekstu.

Informatyka obejmuje rozwój automatyzacji pracy umysłowej:

- inżynieria obliczeń – komputer jako środek do obliczeń (*computer as a computer*);
- inżynieria rozwiązywania problemów – komputer jako środek do rozwiązywania problemów (*computer as a problem solver*);
- inżynieria informacji – komputer jako środek do gromadzenia i przetwarzania informacji (*computer as an information collector and processor*);
- inżynieria wiedzy – komputer jako ekspert *(computer as an expert*).

Kontekst społeczny najlepiej zdefiniował Peter F. Drucker: „Podstawowym bogactwem gospodarczym jest wiedza (...). Grupą rządzącą będą robotnicy wiedzy, dyrektorzy do spraw wiedzy, specjaliści od wiedzy i przedsiębiorcy, którzy mają intuicję, jak alokować wiedzę, żeby ją wykorzystać tak samo, jak kapitaliści wiedzieli, gdzie alokować kapitał" [7, s. 129].

Dzięki cybernetyce Norberta Wienera i teorii informacji Claude'a E. Shannona, informacja stała się – obok materii i energii – kategorią mierzalną. Dzięki zastosowaniom zaawansowanych technologii informacja stała się towarem, zaś zasoby informacyjne organizacji – jej zasobem strategicznym. Dzięki zdumiewającemu rozwojowi Internetu powstała Nowa gospodarka (*E-biznes, New Economy, Net Economy*), czyli Gospodarka oparta na wiedzy (GOW). Rozwija się telepraca i teleedukacja, a funkcjonownie sieci teleinformatycznych stało się warunkiem efektywności administracji państwowej i tzw. krytycznej infrastruktury państwa (KIP). Przykłady podobne można mnożyć bez końca. I nie po to, by dowodzić, że żyjemy w „wieku informacji", bo to wszak oczywiste dla każdego. W czołowych laboratoriach naukowych trwają prace nad biokomputerem i komputerem kwantowym. Coraz bardziej przybliża się wizja realizacji komputera HAL 9000 z filmu „2001: Odyseja kosmiczna" Stanleja Kubricka. Dzięki nieograniczonemu wprost rozwojowi Internetu nastąpiła swoista kompresja czasu i przestrzeni.

Rozwój informatyki i telekomunikacji, technologii informacyjnych: komputerów i sieci teleinformatycznych, wreszcie powstanie i rozwój społeczeństwa informacyjnego niesie wielkie nadzieje, którym towarzyszą też obawy i zagrożenia. Pojawiło się bowiem groźne zjawisko cyberterroryzmu. Istnieje potrzeba gruntownych analiz systemowych. Pięknie pisał Antoine de Saint-Exupery: „Ci, których przerażają postępy techniki, nie odróżniają celu od środków. Kto staje do walki z nadzieją na zdobycie dóbr materialnych li tylko, nie zbierze nic, dla czego warto żyć".

Rysunek 23.
HAL 9000 – bohater filmu „2001: Odyseja kosmiczna" [źrodło: 7]

LITERATURA

1. Drucker P. F., *Zarządzanie w XXI wieku,* Muza SA, Warszawa 2000
2. Ifrah G., *Historia powszechna cyfr*, t. I, II, WAB, Warszawa 2006
3. Ligonniere R., *Prehistoria i historia komputerów*, Ossolineum, Wrocław 1992
4. Sienkiewicz P., *Od Eniaca do Internetu i społeczeństwa wiedzy*, Zeszyty Naukowe WWSI, Nr 1, Warszawa 2006
5. Sienkiewicz P., Nowak J.S., *Sześćdziesiąt lat polskiej informatyki*, Zeszyty Naukowe WWSI, Nr 3, Warszawa 2009
6. Targowski A., *Informatyka. Modele systemów i rozwoju*, PWE, Warszawa 1980
7. Wurster C., *Computers: An Illustrated History*, Taschen, Köln 2002

Algorytmika i programowanie

Porządek wśród informacji kluczem do szybkiego wyszukiwania

Czy wszystko można policzyć na komputerze

Dlaczego możemy się czuć bezpieczni w sieci, czyli o szyfrowaniu informacji

Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew

O relacjach i algorytmach

Porządek wśród informacji kluczem do szybkiego wyszukiwania

Maciej M. Sysło

Uniwersytet Wrocławski, UMK w Toruniu

syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl

http://mmsyslo.pl/

Streszczenie

Wykład ten jest wprowadzeniem do algorytmiki i zawiera elementy implementacji algorytmów w języku programowania. Na przykładach bardzo prostych problemów przedstawione jest podejście do rozwiązywania problemów w postaci algorytmów i do ich komputerowej implementacji w języku programowania. Omawiane są m.in.: specyfikacja problemu, schematy blokowe algorytmów, podstawowe struktury danych (ciąg i tablica) oraz pracochłonność algorytmów. Wykorzystywane jest oprogramowanie edukacyjne, ułatwiające zrozumienie działania algorytmów i umożliwiające wykonywanie eksperymentów z algorytmami bez konieczności ich programowania. Przytoczono ciekawe przykłady zastosowań omawianych zagadnień.

Zakres tematyczny tego wykładu obejmuje problemy poszukiwania elementów (informacji) w zbiorach nieuporządkowanych i uporządkowanych oraz problem porządkowania (sortowania), któremu poświęcamy szczególnie wiele miejsca ze względu na jego znaczenie w obliczeniach. Algorytmy sortowania są okazją, by na ich przykładzie zademonstrować różne techniki rozwiązywania problemów, jak metodę dziel i zwyciężaj oraz rekurencję.

Spis treści

Część materiałów pochodzi z książek, których Maciej M. Sysło jest autorem lub współautorem [2, 7, 8]. Oprogramowanie edukacyjne wykorzystane w materiałach można pobrać ze strony http://mmsysło.pl/. Tabele i rysunki pochodzą z książek autora lub są przez niego opracowane.

1 PRZESZUKIWANIE ZBIORU

Zajmiemy się bardzo prostym problemem, który każdy z Was rozwiązuje wielokrotnie w ciągu dnia. Chodzi o znajdowanie w zbiorze elementu, który ma określoną własność. Oto przykładowe sytuacje problemowe:

- znajdź najwyższego ucznia w swojej klasie; a jak zmieni się Twój algorytm, jeśli chciałbyś znaleźć w klasie najniższego ucznia?
- poszukaj w swojej klasie ucznia, któremu droga do szkoły zabiera najwięcej czasu;
- znajdź najstarszego ucznia w swojej szkole; jak zmieni się Twój algorytm, gdybyś chciał znaleźć w szkole najmłodszego ucznia?
- odszukaj największą kartę w potasowanej talii kart;
- znajdź najlepszego gracza w warcaby w swojej klasie (zakładamy, że wszyscy potrafią grać w warcaby);
- odnajdź najlepszego tenisistę w swojej klasie.

Tego typu problemy pojawiają się bardzo często, również w obliczeniach komputerowych, i na ogół są rozwiązywane w dość naturalny sposób – przeglądany jest cały zbiór, by znaleźć poszukiwany element. Zastanowimy się, jak dobra jest to metoda, i czy może istnieje szybsza metoda znajdowania w zbiorze elementu o określonych własnościach.

Postawiony problem może wydać się zbyt prosty, by zajmować się nim na informatyce – każdy uczeń zapewne potrafi wskazać metodę rozwiązywania, polegającą na systematycznym przeszukaniu całego zbioru danych. To metoda **przeszukiwania** ciągu, którą można nazwać przeszukiwaniem **liniowym**. Przy tej okazji w dyskusji pojawi się zapewne również **metoda pucharowa**, która jest często stosowana w rozgrywkach turniejowych. Metodzie pucharowej odpowiada **drzewo algorytmu**, służące wyjaśnieniu wielu innych kwestii związanych głównie ze złożonością algorytmów.

Założenia

Poczyńmy najpierw pewne założenia.

Założenie 1. Na początku wykluczamy, że przeszukiwane zbiory elementów są uporządkowane, np. klasa – od najwyższego do najniższego ucznia lub odwrotnie, szkoła – od najmłodszego do najstarszego ucznia lub odwrotnie, Gdyby tak było, to rozwiązanie wymienionych wyżej problemów i im podobnych byłoby dziecinnie łatwe – wystarczyłoby wziąć element z początku albo z końca takiego uporządkowania.

Założenie 2. Przyjmujemy także, że nie interesują nas algorytmy rozwiązywania przedstawionych sytuacji problemowych, które w pierwszym kroku porządkują zbiór przeszukiwany, a następnie już prosto znajdują poszukiwane elementy – to założenie wynika z faktu, że sortowanie ciągu jest znacznie bardziej pracochłonne niż znajdowanie wyróżnionych elementów w ciągu. Przeszukiwaniem zbiorów uporządkowanych zajmiemy się w punkcie 6.2.

Z powyższych założeń wynika dość naturalny wniosek, że aby znaleźć w zbiorze poszukiwany element musimy przejrzeć wszystkie elementy zbioru, gdyż jakikolwiek pominięty element mógłby okazać się tym szukanym elementem.

Przy projektowaniu algorytmów istotne jest również określenie, jakie działania (operacje) mogą być wykonywane w algorytmie. W przypadku problemu poszukiwania szczególnego elementu w zbiorze wystarczy, jeśli będziemy umieli porównać elementy między sobą. Co więcej, w większości wymienionych problemów porównanie elementów sprowadza się do porównania liczb, właściwych dla branych pod uwagę elementów, a oznaczających: wzrost, czas na dojście do szkoły (np. liczony w minutach), wiek. Elementy zbiorów utożsamiamy więc z ich wartościami i wartości te nazywamy **danymi**, chociaż często prowadzimy rozważania w języku problemu, posługując się nazwami elementów: wzrost, wiek itp.

Przy porównywaniu kart należy uwzględnić ich kolory i wartości. Natomiast, by znaleźć w klasie najlepszego gracza w tenisa, należy zorganizować turniej – ten problem omówimy w dalszej części wykładu.

1.1 SPECYFIKACJA PROBLEMU I ALGORYTM

Dla uproszczenia rozważań można więc założyć, że dany jest pewien zbiór liczb *A* i w tym zbiorze należy znaleźć liczbę najmniejszą (lub największą). Przyjmijmy, że tych liczb jest *n* i oznaczmy je jako ciąg liczb: $x_1, x_2, \ldots, x_n$. Możemy teraz podać **specyfikację** rozważanego problemu:

Problem Min – Znajdowanie najmniejszego elementu w zbiorze

Dane: Liczba naturalna *n* i zbiór *n* liczb, dany w postaci ciągu $x_1, x_2, \ldots, x_n$.

Wynik: Najmniejsza spośród liczb $x_1, x_2, \ldots, x_n$ – oznaczmy jej wartość przez *min*.

Algorytm Min

Dla powyższej specyfikacji podamy teraz algorytm, który polega na przejrzeniu ciągu danych od początku do końca. Opis algorytmu poprzedza specyfikację problemu, który ten algorytm rozwiązuje – tak będziemy na ogół postępować w każdym przypadku. Przedstawiony poniżej opis algorytmu ma postać **listy kroków**. O innych sposobach przedstawiania algorytmów piszemy w dalszej części.

Algorytm Min – znajdowanie najmniejszego elementu w zbiorze

Dane: Liczba naturalna *n* i zbiór *n* liczb, dany w postaci ciągu $x_1, x_2, \ldots, x_n$.

Wynik: Najmniejsza spośród liczb $x_1, x_2, \ldots, x_n$ – oznaczmy jej wartość przez *min*.

Krok 1. Przyjmij za *min* pierwszy element w zbiorze (w ciągu), czyli przypisz $min := x_1$.

Krok 2. Dla kolejnych elementów x_i, gdzie $i = 2, 3, \ldots, n$, jeśli *min* jest większe niż x_i, to za *min* przyjmij x_i, czyli, jeśli $min > x_i$, to przypisz $min := x_i$.

Uwaga. W opisie algorytmu pojawiło się polecenie (instrukcja) **przypisania**[13], np. $min := x_1$, w której występuje symbol :=, złożony z dwóch znaków: dwukropka i równości. Przypisanie oznacza nadanie wielkości zmiennej stojącej po lewej stronie tego symbolu wartości równej wartości wyrażenia (w szczególnym przypadku to wyrażenie może być zmienną) znajdującego się po prawej stronie tego symbolu. Przypisanie jest stosowane na przykład wtedy, gdy należy zmienić wartość zmiennej, np. $i := i + 1$ – w tym przypadku ta sama zmienna występuje po lewej i po prawej stronie symbolu przypisania. Polecenie przypisania występuje w większości języków programowania, stosowane są tylko różne symbole i ich uproszczenia dla jego oznaczenia. W schemacie blokowym na rysunku 2 symbolem przypisania jest strzałka ←.

Metoda zastosowana w algorytmie **Min**, polegająca na badaniu elementów ciągu danych w kolejności, w jakiej są ustawione, nazywa się **przeszukiwaniem liniowym**, w odróżnieniu od przeszukiwania przez połowienie (lub binarnego), o którym jest mowa w rozdziale 6.

Demonstracja działania algorytmu Min

Działanie algorytmu **Min** można zademonstrować posługując się programem edukacyjnym **Maszyna sortująca** (patrz rys. 1), który jest udostępniony wraz z tymi materiałami. W tym programie można ustalić liczbę elementów w ciągu (między 1 i 16) i wybrać rodzaj ciągu danych, który może zawierać elementy: losowe, posortowane rosnąco lub posortowane malejąco. Ponieważ ten program służy do porządkowania ciągu, o czym będzie mowa w dalszej części zajęć (patrz rozdz. 4), zalecamy tutaj wykonanie demonstracji pracą krokową i przerwanie jej po znalezieniu najmniejszego elementu w ciągu – następuje to w momencie, gdy dolna zielona strzałka znajdzie się pod ostatnim elementem w ciągu i wygaszony zostanie czerwony kolor, wyróżniający porównywane elementy.

[13] Polecenie przypisania jest czasem nazywane niepoprawnie podstawieniem.

Algorytm Max – Prosta modyfikacja algorytmu Min

Podany powyżej algorytm **Min**, służący do znajdowania najmniejszej liczby w ciągu danych, może być łatwo zmodyfikowany do znajdowania największego elementu ciągu – wystarczy w tym celu zmienić tylko zwrot nierówności.

Jeszcze jedna modyfikacja algorytmu Min

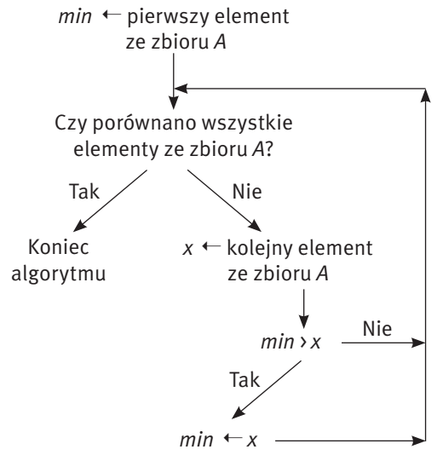
Często, poza znalezieniem elementu najmniejszego (lub największego), chcielibyśmy znać jego położnie, czyli miejsce (numer) w ciągu danych. W tym celu wystarczy wprowadzić nową zmienną, np. *imin*, w której będzie przechowywany numer aktualnie najmniejszego elementu – szczegóły tej modyfikacji pozostawiamy do samodzielnego wykonania.

Rysunek 1.
Demonstracja działania algorytmu **Min** w programie **Maszyna sortująca**

1.2 SCHEMAT BLOKOWY ALGORYTMU MIN

Schemat blokowy algorytmu (zwany również **siecią działań** lub **siecią obliczeń**) jest graficznym opisem: działań składających się na algorytm, ich wzajemnych powiązań i kolejności ich wykonywania. W informatyce miejsce schematów blokowych jest pomiędzy opisem algorytmu w postaci listy kroków, a programem, napisanym w wybranym języku programowania. Należą one do kanonu wiedzy informatycznej, nie są jednak niezbędnym jej elementem, chociaż mogą okazać się bardzo przydatne na początkowym etapie projektowania algorytmów i programów komputerowych. Z drugiej strony, w wielu publikacjach algorytmy są przedstawiane w postaci schematów blokowych, pożądana jest więc umiejętność ich odczytywania i rozumienia. Ten sposób reprezentowania algorytmów pojawia się również w zadaniach maturalnych z informatyki.

Na rysunku 2 przedstawiono schemat algorytmu **Min**. Jest to bardziej schemat ideowy działania algorytmu niż jego schemat blokowy, jest bardzo ogólny, gdyż zawarto w nim jedynie najważniejsze polecenia i pominięto szczegóły realizacji poszczególnych poleceń.



Rysunek 2.
Pierwszy, zgrubny schemat blokowy algorytmu **Min**

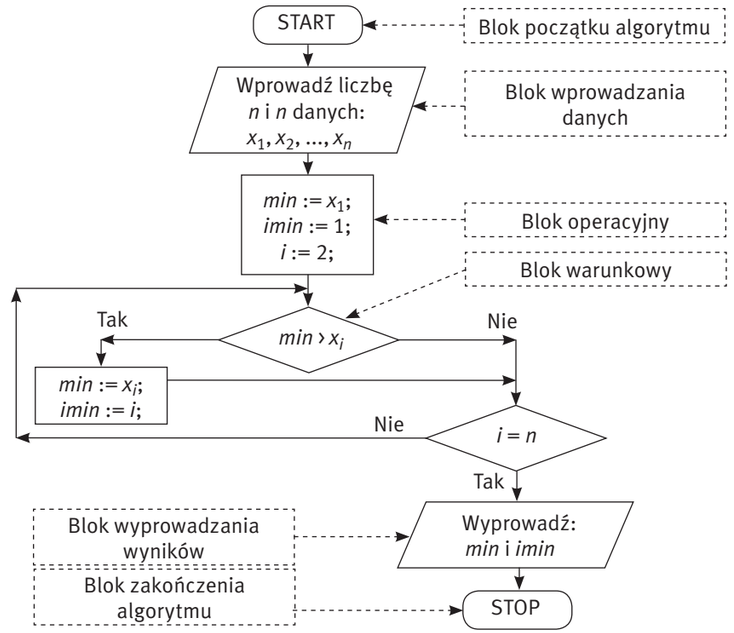
Na rysunku 3 przedstawiony jest szczegółowy schemat blokowy algorytmu **Min**, uwzględniono w nim również modyfikację zaproponowaną pod koniec poprzedniego punktu oraz zawarto bloki wczytywania danych i wyprowadzania wyników. Jest on zbudowany z bloków, których kształty zależą od rodzaju wykonywanych w nich poleceń. I tak mamy:

- blok początku i blok końca algorytmu;
- blok wprowadzania (wczytywania) danych i wyprowadzania (drukowania) wyników – bloki te mają taki sam kształt;
- blok operacyjny, w którym są wykonywane operacje przypisania;
- blok warunkowy, w którym jest formułowany warunek;
- blok informacyjny, który może służyć do komentowania fragmentów schematu lub łączenia ze sobą części większych schematów blokowych.

Nie istnieje pełny układ zasad poprawnego konstruowania schematów blokowych. Można natomiast wymienić dość naturalne zasady, wynikające z charakteru bloków:

- schemat zawiera dokładnie jeden blok początkowy, ale może zawierać wiele bloków końcowych – początek algorytmu jest jednoznacznie określony, ale algorytm może się kończyć na wiele różnych sposobów;
- z bloków: początkowego, wprowadzania danych, wyprowadzania wyników, operacyjnego wychodzi dokładnie jedno połączenie, może jednak wchodzić do nich wiele połączeń;
- z bloku warunkowego wychodzą dwa połączenia, oznaczone wartością warunku: TAK i NIE;
- połączenia wychodzące mogą dochodzić do bloków lub do innych połączeń.

Schematy blokowe mają wady trudne do wyeliminowania. Łatwo konstruuje się z ich pomocą algorytmy dla obliczeń niezawierających iteracji i warunków, którym w schematach odpowiadają rozgałęzienia, nieco trudniej dla obliczeń z rozgałęzieniami, a trudniej dla obliczeń iteracyjnych (wczytywanie ciągu i realizacja kroku 2 z algorytmu **Min**). Za pomocą schematów blokowych nie można w naturalny sposób zapisać rekurencji oraz objaśnić znaczenia wielu pojęć związanych z algorytmiką, takich np. jak programowanie z użyciem procedur, czyli podprogramów z parametrami.



Rysunek 3.
Szczegółowy schemat blokowy algorytmu **Min**

1.3 KOMPUTEROWA REALIZACJA ALGORYTMU

Reprezentowanie danych w algorytmach

Zanim podamy komputerową realizację pierwszego algorytmu, musimy ustalić, w jaki sposób będą reprezentowane w algorytmie dane i jak będziemy je podawać do algorytmu.

Wspomnieliśmy już przy projektowaniu algorytmu **Min**, że dane dla tego algorytmu są zapisane w postaci ciągu n liczb $x_1, x_2, \ldots, x_n$. Liczby te mogą być naturalne (czyli całkowite i dodatnie), całkowite lub dziesiętne (np. z kropką). Rodzaj danych liczb nazywa się **typem danych**. Przyjmujemy dla uproszczenia, że danymi dla algorytmów omawianych na tych zajęciach są liczby całkowite.

Zbiór danych, który jest przetwarzany za pomocą algorytmu, może być podawany (czytany) z klawiatury, czytany z pliku lub może być zapisany w innej strukturze danych.

Dla wygody będziemy zakładać, że wiemy, ile będzie danych i ta liczba danych występuje na początku danych – jest nią liczba n w opisie algorytmu **Min**. W ogólności, jeśli np. dane napływają do komputera z jakiegoś urządzenia pomiarowego, możemy nie wiedzieć, ile ich będzie. W takich przypadkach wprowadza się dane aż do specjalnego znaku, który świadczy o ich końcu. Takim znakiem może być koniec pliku, jeśli dane są umieszczone w pliku. Może nim być również wyróżniona liczba zwana **wartownikiem**, której rolą jest pilnowanie końca danych. O użyciu wartownika powiemy później (punkt 6.1), a teraz przedstawimy realizację algorytmu **Min** dla danych podawanych z klawiatury.

Uwaga. Piszemy „zbiór danych”, ale użycie tutaj pojęcia zbiór nie zawsze jest matematycznie poprawne. W zbiorze elementy się nie powtarzają, a w danych mogą występować takie same liczby. Całkowicie poprawnie powinniśmy mówić o tzw. **multizbiorach**, czyli zbiorach, w których elementy mogą się powtarzać, ale dla wygody będziemy stosować pojęcie zbioru, pamiętając, że mogą powtarzać się w nim elementy. Sytuację

upraszcza nam założenie, że zbiór danych w algorytmie będzie przedstawiony w postaci ciągu elementów, a w ciągu elementy mogą się powtarzać.

Komputerowa realizacja algorytmu Min – dane z klawiatury

Zapiszemy teraz algorytm **Min** posługując się poleceniami języka Pascal. Przyjmujemy, że dane są podawane z klawiatury – na początku należy wpisać liczbę wszystkich danych, a po niej kolejne elementy danych w podanej ilości. Po każdej danej liczbie naciskamy klawisz Enter. Program, który jest zapisem algorytmu **Min** w języku Pascal, jest umieszczony w drugiej kolumnie w tabeli 1. Język Pascal jest zrozumiały dla komputerów, które są wyposażone w specjalne programy, tzw. **kompilatory**, przekładające programy użytkowników na język wewnętrzny komputerów. Program w tabeli 1 bez większego trudu zrozumie także człowiek dysponujący opisem algorytmu **Min** w postaci listy kroków. W wierszu nr 2 znajdują się **deklaracje**, czyli opisy zmiennych – komputer musi wiedzieć, jakimi wielkościami posługuje się algorytm i jakiego są one **typu**, `integer` oznacza liczby całkowite. Polecenia w językach programowania nazywają się **instrukcjami**. Jeśli chcemy z kilku instrukcji zrobić jedną, to tworzymy z nich **blok**, umieszczając na jego początku słowo `begin`, a na końcu – `end`. Pojedyncze instrukcje kończymy **średnikiem**. Na końcu programu stawiamy kropkę.

Ada Augusta, córka Byrona, uznawana powszechnie za pierwszą programistkę komputerów, przełomowe znaczenie maszyny analitycznej Ch. Babbage'a, pierwowzoru dzisiejszych komputerów, upatrywała właśnie „w możliwości wielokrotnego wykonywania przez nią danego ciągu instrukcji, z liczbą powtórzeń z góry zadaną lub zależną od wyników obliczeń", a więc w iteracji.

Dwie instrukcje wymagają wytłumaczenia, chociaż również są dość oczywiste. W wierszach 6–11 znajdują się instrukcje, które realizują krok 2 algorytmu, polegający na wykonaniu wielokrotnie sprawdzenia warunku. Instrukcja, służąca do wielokrotnego wykonania innych instrukcji nazywa się **instrukcją iteracyjną** lub **instrukcją pętli**. W programie w tabeli 1 ta instrukcja zaczyna się w wierszu nr 6, a kończy w wierszu nr 11:

```
for i:=2 to n do begin
...
end
```

Ta instrukcja iteracyjna służy do powtórzenia **instrukcji warunkowej**, która zaczyna się w wierszu nr 8 i kończy w wierszu nr 10. Ma tutaj postać:

```
if min>x then begin
...
end
```

Inne typy instrukcji iteracyjnej i warunkowej będą wprowadzane sukcesywnie.

1.4. PRACOCHŁONNOŚĆ (ZŁOŻONOŚĆ) ALGORYTMU MIN

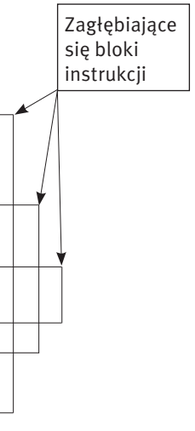
Problem znajdowania najmniejszego (lub największego) elementu w zbiorze jest jednym z elementarnych problemów najczęściej rozwiązywanych przez człowieka i przez komputer, dlatego interesujące jest pytanie, czy rozwiązujemy go możliwie najszybciej. W szczególności, czy podany przez nas algorytm **Min** i jego komputerowe **implementacje**[14] są najszybszymi metodami rozwiązywania tego problemu.

W algorytmach **Min** i **Max**, i w ich implementacjach, podstawową operacją jest porównanie dwóch elementów ze zbioru danych – policzmy więc, ile porównań jest wykonywanych w tych algorytmach. Liczba tych porównań w algorytmie zależy od liczby danych. W każdej iteracji algorytmu jest wykonywane jedno porównanie $min > x_i$, a zatem w każdym z tych algorytmów jest wykonywanych $n - 1$ porównań (tyle razy bowiem

[14] Terminem **implementacja** określa się w informatyce komputerową realizację algorytmu.

Tabela 1.
Program w języku Pascal (druga kolumna)

Lp	Program w języku Pascal	Odpowiedniki instrukcji po polsku
1.	`Program MinKlawiatura;`	nazwa programu
2.	`var i,imin,min,n,x:integer;`	deklaracja zmiennych: i, imin, min, n, x
3.	`begin`	początek głównego bloku programu
4.	`read(n);`	czytaj(n);
5.	`read(x); min:=x; imin:=1;`	czytaj(x); początek szukania min
6.	`for i:=2 to n do begin`	dla i:=2 do n wykonaj – początek iteracji
7.	`read(x);`	czytaj(x);
8.	`if min>x then begin`	jeśli min>x to – instrukcja warunkowa
9.	`min:=x; imin:=i`	min:=x; imin:=i
10.	`end`	koniec instrukcji warunkowej
11.	`end;`	koniec iteracji
12.	`write(imin,min)`	drukuj(imin, min)
13.	`end.`	koniec. – na końcu stawiamy kropkę



jest wykonywana iteracja w kroku 2). Pozostałe operacje służą głównie do organizacji obliczeń i ich liczba jest związana z liczbą porównań. Na przykład, operacja przypisania $min := x_i$ może być wykonana tylko o jeden raz więcej – w kroku 1 i $n - 1$ razy w kroku 2.

Możemy więc podsumować nasze rozumowanie:

> najmniejszy (lub największy) element w niepustym zbiorze danych można znaleźć wykonując o jedno porównanie mniej, niż wynosi liczba wszystkich elementów w tym zbiorze.

To nie jest specjalnie wielkie odkrycie, a jedynie sformułowanie dość oczywistej własności postępowania, które często wykonujemy niemal automatycznie, nie zastanawiając się nawet, w jaki sposób to robimy. Już większym wyzwaniem jest pytanie:

Czy w zbiorze złożonym z n liczb można znaleźć najmniejszy element wykonując mniej niż $n - 1$ porównań elementów tego zbioru?

Udzielimy negatywnej odpowiedzi na to pytanie[15], posługując się interpretacją wziętą z klasowego turnieju tenisa. Ile należy rozegrać meczów (to są właśnie porównania w przypadku tego problemu), aby wyłonić najlepszego tenisistę w klasie? Lub inaczej – kiedy możemy powiedzieć, że Tomek jest w naszej klasie najlepszym tenisistą? Musimy mieć pewność, że wszyscy pozostali uczniowie są od niego gorsi, czyli przegrali z nim, bezpośrednio lub pośrednio. A zatem każdy inny uczeń przegrał minimum jeden mecz, czyli rozegranych zostało przynajmniej tyle meczów, ilu jest uczniów w klasie mniej jeden. I to kończy nasze uzasadnienie.

Z dotychczasowych rozważań możemy wyciągnąć wniosek, że algorytmy **Min** i **Max** oraz ich komputerowe implementacje są najlepszymi algorytmami służącymi do znajdowania najmniejszego i największego elementu, gdyż wykonywanych jest w nich tyle porównań, ile musi wykonać jakikolwiek algorytm rozwiązywania tych problemów. O takim algorytmie mówimy, że jest **algorytmem optymalnym pod względem złożoności obliczeniowej**.

[15] Posługujemy się tutaj argumentacją zaczerpniętą z książki Hugona Steinhausa, [rozdz. III, 6].

2 KOMPLETOWANIE PODIUM ZWYCIĘZCÓW TURNIEJU

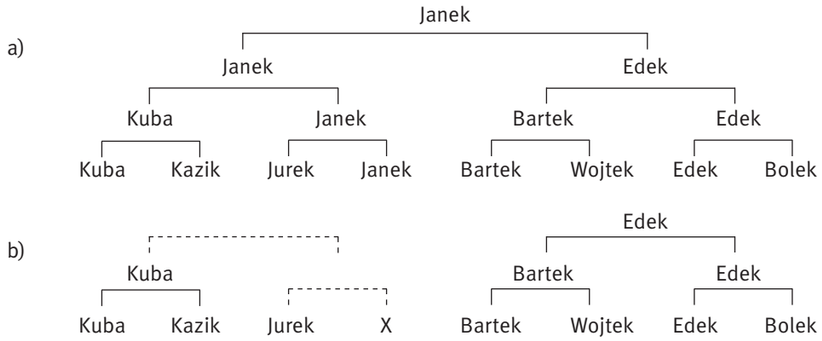
Przedstawione w poprzednim rozdziale postępowanie nie jest jedyną metodą służącą do znajdowania najlepszego elementu w zbiorze. Inną metodą jest tzw. **system pucharowy**, stosowany często przy wyłanianiu najlepszego zawodnika bądź drużyny w turnieju. W metodzie tej „porównanie" dwóch zawodników (lub drużyn), by stwierdzić, który jest lepszy („większy"), polega na rozegraniu meczu. W rozgrywkach systemem pucharowym zakłada się, że wszystkie mecze kończą się zwycięstwem jednego z zawodników, dlatego w dalszej części będziemy pisać o rozgrywkach w tenisa, a nie o turnieju w warcaby, gdyż w przypadku warcabów (jak i szachów) partie mogą kończyć się remisem, podczas gdy w meczach w tenisa można wymóc, by mecz między dwoma zawodnikami zawsze kończył się zwycięstwem jednego z nich.

Wyłanianie zwycięzcy w turnieju

Nurtować może pytanie, czy znajdowanie najlepszego zawodnika systemem pucharowym nie jest czasem metodą bardziej efektywną pod względem liczby wykonywanych porównań (czyli rozegranych meczy) niż przeszukiwanie liniowe, opisane w poprzednim rozdziale.

Na rysunku 4a) jest przedstawiony fragment turnieju, rozegranego między ośmioma zawodnikami. Zwycięzcą okazał się Janek po rozegraniu w całym turnieju siedmiu meczów. A zatem, podobnie jak w przypadku metody liniowej, aby wyłonić zwycięzcę, czyli najlepszego zawodnika (elementu) wśród ośmiu zawodników, należało rozegrać o jeden mecz mniej, niż wystąpiło w turnieju zawodników. Nie jest to przypadek. Ten fakt jest prawdziwy dla dowolnej liczby zawodników występujących w turnieju, rozgrywanym metodą pucharową – sprawdź dla 6, 7, 9, 13, 15 zawodników.

Powyższa prawidłowość wynika z następującego faktu: schemat turnieju jest drzewem binarnym, a w takim drzewie liczba wierzchołków pośrednich jest o jeden mniejsza od liczby wierzchołków końcowych. Wierzchołki końcowe to zawodnicy przystępujący do turnieju, a wierzchołki pośrednie odpowiadają rozegranym meczom.



Rysunek 4.
Drzewo przykładowych rozgrywek w turnieju tenisowym (a) oraz drzewo znajdowania drugiego najlepszego zawodnika turnieju (b)

Wyłanianie drugiego najlepszego zawodnika turnieju

Bardzo ciekawy problem postawił około 1930 roku Hugo Steinhaus. Zastanawiał się on bowiem, jaka jest najmniejsza liczba meczów tenisowych do rozegrania w grupie zawodników, niezbędna do tego, aby wyłonić wśród nich najlepszego i drugiego najlepszego zawodnika. Wtedy, tak jak i dzisiaj, rozgrywano turnieje tenisowe systemem pucharowym. Zapewnia on, że zwycięzca finału jest najlepszym zawodnikiem, gdyż pokonał wszystkich uczestników turnieju: niektórych bezpośrednio – wygrywając z nimi w spotkaniach,

Pamiętaj. Wicemistrz wyłoniony systemem pucharowym na ogół nie jest drugą najlepszą drużyną (zawodnikiem) turnieju.

a niektórych pośrednio – pokonując ich zwycięzców. W takich turniejach drugą nagrodę otrzymuje zwykle zawodnik pokonany w finale. I tutaj Steinhaus miał słuszne wątpliwości, czy jest to właściwa decyzja, tzn., czy pokonany w finale jest drugim najlepszym zawodnikiem turnieju, czyli czy jest lepszy od wszystkich pozostałych zawodników z wyjątkiem zwycięzcy turnieju.

By się przekonać, że wątpliwości H. Steinhausa były rzeczywiście uzasadnione, spójrzmy na drzewo turnieju przedstawione na rysunku 4a). Zwycięzcą w tym turnieju jest Janek, który w finale pokonał Edka. Edkowi przyznano więc drugą nagrodę, chociaż wykazał, że jest lepszy jedynie od Bolka, Bartka i Wojtka (gdyż przegrał z Bartkiem). Nic nie wiemy, jak Edek by grał przeciwko zawodnikom z poddrzewa, z którego jako zwycięzca został wyłoniony Janek. Jak można naprawić ten błąd organizatorów rozgrywek tenisowych? Istnieje prosty sposób znalezienia drugiego najlepszego zawodnika turnieju – rozegrać jeszcze jedną pełną rundę z pominięciem zwycięzcy turnieju głównego. Wówczas, najlepszy i drugi najlepszy zawodnik zawodów zostaliby wyłonieni w $2n-3$ meczach. Hugo Steinhaus oczywiście znał to rozwiązanie, pytał więc o najmniejszą potrzebną liczbę meczów, i takiej odpowiedzi udzielił w 1932 inny polski matematyk Józef Schreier, chociaż jego dowód nie był w pełni poprawny i został skorygowany dopiero po 32 latach (w 1964 roku przez Sergeia Sergeyevicha Kislitsyna).

Jeśli chcemy, aby drugi najlepszy zawodnik nie musiał być wyłaniany w nowym pełnym turnieju, to musimy umieć skorzystać ze wszystkich wyników głównego turnieju. Posłużymy się drzewem turnieju z rysunku 4a). Zauważmy, że Edek jest oczywiście najlepszy wśród zawodników, którzy w drzewie rozgrywek znajdują się w wierzchołkach leżących poniżej najwyższego wierzchołka, który on zajmuje. Musimy więc jedynie porównać go z zawodnikami drugiego poddrzewa. Aby i w tym poddrzewie wykorzystać wyniki dotychczasowych meczów, eliminujemy z niego Janka — zwycięzcę turnieju i wstawiamy Edka na jego początkowe miejsce X. Spowoduje to, że Edek zostanie porównany z najlepszymi zawodnikami w drugim poddrzewie. Na rysunku 4b) oznaczyliśmy przerywaną linią mecze, które zostaną rozegrane w tej części turnieju – Jurek z Edkiem i zwycięzca tego meczu z Kubą, a więc dwa dodatkowe mecze.

Algorytm ten można, po zmianie słownictwa, zastosować do znajdowania największej i drugiej największej liczby w zbiorze danych.

Złożoność wyłaniania zwycięzcy i drugiego najlepszego zawodnika turnieju

Ile porównań jest wykonywanych w opisanym algorytmie znajdowania najlepszego i drugiego najlepszego zawodnika w turnieju? Najlepszy zawodnik jest wyłaniany w $n-1$ meczach, gdzie n jest liczbą wszystkich zawodników. Z kolei, aby wyłonić drugiego najlepszego zawodnika, trzeba rozegrać tyle meczów, ile jest poziomów w drzewie turnieju głównego (z wyjątkiem pierwszego poziomu). A zatem, jaka jest wysokość drzewa turnieju? Dla uproszczenia przyjmijmy, że drzewo jest **pełne**, tzn. każdy zawodnik ma parę, czyli w każdej rundzie turnieju gra parzysta liczba zawodników. Stąd wynika, że na najwyższym poziomie jest jeden zawodnik, na poziomie niższym – dwóch, na kolejnym – czterech itd. Czyli liczba zawodników rozpoczynających turniej jest potęgą liczby 2, zatem $n = 2^k$, gdzie k jest liczbą poziomów drzewa – oznaczmy ją przez $\log_2 n$. Algorytm wykonuje więc $(n-1) + (\log_2 n - 1) = n + \log_2 n - 2$ porównań. Jeśli n nie jest potęgą liczby 2, to na ogół w turnieju niektórzy zawodnicy otrzymują wolną kartę, a podana liczba jest oszacowaniem z góry liczby rozegranych meczów.

Porównaj wartości dwóch wyrażeń: $2n-3$ oraz $n + \log_2 n - 2$, odpowiadających liczbie porównań wykonywanych w dwóch omówionych wyżej algorytmach znajdowania najlepszego i drugiego najlepszego zawodnika turnieju. Dla ułatwienia obliczeń przyjmij, że n jest potęgą liczby 2.

Przedstawiony powyżej algorytm znajdowania najlepszego i drugiego najlepszego zawodnika turnieju jest optymalny, tzn. najszybszy w sensie liczby rozegranych meczów (porównań).

Na naszym podium zwycięzców turnieju tenisowego brakuje jeszcze trzeciego najlepszego zawodnika, czyli kogoś, kto jest lepszy od wszystkich pozostałych zawodników z wyjątkiem już wyłonionych – najlepszego i drugiego najlepszego. Znalezienie go bardzo przypomina wyłanianie drugiego najlepszego zawodnika. Przy-

puśćmy, że w naszym przykładowym turnieju, drugie miejsce zajął Kuba. W jaki sposób należy zorganizować dogrywkę, by wyłonić trzeciego najlepszego zawodnika turnieju. A jeśli drugie miejsce zajął jednak Edek – jak należy postępować w tym przypadku? Sformułuj ogólną zasadę i oblicz, ile meczów należy rozegrać, by wyłonić zawodnika zajmującego trzecie miejsce?

To postępowanie można kontynuować wyznaczając czwartego, piątego itd. zawodnika turnieju. Ostatecznie otrzymamy pełne uporządkowanie wszystkich zawodników biorących udział w turnieju. Taka metoda nazywa się **porządkowaniem na drzewie** i może być stosowana również do porządkowania liczb.

3 JEDNOCZESNE ZNAJDOWANIE NAJMNIEJSZEGO I NAJWIĘKSZEGO ELEMENTU

Jedną z miar, określającą, jak bardzo są porozrzucane wartości obserwowanej w doświadczeniu wielkości, jest **rozpiętość** zbioru, czyli różnica między największą (w skrócie, maksimum) a najmniejszą wartością elementu (w skrócie, minimum) w zbiorze. Im większa jest rozpiętość, tym większy jest rozrzut wartości elementów zbioru. Interesujące jest więc jednoczesne znalezienie najmniejszej i największej wartości w zbiorze liczb.

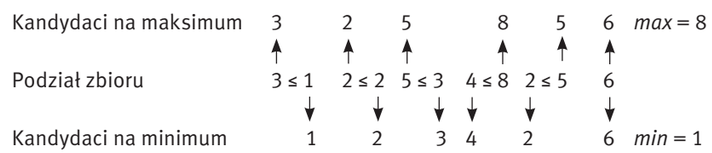
Rozwiązanie naiwne

Na podstawie dotychczasowych rozważań, dotyczących wyznaczania najmniejszej i największej wartości w zbiorze liczb, zapewne łatwo podasz algorytm znajdowania jednocześnie obu tych elementów w zbiorze. Ile należy w tym celu wykonać porównań?

Odpowiedź na to pytanie ilustruje częste podejście, stosowane w matematyce i informatyce, które polega na tym, że w rozwiązaniu nowego problemu korzystamy ze znanej już metody. Stosujemy więc najpierw algorytm **Min** do całego zbioru, a później algorytm **Max** do zbioru z usuniętym minimum. W takim algorytmie jednoczesnego wyznaczania minimum i maksimum w ciągu złożonym z n liczb jest wykonywanych $(n - 1) + (n - 2) = 2n - 3$ porównań. Ale czy rzeczywiście te dwie wielkości są wyznaczane jednocześnie?

Rozwiązanie metodą dziel i zwyciężaj

Postaramy się znacznie przyspieszyć to postępowanie, a będzie to polegało na rzeczywiście jednoczesnym szukaniu najmniejszego i największego elementu w całym zbiorze, jak również wykorzystaniu poznanego algorytmu znajdowania tych elementów w pewnych podzbiorach rozważanego zbioru. W tym celu rozważmy ponownie podstawową operację – porównanie elementów – i zauważmy, że jeśli dwie liczby x i y spełniają nierówność $x \le y$, to x jest kandydatem na najmniejszą liczbę w zbiorze, a y jest kandydatem na największą liczbę w zbiorze. (Jeśli prawdziwa jest nierówność odwrotna, to wnioskujemy odwrotnie.) A zatem, porównując elementy parami, można podzielić dany zbiór elementów na dwa podzbiory, kandydatów na minimum i kandydatów na maksimum, i w tych zbiorach – które są niemal o połowę mniejsze niż oryginalny zbiór! – szukać odpowiednio minimum i maksimum. Pewnym problemem jest to, co zrobić z ostatnim elementem ciągu, gdy zbiór ma nieparzystą liczbę elementów. W tym przypadku możemy dodać ten element do jednego i do drugiego podzbioru kandydatów. Postępowanie to jest zilustrowane przykładem na rysunku 5.



Rysunek 5.
Przykład postępowania podczas jednoczesnego znajdowania minimum i maksimum w ciągu liczb

Zapiszmy opisane postępowanie w postaci algorytmu poprzedzając go specyfikacją.

Algorytm Min-i-Max – jednoczesne znajdowanie największego i najmniejszego elementu w zbiorze

Dane: Liczba naturalna n i zbiór n liczb dany w postaci ciągu $x_1, x_2, \ldots, x_n$.
Wynik: Najmniejsza liczba *min* i największa liczba *max* wśród liczb $x_1, x_2, \ldots, x_n$.
Krok 1. {Podział zbioru danych na dwa podzbiory: M – zbiór kandydatów na minimum i N – zbiór kandydatów na maksimum. Na początku te zbiory są puste.}
Jeśli n jest liczbą parzystą, to dla $i = 1, 3, \ldots, n - 1$, a jeśli n jest liczbą nieparzystą, to dla $i = 1, 3, \ldots, n - 2$ wykonaj:
jeśli $x_i \le x_{i+1}$, to dołącz x_i do M, a x_{i+1} do N,
a w przeciwnym razie dołącz x_i do N, a x_{i+1} do M.
Jeśli n jest liczbą nieparzystą, to dołącz x_n do obu zbiorów M i N.
Krok 2. Znajdź *min* w zbiorze M, stosując algorytm **Min**.
Krok 3. Znajdź *max* w zbiorze N, stosując algorytm **Max**.

Ten algorytm jest przykładem metody, leżącej u podstaw bardzo wielu efektywnych algorytmów. Można w nim wyróżnić dwa etapy:

- podziału danych na dwa podzbiory równoliczne (krok 1);
- zastosowania znanych już algorytmów **Min** i **Max** do utworzonych podzbiorów danych (kroki 2 i 3).

Jest to przykład zasady (metody) rozwiązywania problemów. która wielokrotnie pojawia się na zajęciach z algorytmiki i programowania. Nosi ona nazwę **dziel i zwyciężaj** i jest jedną z najefektywniejszych metod algorytmicznych w informatyce (patrz rozdz. 7). **Dziel** – odnosi się do podziału zbioru danych na podzbiory, zwykle o jednakowej liczbie elementów, do których następnie są stosowane odpowiednie algorytmy. **Zwyciężstwo** – to efekt końcowy, czyli efektywne rozwiązanie rozważanego problemu.

Pracochłonność jednoczesnego znajdowania minimum i maksimum

Obliczmy, ile porównań między elementami danych jest wykonywanych w algorytmie **Min-i-Max**. Rozważmy najpierw przypadek, gdy n jest liczbą parzystą. W takim przypadku, w kroku podziału jest wykonywanych $n/2$ porównań, a znalezienie *min* oraz znalezienie *max* wymaga każde $n/2 - 1$ porównań. Razem jest to $3n/2 - 2$ porównania. Gdy n jest liczbą nieparzystą, to otrzymujemy liczbę porównań $\lceil 3n/2 \rceil - 2$, gdzie $\lceil x \rceil$ oznacza tzw. **powałę liczby**[16], czyli najmniejszą liczbę całkowitą k spełniającą nierówność $x \le k$.

A zatem, w algorytmie **Min-i-Max** wykonuje się $\lceil 3n/2 \rceil - 2$ porównania, czyli ok. $n/2$ mniej porównań niż w algorytmie naiwnym, podanym na początku tego punktu.

Zastosowana tutaj zasada **dziel i zwyciężaj** jest na ogół stosowana w sposób **rekurencyjny** – problem jest dzielony na podproblemy, te są ponownie dzielone na podproblemy, i tak dalej, aż do otrzymania podproblemów, dla których rozwiązanie można łatwo wskazać, np. gdy liczba danych w podproblemie wynosi 1 lub 2. Tę ogólną metodę dziel i zwyciężaj ilustrujemy dalej na przykładach poszukiwania elementu w zbiorze uporządkowanym (punkt 6.2) oraz sortowania przez scalanie.

4 PROBLEM PORZĄDKOWANIA – PORZĄDKOWANIE PRZEZ WYBÓR

Porządkowanie, nazywane również często **sortowaniem** (będziemy tych terminów używali zamiennie) ma olbrzymie znaczenie niemal w każdej działalności człowieka. Jeśli elementy w zbiorze są uporządkowane

[16] Funkcja **powała** (i towarzysząca jej funkcja **podłoga**) odgrywają ważną rolę w rozważaniach informatycznych.

zgodnie z jakąś regułą (np. książki lub ich karty katalogowe według liter alfabetu, słowa w encyklopedii, daty, numery telefonów według nazwisk właścicieli), to wykonywanie wielu operacji na tym zbiorze staje się znacznie łatwiejsze i szybsze. Między innymi dotyczy to operacji:

- sprawdzenia, czy dany element, czyli element o ustalonej wartości cechy, według której zbiór został uporządkowany, znajduje się w zbiorze;
- znalezienia elementu w zbiorze, jeśli w nim jest;
- dołączenia nowego elementu w odpowiednie miejsce, aby zbiór pozostał nadal uporządkowany.

Komputery w dużym stopniu zawdzięczają swoją szybkość temu, że działają na uporządkowanych informacjach. To samo odnosi się do nas – ludzi, gdy posługujemy się nimi, informacjami i komputerami. Jeśli chcemy na przykład sprawdzić, czy w jakimś katalogu dyskowym znajduje się plik o podanej nazwie, rozszerzeniu, czasie utworzenia lub rozmiarze, to najpierw odpowiednio porządkujemy listę plików (np. w programie Eksplorator Windows) i wtedy na ogół znajdujemy odpowiedź natychmiast. Porządkowanie jest również podstawową operacją wykonywaną na dużych zbiorach informacji, np. w bazach danych.

Często porządkujemy różne elementy lub wykonujemy powyższe operacje na uporządkowanych zbiorach nie korzystając z komputera – w tym również mogą nam pomóc metody porządkowania i algorytmy działające na uporządkowanych zbiorach omówione na zajęciach komputerowych.

4.1 PROBLEM PORZĄDKOWANIA

Na tych zajęciach będziemy zajmować się głównie porządkowaniem liczb, chociaż wiele praktycznych problemów dotyczy porządkowania innych obiektów przechowywanych w komputerze. Przyjmijmy więc następującą specyfikację tego problemu.

Problem porządkowania (sortowania)
Dane: Liczba naturalna n i ciąg n liczb $x_1, x_2, \ldots, x_n$.
Wynik: Uporządkowanie tego ciągu liczb od najmniejszej do największej.

Z założenia, że porządkujemy tylko liczby względem ich wartości wynika, że interesują nas algorytmy, w których główną operacją jest porównanie, wykonywane między elementami danych.

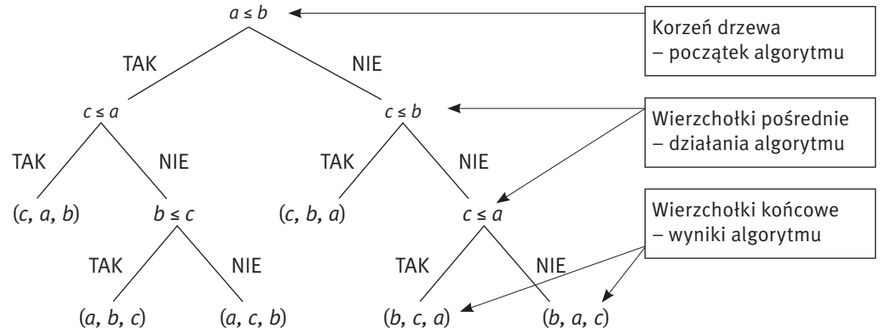
W ogólnym sformułowaniu problemu porządkowania nie czynimy żadnych założeń dotyczących elementów, które porządkujemy, ważna jest jedynie relacja między wartościami liczb. W ogólnym przypadku na ogół zakładamy, że porządkowane liczby są całkowite dodatnie. W wielu sytuacjach praktycznych porządkowane są najróżniejsze obiekty, np. słowa (przy tworzeniu słownika), adresy (do książki telefonicznej) czy bardzo złożone zestawy danych, jak np. informacje o koncie bankowym i jego właścicielu. Szczególnym przypadkiem porządkowania liczb jest sytuacja, w której liczby mają niewielkie wartości – specjalny algorytm porządkowania takich liczb jest zamieszczony w rozdziale 5.

4.2 PORZĄDKOWANIE KILKU ELEMENTÓW

Jeśli liczba elementów w ciągu jest mała, np. $n = 2, 3, 4$, to łatwo można podać algorytmy, w których jest wykonywana możliwie najmniejsza liczba porównań.

Mając dwa elementy, wystarczy jedno porównanie, by ustawić je w porządku. Nieco więcej zachodu wymaga porządkowanie trzech liczb. Mając do uporządkowania trzy liczby, możemy postąpić następująco: najpierw ustawić a i b w odpowiedniej kolejności, a potem wstawić c w odpowiednie miejsce względem a i b. W pierwszym etapie wykonujemy jedno porównanie, a w drugim? Załóżmy, że po pierwszym porównaniu, czyli $a \le b$, otrzymujemy uporządkowanie (a,b). Wtedy, jeśli $c \le a$, to (c,a,b) jest szukanym uporządkowaniem. W przeciwnym razie musimy jeszcze sprawdzić, czy $c \le b$. Jeśli tak, to otrzymujemy uporządkowanie (a,c,b), a w przeciwnym razie – (a,b,c). Podobnie postępujemy, gdy po pierwszym porównaniu kolejność liczb a i b jest (b,a).

Wszystkie te przypadki można zapisać w postaci **drzewa algorytmu**, przedstawionego na rysunku 6. Drzewo algorytmu jest pewnego rodzaju „schematem blokowym" algorytmu – w sposób graficzny ilustruje przebieg działania algorytmu dla dowolnych danych. Ta reprezentacja algorytmu jest bardzo przydatna do śledzenia, jak algorytm działa dla poszczególnych danych. Wygodna jest również do analizy pracochłonności algorytmu, czyli do wyznaczania, ile porównań wykonuje algorytm dla poszczególnych danych.



Rysunek 6.
Drzewo algorytmu porządkującego trzy liczby

Wykonywanie algorytmu, zapisanego w postaci drzewa, rozpoczyna się w **korzeniu** tego drzewa, przechodzi przez **wierzchołki pośrednie**, odpowiadające operacjom wykonywanym w algorytmie, i kończy się w **wierzchołku końcowym**. Wierzchołki końcowe drzewa algorytmu zawierają wszystkie możliwe rozwiązania – w naszym przykładzie są to wszystkie możliwe uporządkowania trzech elementów. Takich uporządkowań jest 6.

Może się zdarzyć, że porządkowane liczby są uporządkowane, mogą się również powtarzać – algorytm porządkowania powinien dawać poprawną odpowiedź również w tych przypadkach, czyli działać jak „bezmyślny automat" do wykonywania obliczeń. W tym konkretnym przypadku nie powinien „zauważyć", że liczby są już uporządkowane i przystąpić do działania, jak dla dowolnej innej trójki liczb. Jest to potwierdzenie **uniwersalności** algorytmu porządkowania trzech liczb – może on być stosowany do dowolnej trójki liczb, żadnej specjalnie nie wyróżniając.

Algorytm porządkowania trzech liczb może być łatwo rozszerzony do algorytmu porządkowania czterech liczb przez wstawienie czwartej liczby do już uporządkowanego ciągu trzech liczb w każdym wierzchołku wiszącym drzewa – wymaga to wykonania dodatkowo dwóch porównań.

Niestety, w podobny sposób nie da się otrzymać najlepszego algorytmu porządkowania pięciu liczb – odsyłamy do książki [7], gdzie szczegółowo opisano taki algorytm, który wykonuje 7 porównań w najgorszym przypadku.

Wspomnieliśmy, że drzewo algorytmu ułatwia analizę pracochłonności algorytmu. Na podstawie drzewa z rysunku 6 wynika, że przy porządkowaniu trzech liczb trzeba wykonać co najwyżej 3 porównania – jest to bowiem najdłuższa (w sensie liczby wierzchołków pośrednich, które odpowiadają działaniom w algorytmie) droga z korzenia do wierzchołka końcowego. Długość takiej drogi nazywamy **wysokością drzewa**.

4.3 PORZĄDKOWANIE PRZEZ WYBÓR

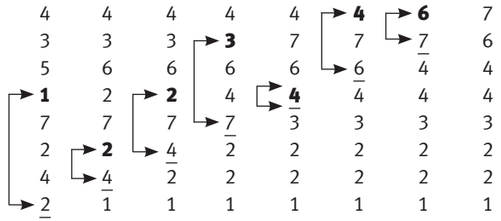
Zajmiemy się teraz porządkowaniem ciągów, które mogą zawierać dowolną liczbę elementów. Wykorzystamy w tym celu jeden z poznanych wcześniej algorytmów. O innych algorytmach porządkowania wspominamy w dalszej części wykładu.

Jeden z najprostszych algorytmów porządkowania można wyprowadzić korzystając z tego, co już poznaliśmy w poprzednich punktach. Zauważmy, że jeśli mamy ustawić elementy w kolejności od najmniejszego

do największego, to najmniejszy element w zbiorze powinien się znaleźć na początku tworzonego ciągu, za nim powinien być umieszczony najmniejszy element w zbiorze pozostałym po usunięciu najmniejszego elementu itd. Taki algorytm jest więc **iteracją** znanego algorytmu znajdowania **Min** w ciągu i nosi nazwę **algorytmu porządkowania przez wybór**.

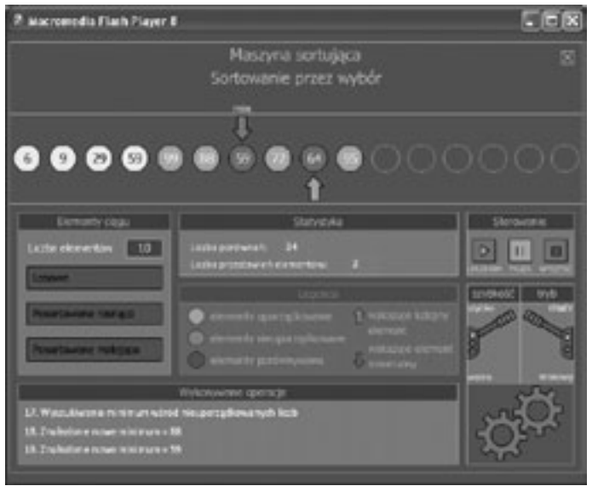
Demonstracja działania porządkowania przez wybór

Aby zilustrować działanie tego algorytmu, załóżmy, że ciąg elementów, który mamy uporządkować, jest zapisany w kolumnie (patrz rys. 7). Chcemy ponadto, aby wynik, czyli ciąg uporządkowany, znalazł się w tym samym ciągu – o takim algorytmie mówimy, że działa ***in situ***, czyli „w miejscu". W tym celu wystarczy znaleziony najmniejszy element w ciągu zamienić miejscami z pierwszym elementem tego ciągu. Rysunek 7 ilustruje kolejne kroki działania algorytmu porządkowania przez wybór.



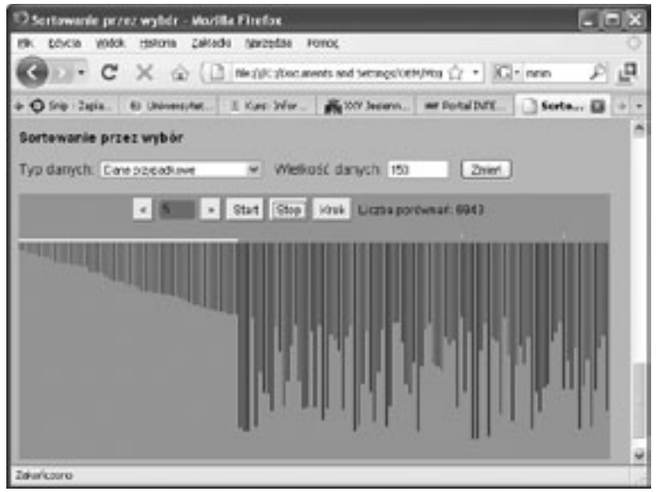
Rysunek 7.
Ilustracja działania algorytmu porządkowania przez wybór. W każdej kolumnie, pogrubiony został najmniejszy element w podciągu od góry do kreski, a klamra wskazuje zamianę elementów miejscami

Zanim podamy szczegółowy opis tego algorytmu porządkowania, przyjrzyj się pełnej demonstracji jego działania w programie **Maszyna sortująca**, który był wykorzystany do demonstracji działania algorytmu znajdującego najmniejszy element w ciągu.



Rysunek 8.
Demonstracja działania algorytmu porządkowania przez wybór w programie **Maszyna sortująca** – cztery pierwsze elementy znajdują się już na swoim miejscu w ciągu uporządkowanym i szukany jest najmniejszy element w pozostałej części ciągu, by przenieść go na miejsce piąte

Polecamy również inny program **Sortowanie**, który służy do demonstracji działania oraz porównywania między sobą wielu algorytmów porządkujących. Ten program jest również załączony do materiałów do tych zajęć i może być wykorzystany w celach edukacyjnych.



Rysunek 9.
Demonstracja działania algorytmu porządkowania przez wybór w programie **Sortowanie**

Opis algorytmu porządkowania przez wybór

Przedstawmy teraz ścisły opis algorytmu porządkowania przez wybór, znanego jako **SelectionSort**.

Algorytm porządkowania przez wybór – SelectionSort

Dane: Liczba naturalna n i ciąg n liczb $x_1, x_2, \ldots, x_n$.
Wynik: Uporządkowanie danego ciągu liczb od najmniejszej do największej, czyli ciąg wynikowy spełnia nierówności $x_1 \le x_2 \le \ldots \le x_n$. (***Uwaga.*** Elementy ciągu danego i wynikowego oznaczamy tak samo, gdyż porządkowanie odbywa się „w tym samym miejscu".)
Krok 1. Dla $i = 1, 2, \ldots, n-1$ wykonaj kroki 2 i 3, a następnie zakończ algorytm.
Krok 2. Znajdź k takie, że x_k jest najmniejszym elementem w ciągu $x_i, \ldots, x_n$.
Krok 3. Zamień miejscami elementy x_i oraz x_k.

Uwaga. Zauważ, że liczba k znaleziona w kroku 2 może być równa i w tym kroku, a zatem w kroku 3 ten sam element jest zamieniany miejscami ze sobą. Z taką sytuacja mamy do czynienia w piątej iteracji przykładowej demonstracji działania algorytmu, przedstawionej na rysunku 7, gdzie element 4 jest zamieniany ze sobą.

Złożoność algorytmu SelectionSort

Obliczmy teraz, ile porównań i zamian elementów, w zależności od liczby elementów w ciągu n, jest wykonywanych w algorytmie **SelectionSort** oraz w jego komputerowych implementacjach. W tym celu wystarczy zauważyć, o czym pisaliśmy już powyżej, że algorytm jest iteracją algorytmu znajdowania najmniejszego elementu w ciągu, a ciąg, w którym szukamy najmniejszego elementu, jest w kolejnych iteracjach coraz krótszy. Liczba przestawień elementów jest równa liczbie iteracji, gdyż elementy są przestawiane jedynie na końcu każdej iteracji, których jest $n - 1$, a więc wynosi $n - 1$. Jeśli zaś chodzi o liczbę porównań, to wiemy już, że

algorytm znajdowania minimum w ciągu wykonuje o jedno porównanie mniej niż jest elementów w ciągu. Ponieważ w każdym kroku liczba elementów w przeszukiwanym podciągu jest o jeden mniejsza, cały algorytm porządkowania przez wybór, dla ciągu danych złożonego na początku z n elementów wykonuje liczbę porównań równą:

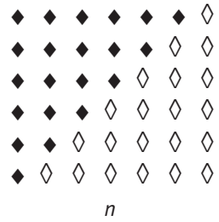
$$(n-1) + (n-2) + (n-3) + \ldots + 3 + 2 + 1$$

Wartość tej sumy można obliczyć wieloma sposobami. Przedstawimy dwa z nich – są one ciekawe przez swoją prostotę. Inny sposób, w którym korzysta się ze wzoru na sumę postępu arytmetycznego, pomijamy tutaj, jako oczywisty dla tych, którzy wiedzą, co to jest postęp arytmetyczny – nasze sposoby tego nie wymagają.

Dowód geometryczny

Kolejne liczby naturalne od 1 do $n-1$ można przedstawić w postaci trójkąta, którego wiersz i, licząc od dołu, zawiera i diamentów (na rys. 10 są to czarne diamenty). Dwa takie same trójkąty pasują do siebie i tworzą prostokąt zawierający $(n-1)n$ diamentów, zatem wartość powyższej sumy jest połową liczby wszystkich diamentów w całym prostokącie, czyli jest równa:

$$\frac{(n-1)n}{2}$$



Ten geometryczny dowód, zamieszczony obok, znali już starożytni Grecy. Na podstawie geometrycznej interpretacji, wartość sumy kolejnych liczb naturalnych nazywali **liczbami trójkątnymi.**

Rysunek 10.
Ilustracja geometrycznego wyznaczania wartości sumy kolejnych liczb naturalnych od 1 do $n-1$

Spostrzeżenie nudzącego się geniusza

Geniusz – gen i już.
Hugo Steinhaus

Anegdota mówi, że nauczyciel matematyki w klasie, do której uczęszczał młody Carl Friedrich Gauss (1777-1855), jeden z największych matematyków w historii, by zająć przez dłuższy czas swoich uczniów żmudnymi rachunkami, dał im do obliczenia wartość sumy stu początkowych liczb naturalnych, czyli 1 + 2 + 3 + ... + 98 + 99 + 100. Nie cieszył się jednak zbyt długo spokojem, po chwili otrzymał bowiem gotową odpowiedź od Carla, który szybko zauważył, że suma liczb w skrajnych parach, 1+100, 2+99, 3+98 itd. aż do 50+51 jest taka sama, a takich par jest połowa ze stu, czyli z liczby wszystkich elementów. Stąd natychmiast otrzymał powyższy wzór.

Otrzymaliśmy wzór na liczbę porównań w algorytmie porządkowania przez wybór, który zależy tylko od liczby porządkowanych elementów. Można uznać za słabą stronę tego algorytmu, że wykonuje on taką samą liczbę działań (porównań i przestawień) na ciągach o tej samej długości, bez względu na stopień ich uporządkowania. W następnym podpunkcie wspominamy o metodzie porządkowania, która jest „bardzo czuła" na stopień uporządkowania elementów w porządkowanym ciągu.

4.4 INNE ALGORYTMY PORZĄDKOWANIA

Znanych jest bardzo wiele algorytmów porządkowania liczb i innych obiektów przechowywanych w komputerze, jak słów, dat (to nie są liczby tylko układy liczb), rekordów (czyli układów danych różnych typów). Temu zagadnieniu poświęcono wiele opasłych książek, np. Donald Knuth napisał na ten temat ponad 1000 stron

jeszcze w latach 60. XX wieku (patrz [4]). Na tych zajęciach problem sortowania pojawia się jeszcze kilka razy. W rozdziale 5 opisujemy algorytm porządkowania małych liczb, a w rozdziale 7 przedstawiamy algorytm sortowania przez scalanie bazując na metodzie dziel i zwyciężaj, będący jednym z najszybszych metod sortowania.

5 PORZĄDKOWANIE PRZEZ ZLICZANIE

W algorytmie porządkowania, który opisaliśmy w poprzednim rozdziale, nie poczyniliśmy żadnych założeń dotyczących elementów, które porządkujemy. Służy on do porządkowania dowolnych liczb (przyjęliśmy, że porządkowane liczby są całkowite), wykonując porównania między tymi liczbami – ważne jest tylko, która z dwóch liczby jest większa, a która mniejsza. W rzeczywistych sytuacjach porządkowane mogą być liczby szczególnej postaci, a ogólnie – porządkowane są najróżniejsze obiekty, np. słowa (do słownika), adresy (do książki telefonicznej) czy bardzo złożone zestawy danych, jak np. informacje o koncie bankowym i jego właścicielu.

Szczególnym przypadkiem porządkowania liczb jest sytuacja, w której liczby są niewielkie, jest ich niewiele różnych i należą do niewielkiego przedziału. Takie własności ma na przykład ciąg stopni wierzchołków w grafie (będzie o tym mowa na innych zajęciach). Na potrzeby tego rozdziału zmieńmy więc nieco opis problemu porządkowania, jak następuje:

Problem porządkowania niewielkich liczb

Dane: Liczba naturalna n i ciąg n liczb całkowitych $x_1, x_2, \ldots, x_n$, należących do przedziału $[1..M]$, gdzie M jest porównywalne co do wartości z n (na ogół przyjmuje się, że $M < n$).
Wynik: Uporządkowanie tego ciągu liczb od najmniejszej do największej.

Aby uporządkować liczby, które spełniają warunek z opisu danych, wystarczy policzyć, ile wśród danych jest liczb równych 1, liczby równych 2 itd. A następnie po policzeniu, ile jest konkretnych liczb, ustawić je w kolejności: najpierw elementy równe 1, później elementy równe 2 itd. Taki algorytm nazywa się **porządkowaniem przez zliczanie** (ang. *counting sort*). Załóżmy, że c_i oznacza liczbę elementów porządkowanego ciągu równych i – c_i można więc uznać za **licznik** elementów równych i. Ten algorytm składa się z dwóch kroków (krok 1 jest tylko przygotowaniem do porządkowania):

Algorytm. Porządkowanie przez zliczanie – CountingSort

Krok 1. Dla $i = 1, 2, \ldots, M$, przyjmij $c_i = 0$. {Zerowanie liczników elementów.}
Krok 2. Dla $i = 1, 2, \ldots, n$, zwiększ c_k o 1, gdzie $k = x_i$.
Krok 3. Zacznij od pierwszej pozycji w ciągu x i
dla $i = 1, 2, \ldots, M$, na kolejnych c_i pozycjach w ciągu x ustaw element i.

Z postaci algorytmu wynika, że jeśli porządkowane elementy przyjmują nie więcej niż M wartości i M nie jest większe od liczby porządkowanych elementów ($M < n$), to algorytm porządkowania przez zliczanie jest algorytmem o złożoności liniowej względem liczby elementów w porządkowanym ciągu. A zatem jest znacznie szybszy niż algorytm porządkowania przez wybór, przedstawiony w rozdziale 4.

6 POSZUKIWANIE INFORMACJI W ZBIORZE

Komputery bardzo ułatwiają szybkie poszukiwanie informacji umieszczonych na płytach CD i na serwerach sieci Internet. Jest to zasługą nie tylko szybkości działania procesorów, ale też dobrej organizacji pracy, dzięki czemu pozostaje im... niewiele do roboty. Przypomnijmy poczynione założenia, dotyczące przeszukiwanych

zbiorów. Zbiór może zawierać elementy powtarzające się (czyli o takich samych wartościach) i w obliczeniach jest dany w postaci ciągu, który na ogół jest poprzedzony liczbą jego elementów. Ciąg ten może być uporządkowany, np. od najmniejszego do największego elementu, lub nieuporządkowany.

W drugim rozdziale zajmowaliśmy się znajdowaniem szczególnych elementów w zbiorze nieuporządkowanym, najmniejszego i największego. Teraz będziemy rozważać problem poszukiwania w ogólniejszej postaci.

Problem poszukiwania elementu w zbiorze

Dane: Zbiór elementów w postaci ciągu *n* liczb $x_1, x_2, ..., x_n$. Wyróżniony element *y*.
Wynik: Jeśli *y* należy do tego zbioru, to podaj jego miejsce (indeks) w ciągu, a w przeciwnym razie – sygnalizuj brak takiego elementu w zbiorze.

Problem poszukiwania ma bardzo wiele zastosowań i jest rozwiązywany przez komputer na przykład wtedy, gdy w jakimś ustalonym zbiorze informacji staramy się znaleźć konkretną informację. Rozważana przez nas wersja tego problemu jest bardzo prosta, na ogół bowiem zbiory i ich elementy mają bardzo złożoną postać, nie są ograniczone tylko do pojedynczych liczb. Przedstawione metody mogą być jednak uogólnione na bardziej złożone sytuacje problemowe.

W następnych podpunktach, najpierw rozwiązujemy problem poszukiwania w dowolnym zbiorze elementów, a później – w zbiorze uporządkowanym.

6.1 POSZUKIWANIE ELEMENTU W ZBIORZE NIEUPORZĄDKOWANYM

Jeśli nic nie wiemy o elementach w ciągu danych $x_1, x_2, ..., x_n$, to aby stwierdzić, czy wśród nich jest element równy danemu *y*, musimy sprawdzić każdy z elementów tego ciągu, gdyż element *y* może się znajdować w dowolnym miejscu ciągu, a w szczególnym przypadku może go tam nie być. W takim przypadku stosujemy **przeszukiwanie** (lub **poszukiwanie**) **liniowe**, które stosowaliśmy w rozdziale 1 do znajdowania w ciągu elementu najmniejszego lub największego. Na ogół takie przeszukiwanie odbywa się od lewej do prawej, czyli od początku do końca ciągu. Można je opisać następująco:

Algorytm poszukiwania liniowego

Dane: Zbiór elementów w postaci ciągu *n* liczb $x_1, x_2, ..., x_n$. Wyróżniony element *y*.
Wynik: Jeśli *y* należy do tego zbioru, to podaj jego miejsce (indeks) w ciągu, a w przeciwnym razie – sygnalizuj brak takiego elementu w zbiorze.
Krok 1. Dla $i = 1, 2, ..., n$, jeśli $x_i = y$, to przejdź do kroku 3.
Krok 2. Komunikat: W ciągu danych nie ma elementu równego *y*. Zakończ algorytm.
Krok 3. Element równy *y* znajduje się na miejscu *i* w ciągu danych. Zakończ algorytm.

Jeśli element *y* znajduje się w przeszukiwanym ciągu, to algorytm kończy działanie po natknięciu się na niego po raz pierwszy, a jeśli nie ma go w tym ciągu to kończy się po dojściu do końca ciągu. W obu przypadkach liczba działań jest proporcjonalna do liczby elementów w ciągu. W pierwszym przypadku najwięcej operacji jest wykonywanych wówczas, gdy poszukiwany element jest na końcu ciągu.

Algorytm poszukiwania może wykonywać różną liczbę iteracji nawet dla ustalonego ciągu danych, zależy ona bowiem od wartości poszukiwanego elementu *y*. Mamy więc okazję, by posłużyć się instrukcją iteracyjną, w której liczba iteracji może zależeć od spełnienia podanego warunku. W przypadku algorytmu poszukiwania, kończy on działanie w jednej z dwóch sytuacji: albo został znaleziony poszukiwany element *y*, albo został przejrzany cały ciąg i nie znaleziono tego elementu. Musimy uwzględnić jedną i drugą ewentualność. Umożliwia nam to następująca funkcja – przyjmujemy, że wartością funkcji jest indeks znalezionego elementu, jeśli znajduje się on w ciągu, lub – 1, jeśli element o wartości *y* nie istnieje w ciągu:


```
function PrzeszukiwanieLiniowe(n,y:integer; x:tablicax):integer;
 {Wartoscia funkcji jest indeks elementu tablicy
  rownego y, lub -1, jesli brak takiego elementu w ciągu.}
 var i:integer;
begin
 i:=1;
 while (x[i]<>y) and (i<n) do i:=i+1;
 if x[i]=y then PrzeszukiwanieLiniowe:=i
 else PrzeszukiwanieLiniowe:=-1
end; {PrzeszukiwanieLiniowe}
```

Warunkowa instrukcja iteracyjna: `while (x[i]<>y) and (i<n) do i:=i+1;`

jest wykonywana tak długo, jak długo spełniony jest warunek: `(x[i]<>y) and (i<n)`

Czyli, gdy badany element ciągu jest różny od *y* oraz nie został jeszcze osiągnięty koniec ciągu. Wtedy zwiększany jest bieżący indeks elementów ciągu. Po tej instrukcji następuje złożona instrukcja warunkowa:

```
if x[i]=y then PrzeszukiwanieLiniowe:=i
else PrzeszukiwanieLiniowe:=-1
```

której zadaniem jest zbadanie, z jakiego powodu nastąpiło zakończenie iteracji. Jeśli `x[i]=y`, to w ciągu został znaleziony element równy `y`, a w przeciwnym razie (`else`) nie ma w ciągu elementu o wartości `y`.

Przeszukiwanie liniowe z wartownikiem

Ciekawe własności ma niewielka modyfikacja powyższego algorytmu, wykorzystująca specjalny element, umieszczony na końcu ciągu, zwany **wartownikiem**. Rolą wartownika jest „pilnowanie", by proces przeszukiwania nie wyszedł poza ciąg. Jak wiemy, gdy ciąg zawiera element o wartości *y*, to przeszukiwanie kończy się na tym elemencie. Aby mieć pewność, że przeszukiwanie zawsze zakończy się na elemencie o wartości *y*, dołączamy na końcu ciągu element – wartownika – właśnie o wartości *y*. W efekcie, przeszukiwanie zawsze zakończy się znalezieniem elementu o wartości *y*, należy jedynie sprawdzić, czy znaleziony element *y* znajduje się na dołączonej pozycji zbioru, czy też wystąpił wcześniej. W pierwszym przypadku, badany zbiór nie zawiera elementu równego *y*, a w drugim – *y* należy do zbioru. Widać stąd, że dołączony do zbioru element odgrywa rolę jego wartownika – nie musimy bowiem sprawdzać, czy przeglądanie objęło cały zbiór czy nie – zawsze zatrzyma się ono na szukanym elemencie, którym może być dołączony właśnie element. A oto fragment przeszukiwania z wartownikiem:

```
begin
 i:=1;
 x[n+1]:=y;
 while x[i]<>y do i:=i+1;
 if i<=n then PrzeszukiwanieLinioweWartownik:=i
 else PrzeszukiwanieLinioweWartownik:=-1
end;
```

6.2 POSZUKIWANIE ELEMENTU W ZBIORZE UPORZĄDKOWANYM

W tym podrozdziale zakładać będziemy, że poszukiwania elementów (informacji) są prowadzone w uporządkowanych zbiorach (ciągach) elementów – chcemy albo znaleźć element, albo umieścić go w takim zbiorze z zachowaniem uporządkowania.

Porządek w informacjach

Zbiory mogą mieć różną strukturę – mogą to być książki w bibliotece, hasła w encyklopedii, liczba w ustalonym przedziale lub numery w książce telefonicznej. Te przykłady są bliskie codziennym sytuacjom, w których należy odszukać pewną informację i zapewne stosowane przez Was w tych przypadkach metody są podobne do opisanych tutaj. Trzeba wyraźnie podkreślić, że:

integralną częścią informacji jest jej uporządkowanie,

gdyż w przeciwnym razie... nie jest to informacja. To stwierdzenie nie jest naukowym określeniem informacji[17], ale odnosi się do informacji w potocznym znaczeniu, do informacji, które nas zalewają i nieraz przytłaczają, do informacji, wśród których mamy odnaleźć tę nam potrzebną lub „zrobić wśród nich porządek". Podstawowym przygotowaniem do życia w erze i społeczeństwie informacji jest bowiem nabycie umiejętności takiego postępowania z informacją (uporządkowaną oczywiście), by w posługiwaniu się nią korzystać z jej uporządkowania, nie psuć go i ewentualnie naprawiać, gdy ulega zniszczeniu lub gdy informacja się rozrasta.

Wykonaj teraz ćwiczenie, które zapewne przeprowadziłeś już nieraz w swoim życiu, nie zdając sobie nawet z tego sprawy. Weź do ręki jedną z książek: słownik ortograficzny, słownik polsko-angielski lub książkę telefoniczną, wybierz trzy słowa zaczynające się na litery: *c*, *l* oraz *w* i znajdź je w wybranej książce. Zanotuj, ile razy ją otwierałeś, zanim znalazłeś stronę z poszukiwanym słowem.

Jeśli książka, którą wybrałeś, ma między 1000 a 2000 stron, to dla znalezienia jednego słowa nie powinieneś otwierać jej częściej niż 11 razy; jeśli ma między 500 a 1000 stron – to nie częściej niż 10 razy; jeśli między 250 a 500 stron – to nie częściej niż 9 razy itp.

Skąd to wiemy? Przypuszczamy, że w poszukiwaniu hasła, po zajrzeniu na wybraną stronę wiesz, że znajduje się ono przed nią albo po niej, możesz więc jedną z części książki pominąć w dalszych poszukiwaniach. Co więcej, w nieodrzuconej części kartek wybierasz jako kolejną tę, która jest bliska środka lub leży w pobliżu litery, na którą zaczyna się poszukiwany wyraz. Stosujesz więc – może nawet o tym nie wiedząc – metodę poszukiwania, która polega na **podziale (połowieniu) przeszukiwanego zbioru**. Możesz ją zastosować, bo przeszukiwany zbiór jest uporządkowany. A ile prób musiałbyś wykonać, gdyby hasła w słowniku nie były uporządkowane?

Porównaj teraz:

- W alfabetycznym spisie telefonów na 1000 stronach wystarczy przejrzeć co najwyżej 10 stron, by znaleźć numer telefonu danej osoby.
- A jeśli miałbyś znaleźć osobę, która ma telefon o numerze 1234567, to w najgorszym przypadku musiałbyś przejrzeć wszystkie 1000 stron!

Pamiętaj. Staraj się wykorzystywać uporządkowanie zbioru, który przeszukujesz – znacznie skraca to Twoją pracę. Utrzymuj również porządek w swoich zbiorach (rzeczach), a będziesz mniej czasu tracił na ich przeszukiwanie.

Czy to porównanie nie świadczy o potędze uporządkowania i o sile algorytmu zastosowanego do uporządkowanego wykazu?

Zabawa w zgadywanie liczb

Strategię podobną do poszukiwania w alfabetycznych spisach stosuje się podczas gry, polegającej na zgadywania ukrytej przez drugą osobę liczby. Wybierz sobie partnera do gry, która polega na odgadywaniu liczby naturalnej wybranej z przedziału [*m*, *n*]. Partner wybiera liczbę, a Ty masz ją odgadnąć. Na Twój wybór partner

[17] W teorii informacji, informacja jest definiowana jako „miara niepewności zajścia pewnego zdarzenia spośród skończonego zbioru zdarzeń możliwych" – na podstawie *Nowej encyklopedii powszechnej PWN*, t. 1-6, WN PWN, Warszawa 1996.

może jedynie odpowiedzieć: „tak", „za mała" lub „za duża". Jaką przyjmiesz strategię odgadywania liczby, by ją znaleźć w możliwie najmniejszej liczbie prób? Zamieńcie się rolami i powtórzcie te grę dla różnych przedziałów i dla różnych ukrytych liczb.

Rozegraj ze swoim partnerem kilka rund tej gry. Wśród ukrywanych liczb niech będą liczby z obu końców przedziału oraz liczba ze środka przedziału. Za lewy koniec przedziału wybierz również liczbę większą od 1.

Jeśli nie od razu, to na pewno po kilku próbach odkryjesz, że najlepsza strategia polega na podawaniu środkowych liczb z przedziału, w którym znajduje się poszukiwana liczba. Przypuśćmy, że poszukujemy liczby w przedziale [1, 100]. Jeśli pierwszym wyborem byłaby liczba 75 i otrzymalibyśmy odpowiedź „za duża", to pozostałby do przeszukania przedział [1, 75]. Jeśli natomiast wybierzemy w pierwszej próbie 50, to bez względu na to, jaką liczbę wybrał partner, pozostanie do przeszukania nie więcej niż pięćdziesiąt liczb, w przedziale [1, 49] albo [51, 100].

Przedstawione przykłady poszukiwania przez połowienie w zbiorze uporządkowanym ilustrują, że ta metoda jest kolejnym zastosowaniem **zasady dziel i zwyciężaj**.

Algorytm poszukiwania przez połowienie

Algorytm poszukiwania przez połowienie jest zwany również **binarnym poszukiwaniem**. Opiszemy teraz ten algorytm dla trochę ogólniejszej sytuacji niż w zabawie w odgadywanie liczb. Po pierwsze zauważmy, że w podanej wyżej metodzie nie mają znaczenia wartości elementów w tablicy, tak długo, jak długo są uporządkowane. Musimy mieć jedynie pewność, że po porównaniu wskazanej w tablicy liczby z poszukiwaną i po wybraniu połowy zbioru, poszukiwana liczba znajduje się w wybranej połowie, a do tego wystarczy, by elementy w tablicy były uporządkowane, nie muszą być koniecznie kolejnymi liczbami. Tablica może również zawierać takie same liczby – wtedy oczywiście zajmują one miejsca obok siebie. Po drugie, poszukiwana liczba nie musi znajdować się w tablicy – wtedy naszą odpowiedzią będzie jakaś specjalnie wybrana liczba, np. –1.

Przyjmijmy, że przeszukiwany ciąg liczb jest umieszczony w tablicy $x[k..l]$. Załóżmy dodatkowo, że wartość poszukiwanego elementu y mieści się w przedziale wartości elementów w tej tablicy, czyli $x_k \le y \le x_l$. Algorytm, który podajemy gwarantuje, że w trakcie jego działania, podobnie jak w grze w odgadywanie liczby, przeszukiwany przedział zawiera poszukiwany element y, czyli $x_{lewy} \le y \le x_{prawy}$. Ta własność oraz to, że długość tego przedziału zmniejsza się w każdej iteracji (zob. krok 3), zapewniają, że poniższy algorytm jest poprawny.

Algorytm poszukiwania przez połowienie (algorytm binarnego przeszukiwania)

Dane: Uporządkowany ciąg liczb w tablicy $x[k..l]$, tzn. $\boldsymbol{x_k \le x_{k+1} \le \ldots \le x_l}$; oraz element y spełniający nierówności $\boldsymbol{x_k \le y \le x_l}$.

Wyniki: Takie s ($k \le s \le l$), że $x_s = y$, lub przyjąć $s = -1$, jeśli $y \ne x_i$ dla każdego i ($k \le i \le l$).

Krok 1. *lewy* := *k*; *prawy* := *l*; {Początkowe końce przeszukiwanego przedziału.}

Krok 2. Jeśli *lewy* › *prawy*, to przypisz *s* := –1 i zakończ algorytm.
{Oznacza to, że poszukiwanego elementu *y* nie ma w przeszukiwanej tablicy.}

Krok 3. *s*:=(*lewy* + *prawy*) div 2; {Operacja div oznacza dzielenie całkowite.}
Jeśli $x_s = y$, to zakończ algorytm. {Znaleziono element *y* w przeszukiwanej tablicy.}
Jeśli $x_s < y$, to *lewy* :=*s* + 1, a w przeciwnym razie *prawy* := *s* – 1.
Wróć do kroku 2.

Implementacja poszukiwania przez połowienie

Jak podaje Donald E. Knuth [4], napisanie w pełni poprawnej komputerowej implementacji algorytmu poszukiwania przez połowienie, sprawiło kłopot wielu programistom. Oto nasza implementacja:

```
function PrzeszukiwanieBinarne(x:tablicax; k,l:integer;
                               y:integer):integer;
  {Przeszukiwanie binarne ciagu x[k..l] w poszukiwaniu elementu y.
   Wartoscia funkcji jest indeks elementu tablicy rownego y, lub -1,
   jesli brak takiego elementu. W programie glownym nalezy zdefiniowac
   typ danych:  tablicax=array[1..n] of integer.}
 var Lewy,Prawy,Srodek:integer;
begin
 Lewy:=k;  Prawy:=l;
 while Lewy<=Prawy do begin
  Srodek:=(Lewy+Prawy) div 2;
  if x[Srodek]=y then begin
   PrzeszukiwanieBinarne:=Srodek; exit
  end;  {Element y nalezy do przeszukiwanego ciagu.}
  if x[Srodek]<y then Lewy:=Srodek+1
  else Prawy:=Srodek-1
 end;
 PrzeszukiwanieBinarne:=-1
end; {PrzeszukiwanieBinarne}
```

Złożoność algorytmu binarnego przeszukiwania

Nasuwa się teraz pytanie, ile porównań jest wykonywanych w algorytmie binarnego przeszukiwania. Załóżmy, że poszukiwany element znajduje się w ciągu – bo jeśli go tam nie ma, to jest wykonywana jedna dodatkowa iteracja (przekonaj się o tym).

Pytanie o liczbę porównań w powyższym algorytmie można sformułować następująco: ile razy należy odrzucać połowę bieżącego ciągu, by pozostał tylko jeden element (zauważmy tutaj, że jeśli elementu *y* nie ma w ciągu, to kontynuujemy algorytm aż do wyczerpania wszystkich elementów, czyli wykonujemy o jeden krok więcej). Jeśli $n = 32$, to jeden element pozostaje po pięciu podziałach, a jeśli $n = 16$ – to po czterech. Stąd można wywnioskować, że jeśli wartość *n* zawiera się między 16 a 32, to wykonujemy nie więcej niż pięć porównań. Jak tę obserwację można uogólnić? Zapewne jest to związane z potęgą liczby 2, a dokładniej z najmniejszym wykładnikiem potęgi, której wartość nie jest mniejsza od *n*. Pojawia się więc tutaj w naturalny sposób funkcja odwrotna do potęgowania – **logarytm**. Można nawet przyjąć **„informatyczną" definicję tej funkcji**:

> $\log_2 n$ jest równy liczbie kroków prowadzących od *n* do 1, w których bieżąca liczba jest zastępowana przez zaokrąglenie w górę jej połowy.

Algorytm binarnego umieszczania

Algorytm binarnego przeszukiwania ma dość istotne uogólnienie, gdy dla elementu *y*, bez względu na to, czy należy do ciągu czy nie, chcemy znaleźć takie miejsce, by po wstawieniu go tam, ciąg pozostał uporządkowany. Odpowiedni algorytm można w tym przypadku nazwać **binarnym umieszczaniem**.

Poszukiwanie interpolacyjne, czyli poszukiwania w słownikach

Czy rzeczywiście przeszukiwanie binarne jest najszybszą metodą znajdowania elementu w zbiorze uporządkowanym?

Wyobraźmy sobie, że mamy znaleźć w książce telefonicznej numer telefonu pana Bogusza Alfreda. Wtedy zapewne skorzystamy z tego, że litera B jest blisko początku alfabetu i, owszem, zastosujemy metodę podziału. W pierwszej próbie nie będziemy jednak dzielić książki na dwie połowy, ale raczej spróbujemy trafić blisko tych stron, na których znajdują się nazwiska zaczynające się na literę B. W dalszych krokach będziemy postępować

podobnie. Tę obserwację można wykorzystać w algorytmie poszukiwania. Zauważmy najpierw, że w algorytmach binarnych jest sprawdzana jedynie relacja, **czy** dana liczba *y* jest większa (lub mniejsza lub równa) od wybranej z ciągu, natomiast nie sprawdzamy i nie wykorzystujemy tego, **jak bardzo** jest większa. Podczas odnajdywania wyrazów w encyklopediach korzystamy natomiast z informacji, w jakim miejscu alfabetu znajduje się litera, którą rozpoczyna się poszukiwany wyraz, i w zależności od tego wybieramy odpowiednią porcję kartek. Strategia ta nazywa się **interpolacyjnym poszukiwaniem**, gdyż uwzględnia nie tylko położenie szukanej liczby względem środka ciągu, ale uwzględnia jej wartość względem rozpiętości krańcowych wartości w ciągu. Szczegółowe informacje na temat poszukiwania interpolacyjnego można znaleźć w jednej z książek autora [7].

7 DZIEL I ZWYCIĘŻAJ, REKURENCJA – SORTOWANIE PRZEZ SCALANIE

W rozdziale 3 rozważaliśmy problem jednoczesnego znajdowania w zbiorze największego i najmniejszego elementu i podaliśmy algorytm **Max-i-Min**, w którym można wyróżnić trzy etapy:

1. Podział problemu na podproblemy.
2. Rozwiązywanie podproblemów.
3. Połączenie rozwiązań podproblemów w rozwiązanie głównego problemu.

Pierwszy etap polega na podziale zbioru danych na dwa podzbiory: kandydatów na maksimum i kandydatów na minimum, w drugim etapie – maksimum i minimum są znajdowane w zbiorach kandydatów, a trzeci etap w przypadku tego problemu jest jedynie wyprowadzeniem rozwiązania składającego się z dwóch liczb, otrzymanych jako rozwiązania dwóch podproblemów.

Podkreślaliśmy już wielokrotnie, że naturalnym podejściem w rozwiązywaniu problemów powinno być dążenie do wykorzystania znanych rozwiązań problemów – miejscem ku temu w powyższym schemacie jest etap drugi. W przypadku problemu **Max-i-Min**, korzystamy na tym etapie ze znanych algorytmów znajdowania największego i najmniejszego elementu w zbiorze. To naturalne podejście w rozwiązywaniu problemów przyjmuje szczególną postać, gdy tworzone podproblemy rozwiązujemy… tą samą metodą, jaką stosujemy do głównego problemu. A zatem, podproblemy również dzielimy na ich podproblemy i stosujemy… tę samą metodę. Kontynuujemy ten proces podziału tak długo, aż dojdziemy do podproblemu, dla którego znamy rozwiązanie, np. gdy ma mało elementów. W takich metodach rozwiązywania w naturalny sposób pojawia się **rekurencja**, która polega na tym, że algorytm odwołuje się do siebie samego. Na rekurencyjną metodę rozwiązywania problemów można więc spojrzeć również jak na chęć skorzystania ze znanego rozwiązania problemu, tylko że w tym przypadku jest to ten sam problem, a jego rozwiązanie… właśnie otrzymujemy.

> Nazwa zasady **dziel i zwyciężaj** pochodzi od angielskich słów *divide and conquer*. Nie należy jej jednak mylić z podobnie brzmiącą starożytną zasadą **dziel i rządź** (łac. *divide et impera*), która odnosiła się do sposobu rządzenia Cesarstwem Rzymskim, polegającego na dzieleniu wielkich obszarów i społeczeństw na mniejsze części, które w ten sposób miały utrudnioną komunikację między sobą, stanowiły więc mniejsze zagrożenie dla cesarzy. W przypadku zaś zasady **dziel i zwyciężaj** celem jest taki podział problemu na mniejsze części, by ich rozwiązania złożyły się na jak najefektywniejsze rozwiązanie głównego problemu.

Metoda **dziel i zwyciężaj** realizowana rekurencyjnie wraz z jej komputerową implementacją jest jedną z najsilniejszych technik komputerowego rozwiązywania problemów. W tej metodzie część organizacji obliczeń jest „zrzucana na komputer", faktycznie więc wykorzystywana jest potęga komputerów.

Zwróćmy jeszcze uwagę na bardzo ważną cechę metody dziel i zwyciężaj, dzięki której algorytmy tego typu są bardzo efektywne. Otóż na danym etapie podziału problemu na podproblemy staramy się, by problemy były definiowane na równolicznych lub niemal równolicznych podzbiorach danych. Jeśli więc dzielimy problem na

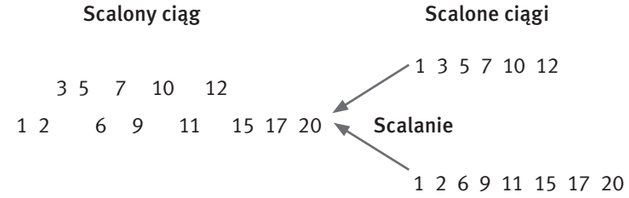
dwa podproblemy, to zwykle te podproblemy są definiowane na połowach zbioru danych. Ta własność nazywa się **równoważeniem** podproblemów.

Jeszcze jeden komentarz w kwestii, na ile podproblemów rozpada się problem w metodzie dziel i zwyciężaj. Najczęściej problem jest dzielony na dwa podproblemy i oba są dalej rozwiązywane. Nie zawsze tak musi być. Przeszukiwanie zbioru uporządkowanego metodą przez połowienie (p. 6.2) jest przykładem algorytmu dziel i zwyciężaj, w którym problem jest dzielony na dwa podproblemy ale dalej jest rozwiązywany tylko jeden z nich – w tej części danych, gdzie może znajdować się poszukiwany element. Znane są zastosowania metody dziel i zwyciężaj – na przykład mnożenie macierzy – w których problem o rozmiarze n jest dzielony na osiem podproblemów o rozmiarze $n/2$.

W tym rozdziale zastosujemy metodę dziel i zwyciężaj w algorytmie porządkowania przez scalanie. W tym algorytmie wykorzystywana jest **metoda scalania** uporządkowanych ciągów, czyli ich łączenia w jeden ciąg.

Scalanie ciągów uporządkowanych

Przyjmijmy, że scalane zbiory są uporządkowanymi ciągami liczb i wynik scalania ma być również ciągiem uporządkowanym. Scalenie dwóch uporządkowanych ciągów w jeden ciąg uporządkowany może być wykonane w bardzo prosty sposób: patrzymy na początki danych ciągów i do tworzonego ciągu przenosimy mniejszy z elementów czołowych lub którykolwiek z nich, jeśli są równe. Ilustrujemy to przykładem na rysunku 11.

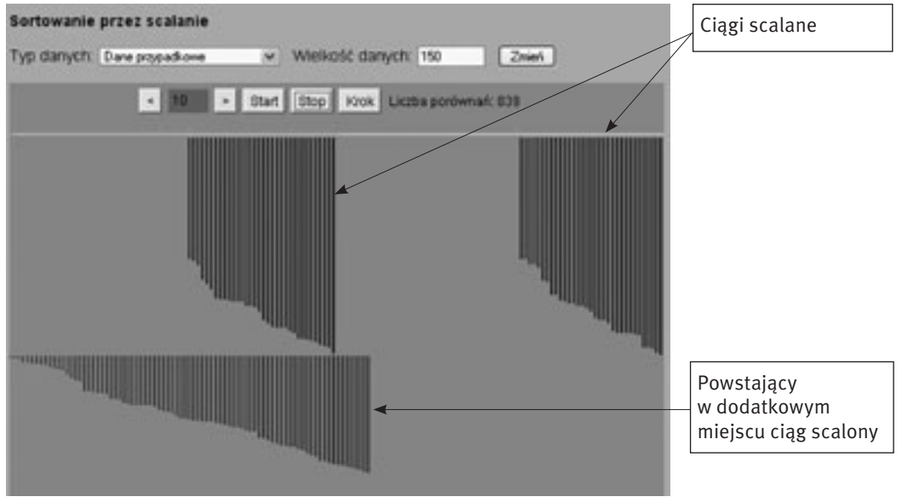


Rysunek 11.
Przykład scalania dwóch ciągów uporządkowanych

Kolejność przenoszenia elementów z ciągów zależy od ich wartości, które mogą powodować, że jeden ciąg może być znacznie wcześniej przeniesiony niż drugi. Czy można określić, ile porównań należy wykonać, aby scalić dwa ciągi? Liczba porównań może zależeć od wartości elementów. Zastanówmy się więc, ile najwięcej porównań musimy wykonać, by scalić dwa ciągi? Rozważanie różnych możliwych układów wartości elementów w obu scalanych ciągach nie daje szybkiej odpowiedzi na to pytanie. Spójrzmy natomiast od strony tworzonego ciągu. Z wyjątkiem elementu przeniesionego do niego na końcu, przeniesienie każdego innego elementu może być związane z wykonaniem porównania między elementami danych ciągów. Tak jest wtedy, gdy w kroku przed przedostatnim, oba scalane ciągi zawierają jeszcze po jednym elemencie. Przypuśćmy więc, że na początku jeden ciąg zawiera k elementów, a drugi ma l elementów razem n, czyli $n = k + l$. Ta dyskusja prowadzi do konkluzji, że bez względu na liczności danych ciągów, ich scalenie może wymagać wykonania $n - 1$ porównań i każde z tych porównań jest związane z przeniesieniem jednego elementu, z wyjątkiem elementu, który trafia do scalonego ciągu na końcu.

Z tej dyskusji wynika jeszcze jeden wniosek – ponieważ nie potrafimy przewidzieć, który z ciągów i kiedy wyczerpie się w trakcie scalania, tworzony ciąg nie może być umieszczany na miejscu ciągów danych do scalenia, musi więc być tworzony na nowym miejscu. Zatem potrzebna jest dodatkowa pamięć – mówimy w takim przypadku, że ta metoda nie działa *in situ*. Ta dodatkowo wykorzystywana pamięć jest słabą stroną operacji scalania.

Polecamy uruchomienie programu **Sortowanie** i przyjrzenie się rozszerzonej demonstracji działania algorytmu porządkowania przez scalanie. Zaobserwuj przebieg etapu scalania, który ma miejsce poniżej sortowanego ciągu, na kolejnym rysunku:



Rysunek 12.
Scalanie dwóch ciągów uporządkowanych – ilustracja z programu **Sortowanie** pokazująca, że scalanie nie może być wykonywane w tym samym miejscu

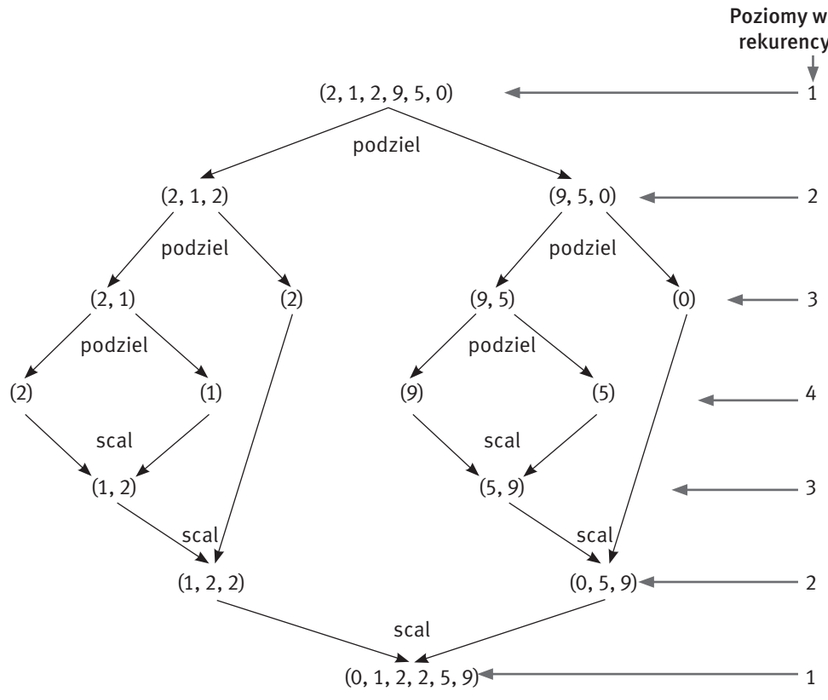
Algorytm scalania dwóch ciągów uporządkowanych – Scal

Dane: Dwa uporządkowane ciągi x i y.
Wynik: Uporządkowany ciąg z, będący scaleniem ciągów x i y.
Krok 1. Dopóki oba ciągi x i y nie są puste wykonuj następującą operację:
przenieś mniejszy z najmniejszych elementów z ciągów x i y do ciągu z.
Krok 2. Do końca ciągu z dopisz elementy pozostałe w jednym z ciągów x lub y.

Porządkowanie przez scalanie

Algorytm scalania dwóch uporządkowanych ciągów można zastosować do tworzenia uporządkowanych ciągów z podciągów już uporządkowanych. A czy może być jakiś pożytek z tego algorytmu scalania uporządkowanych ciągów przy porządkowaniu dowolnych ciągów? Tak – jeśli potrafimy najpierw uporządkować te podciągi. A czy moglibyśmy założyć, że te podciągi zostały uporządkowane... tą samą metodą? Możemy – i tutaj przydaje się **rekurencja**. Dodatkowo założymy, że na każdym etapie podciągi mają prawie taką samą długość. Dochodzimy w ten sposób do metody porządkowania ciągu, która, na **zasadzie dziel i zwyciężaj**, scala dwa prawie równoliczne podciągi, uporządkowane tą samą metodą. Działanie tej metody można prześledzić na rysunku 13.

Zapiszemy teraz algorytm porządkowania przez scalanie w postaci listy kroków. Z powyższego szkicu oraz z ilustracji na rysunku 13 wynika, że porządkowany ciąg jest dzielony w każdym kroku na dwa, niemal równej długości podciągi, które rekurencyjnie są porządkowane tą samą metodą. Warunkiem zakończenia rekurencji w tym algorytmie jest sytuacja, gdy ciąg ma jeden element, wtedy nie można go już podzielić na podciągi, chociaż nie ma nawet po co – jest to już bowiem ciąg uporządkowany. Zatem powrót z wywołań rekurencyjnych rozpoczyna się z ciągami złożonymi z pojedynczych elementów, które są scalane w ciągi o długości dwa, następnie w ciągi o długości trzy lub cztery itd. Jak zwykle, w opisie algorytmu rekurencyjnego, po nazwie algorytmu występuje układ parametrów określających rozwiązywany problem, który jest wykorzystany w treści algorytmu, w odwołaniu do niego samego przy rozwiązywaniu mniejszych podproblemów.

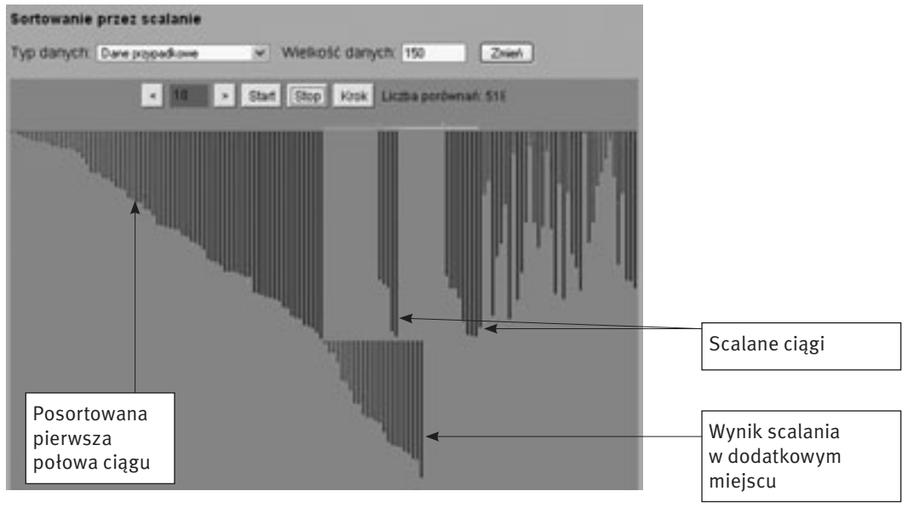


Rysunek 13.
Przykład działania algorytmu porządkowania przez scalanie

Algorytm porządkowania przez scalanie MergeSort(*l,p,x*)
Dane: Ciąg liczb $x_l, x_{l+1}, ..., x_p$.
Wynik: Uporządkowanie tego ciągu liczb od najmniejszej do największej.
Krok 1. Jeśli $l < p$, to przyjmij $s:=(l+p)$ div 2 i wykonaj trzy następne kroki.
Krok 2. Zastosuj ten sam algorytm do ciągu (*l,s,x*), czyli wykonaj **MergeSort**(*l,s,x*).
Krok 3. Zastosuj ten sam algorytm do ciagu (*s+1,p,x*), czyli wykonaj **MergeSort**(*s+1,p,x*).
Krok 4. Zastosuj algorytm scalania **Scal** do ciągów $(x_l, ..., x_s)$, $(x_{s+1}, ..., x_p)$ i wynik umieść z powrotem w ciągu $(x_l, ..., x_p)$.

Obejrzyj na rysunku 14 kolejną demonstrację w programie **Sortowanie**. Zwróć uwagę na poszczególne etapy algorytmu: wywołania rekurencyjne dla coraz krótszych ciągów i scalanie podciągów w coraz dłuższe ciągi.

Jak już wspomnieliśmy przy okazji scalania ciągów – wykonanie tej operacji wymaga dodatkowego miejsca w pamięci komputera (patrz również rys. 12 i 14). Można jednak nie zapełniać tego miejsca w całości, oszczędzając przy tym na liczbie wykonywanych operacji. W algorytmie scalania dwóch uporządkowanych ciągów **Scal**, w kroku 2 końcowa część jednego z ciągów jest dopisywana do ciągu *z*. Gdy są scalane dwa podciągi tego samego ciągu (jak w algorytmie porządkowania przez scalanie), to nie zawsze jest to konieczne: jeśli należy przenieść końcową część pierwszego podciągu, to można od razu umieścić ją na końcu tworzonego ciągu (uwaga, trzeba to robić od końca przenoszonego podciągu), a jeśli należy przenieść końcową część drugiego podciągu, to można zostawić ją tam, gdzie jest. Taka modyfikacja kroku 4 w algorytmie **MergeSort** została uwzględniona w implementacji tego algorytmu na następnej stronie:



Rysunek 14.
Demonstracja działania porządkowania przez scalanie w programie **Sortowanie**. Posortowana jest już pierwsza połowa ciągu i w trakcie sortowania drugiej połowy scalane są dwa podciągi z pierwszej części drugiej połowy, uporządkowane wcześniej rekurencyjnie tą samą metodą

```
procedure MergeSort(Dol,Gora:integer; var x:Tablica1n);
 {Porzadkowanie metoda przez scalanie.}
var s:integer;
procedure Scal(l,s,p:integer);
  {Scalanie podciagow uporzadkowanych x[l..s-1] i x[s..p]
   w ciag uporzadkowany x[l..p].}
 var i,j,k,m:integer;
     z      :Tablica1n;
begin
 i:=l;  j:=s;  m:=l;
 while (i<s) and (j<=p) do begin
  if x[i]<=x[j] then begin
   z[m]:=x[i];  i:=i+1 end
  else begin z[m]:=x[j];  j:=j+1 end;
  m:=m+1
 end;  {while}
 if i<s then begin
  j:=s-1;  k:=p;
  while j>=i do begin
   x[k]:=x[j];
   k:=k-1;  j:=j-1
  end
 end;
 for i:=l to m-1 do x[i]:=z[i]
end;  {Scal}
begin {MergeSort}
```

```
  if Dol<Gora then begin
   s:=(Dol+Gora) div 2;
   MergeSort(Dol,s,x);
   MergeSort(s+1,Gora,x);
   Scal(Dol,s+1,Gora)
  end
 end; {MergeSort}
```

Złożoność sortowania ciągu n liczb przez scalanie wynosi około $n\log_2 n$, jest zatem znacznie mniejsza niż złożoność algorytmu sortowania przez wybór.

LITERATURA

1. Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów*, WNT, Warszawa 1997
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO*, WSiP, Warszawa 2002-2003
3. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
4. Knuth D.E., *Sztuka programowania*, tomy 1–3, WNT, Warszawa 2003
5. Nievergelt J., *Co to jest dydaktyka informatyki?*, „Komputer w Edukacji" 1/1994
6. Steinhaus H., *Kalejdoskop matematyczny*, WSiP, Warszawa 1989
7. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
8. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69
9. Wirth N., *Algorytmy + struktury danych = programy*, WNT, Warszawa 1980

Czy wszystko można policzyć na komputerze

Maciej M. Sysło

Uniwersytet Wrocławski, UMK w Toruniu

syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl

http://mmsyslo.pl/

Streszczenie

Przedmiotem wykładu jest łagodne wprowadzenie do złożoności problemów i algorytmów. Przewodnim pytaniem jest, jak dobrze sprawują się algorytmy i komputery i czy komputery już mogą wszystko obliczyć. Z jednej strony dla niektórych problemów (jak znajdowanie najmniejszego elementu) znane są algorytmy, które nie mają konkurencji, gdyż są bezwzględnie najlepsze, a z drugiej – istnieją takie, o których przypuszcza się, że komputery nigdy nie będą w stanie ich rozwiązywać dostatecznie szybko. Przedstawione zostaną problemy, których nie potrafimy rozwiązywać szybko, nawet z użyciem najszybszych komputerów. Problemy z tej drugiej grupy znajdują zastosowanie na przykład w kryptografii. Podejmowane kwestie będą ilustrowane praktycznymi zastosowaniami omawianych problemów i ich metod obliczeniowych.

Spis treści

1 WPROWADZENIE

Można odnieść wrażenie, że komputery są obecnie w stanie wykonać wszelkie obliczenia i dostarczyć oczekiwany wynik w krótkim czasie. Budowane są jednak coraz szybsze komputery, co może świadczyć o tym, że moc istniejących komputerów nie jest jednak wystarczająca do wykonania wszystkich obliczeń, jakie nas interesują, potrzebne są więc jeszcze szybsze maszyny.

Zgodnie z nieformalnym prawem Moore'a, szybkość działania procesorów stale rośnie i podwaja się co 24 miesiące. Jednak istnieje wiele trudno rozwiązywalnych problemów, których obecnie nie jesteśmy w stanie rozwikłać za pomocą żadnego komputera i zwiększanie ich szybkości niewiele zmienia tę sytuację. Kluczowe staje się więc opracowywanie coraz szybszych algorytmów.

2 SUPERKOMPUTERY I ALGORYTMY

Superkomputery

Komputery, które w danej chwili lub w jakimś okresie mają największą moc obliczeniową przyjęło się nazywać **superkomputerami**. Prowadzony jest ranking najszybszych komputerów świata (patrz strona http://www.top500.org/), a faktycznie odbywa się wyścig wśród producentów superkomputerów i dwa razy w roku publikowane są listy rankingowe. Szybkość działania komputerów jest oceniana na specjalnych zestawach problemów, pochodzących z algebry liniowej – są to na ogół układy równań liniowych, złożone z setek tysięcy, a nawet milionów równań i niewiadomych. Szybkość komputerów podaje się w jednostkach zwanych **FLOPS** (**flops** lub **flop/s**) – jest to akronim od ***FL**oating point **O**perations **P**er **S**econd*, czyli zmiennopozycyjnych operacji na sekundę. Dla uproszczenia można przyjąć, że FLOPS to są działania arytmetyczne (dodawanie, odejmowanie, mnożenie i dzielenie) wykonywane na liczbach dziesiętnych z kropką. W użyciu są jednostki: YFlops (yotta flops) – 10^{24} operacji na sekundeę, ZFlops (zetta flops) – 10^{21}, EFlops (exa flops) – 10^{18}, PFlops (peta flops) – 10^{15}, TFlops (tera flops) – 10^{12}, GFlops (giga flops) – 10^{9}, MFlops (mega flops) – 10^{6}, KFlops (kilo flops) – 10^{3}.

W rankingu z listopada 2009 roku, najszybszym komputerem świata był **Cray XT Jaguar** firmy Cray, którego moc wynosi 1,75 PFlops, czyli wykonuje on ponad 10^{15} operacji na sekundę. Komputer ten pracuje w Department of Energy's Oak Ridge National Laboratory w Stanach Zjednoczonych. Komputer Jaguar jest zbudowany z 224 162 procesorów Opteron firmy AMD. Drugie miejsce należy do Roadrunnera firmy IBM o mocy 1,04 PFlops. Dwa dalsze miejsca zajmują również komputery Cray i IBM, a w pierwszej dziesiątce są jeszcze dwa inne komputery IBM.

Na początku 2010 roku najszybszym procesorem PC był Intel Core i7 980 XE o mocy 107,6 GFlops. Większą moc mają procesory graficzne, np. Tesla C1060 GPU (nVidia) posiada moc 933 GFlops (w pojedynczej dokładności), a HemlockXT 5970 (AMD) osiąga 4640 GFlops (w podwójnej dokładności).

Bardzo dużą moc uzyskują obliczenia wykonywane na rozproszonych komputerach osobistych połączonych ze sobą za pomocą Internetu. Na przykład, ma początku 2010 roku Folding@Home osiągnął moc 3,8 PFlops, a komputery osobiste pracujące w projekcie sieciowym GIMPS nad znalezieniem coraz większej liczby pierwszej uzyskały moc 44 TFlops.

Przy obecnym tempie wzrostu możliwości superkomputerów przewiduje się, że moc 1 EFlops (10^{18} operacji na sekundę) zostanie osiągnięta w 2019 roku. Naukowcy oceniają, że dla pełnego modelowania zmian pogody na Ziemi w ciągu dwóch tygodni jest potrzebny komputer o mocy 1 ZFlops (10^{21} operacji na sekundę). Przewiduje się, że taki komputer powstanie przed 2030 rokiem.

W dalszych rozważaniach będziemy przyjmować, że dysponujemy najszybszym komputerem, który ma moc 1 PFlops, czyli wykonuje 10^{15} = 1 000 000 000 000 000 operacji na sekundę.

Superkomputery a algorytmy

Ciekawi nas teraz, jak duże problemy możemy rozwiązywać za pomocą komputera o mocy 1 PFlops. W tabeli 1 zamieszczano czasy obliczeń wykonanych na tym superkomputerze, posługując się algorytmami o różnej złożoności obliczeniowej (pracochłonności) i chcąc rozwiązywać problemy o różnych rozmiarach.

Tabela 1.
Czasy obliczeń za pomocą algorytmów o podanej złożoności, wykonywanych na superkomputerze o mocy 1 PFlps (« 1 sek. oznacza w tabeli, że czas obliczeń stanowił niewielki ułamek sekundy; parametr n określa rozmiar danych.)

Złożoność algorytmu	$n = 100$	$n = 500$	$n = 1000$	$n = 10000$
$\log n$	« 1 sek.	« 1 sek.	« 1 sek.	« 1 sek.
n	« 1 sek.	« 1 sek.	« 1 sek.	« 1 sek.
$n \log n$	« 1 sek.	« 1 sek.	« 1 sek.	« 1 sek.
n^2	« 1 sek.	« 1 sek.	0,000000001 sek.	0,0000001 sek.
n^5	0,00001 sek.	0,03125 sek.	1 sek.	1,15 dni
2^n	$4*10^7$ lat	$1{,}038*10^{128}$ lat	$3{,}3977*10^{278}$ lat	
$n!$	$2{,}959*10^{135}$ lat			

Wyraźnie widać, że dwa ostatnie algorytmy w tabeli są całkowicie niepraktyczne nawet dla niewielkiej liczby danych ($n = 100$). Istnieją jednak problemy, dla których wszystkich możliwych rozwiązań może być 2^n lub $n!$, zatem nawet superkomputery nie są w stanie przejrzeć wszystkich możliwych rozwiązań w poszukiwaniu najlepszego, a zdecydowanie szybszymi metodami dla tych problemów nie dysponujemy.

Można łatwo przeliczyć, że gdyby nasz superkomputer był szybszy, na przykład miał moc 1 ZFlops (10^{21} operacji na sekundę), to dwa ostatnie wiersze w tabeli 1 uległyby tylko niewielkim zmianom, nie na tyle jednak, by móc stosować dwa ostatnie algorytmy w celach praktycznych.

W następnym rozdziale przedstawimy kilka przykładowych problemów, dla których moc obliczeniowa superkomputerów może nie być wystarczająca, by rozwiązywać je dla praktycznych rozmiarów danych. Ma to złe i dobre strony. Złe – bo nie jesteśmy w stanie rozwiązywać wielu ważnych i istotnych dla człowieka problemów, takich np. jak prognozowanie pogody, przewidywanie trzęsień Ziemi i wybuchów wulkanów, czy też planowanie optymalnych podróży lub komunikacji. Dobre zaś – bo można wykorzystywać trudne obliczeniowo problemy w metodach szyfrowania, dzięki czemu nasz przeciwnik nie jest w stanie, nawet posługując się najszybszymi komputerami, deszyfrować naszych wiadomości, chociaż my jesteśmy w stanie szybko je kodować i wysyłać.

3 PRZYKŁADY ZŁOŻONYCH PROBLEMÓW

W tym rozdziale, jak i w dalszych, problemy będą definiowane w postaci **specyfikacji**, która zawiera ścisły opis *Danych* i oczekiwanych *Wyników*. W specyfikacji są też zawarte zależności między danymi i wynikami. Specyfikacja problemu jest również specyfikacją algorytmu, który ten problem rozwiązuje, co ma na celu dokładne określenie przeznaczenia algorytmu, stanowi zatem powiązanie algorytmu z rozwiązywanym problemem.

3.1 NAJKRÓTSZA TRASA ZAMKNIĘTA

Jednym z najbardziej znanych problemów związanych z wyznaczaniem tras przejazdu, jest **problem komiwojażera**, oznaczany zwykle jako **TSP** (ang. *Travelling Salesman Problem*). Podany jest dany zbiór miejscowości oraz odległości między nimi. Należy znaleźć drogę zamkniętą najkrótszą, przechodzącą przez każdą miejscowość dokładnie jeden raz. Problem ten został omówiony w punkcie 5.2.1 w rozdziale Informatyka – klucz do zrozumienia, kariery, dobrobytu.

3.2 ROZKŁAD LICZBY NA CZYNNIKI PIERWSZE

Liczby pierwsze stanowią w pewnym sensie „pierwiastki" wszystkich liczb, każdą bowiem liczbę całkowitą można jednoznacznie przedstawić, z dokładnością do kolejności, w postaci iloczynu liczb pierwszych. Na przykład, 4 = 2*2; 10 = 2*5; 20 = 2*2*5; 23 = 23; dla liczb pierwszych te iloczyny składają się z jednej liczby.

Matematycy interesowali się liczbami pierwszymi od dawna. Pierwsze spisane rozważania i twierdzenia dotyczące tych liczb znajdujemy w działach Euklidesa. Obecnie liczby pierwsze znajdują ważne zastosowania w kryptografii, m.in. w algorytmach szyfrujących. Do najważniejszych problemów związanych z liczbami pierwszymi należy badanie, czy dana dodatnia liczba całkowita n jest liczbą pierwszą i ewentualny jej rozkład na czynniki, jeśli jest liczbą złożoną.

3.3 PODNOSZENIE DO POTĘGI

Podnoszenie do potęgi jest bardzo prostym, szkolnym zadaniem. Na przykład, aby obliczyć 3^4, wykonujemy trzy mnożenia 4*4*4*4. A zatem w ogólności, aby obliczyć szkolną metodą wartość x^n, należy wykonać $n - 1$ mnożeń, o jedno mniej niż wynosi wykładnik potęgi. Czy ten algorytm jest na tyle szybki, by obliczyć na przykład wartość:

$$x^{12345678912345678912345678912345}$$

która może pojawić przy szyfrowaniu informacji przesyłanych w Internecie? Odpowiedź na to pytanie w punkcie 4.2.

3.4 PORZĄDKOWANIE

Problem **porządkowania**, często nazywany **sortowaniem**, jest jednym z najważniejszych problemów w informatyce i w wielu innych dziedzinach. Jego znaczenie jest ściśle związane z zarządzaniem danymi (informacjami), w szczególności z wykonywaniem takich operacji na danych, jak wyszukiwanie konkretnych danych lub umieszczanie danych w zbiorze.

Warto zauważyć, że mając n elementów, które chcemy uporządkować, istnieje $n!$ możliwych uporządkowań, wśród których chcemy znaleźć właściwe uporządkowanie. A zatem, przestrzeń możliwych rozwiązań w problemie porządkowania n liczb ma moc $n!$, czyli jest taka sama jak w przypadku problemu komiwojażera. Jednak dzięki odpowiednim algorytmom, n elementów można uporządkować w czasie proporcjonalnym do $n \log n$ lub n^2.

Dysponując uporządkowanym zbiorem danych, znalezienie w nim elementu lub umieszczenie nowego z zachowaniem porządku jest znacznie łatwiejsze i szybsze, niż gdyby zbiór nie był uporządkowany.

3.5 OBLICZANIE WARTOŚCI WIELOMIANU – SCHEMAT HORNERA

Obliczanie wartości wielomianu o zadanych współczynnikach jest jedną z najczęściej wykonywanych operacji w komputerze. Wynika to z ważnego faktu matematycznego, zgodnie z którym każdą funkcję (np. funkcje trygonometryczne) można zastąpić wielomianem, którego postać zależy od funkcji i od tego, jaką chcemy uzyskać dokładność obliczeń. Najefektywniejszym sposobem obliczania wartości wielomianu jest **schemat Hornera**, wynikający z następującego przekształcenia postaci wielomianu:

$$\begin{aligned} w_n(x) &= a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n = (a_0x^{n-1} + a_1x^{n-2} + \ldots + a_{n-1})x + a_n \\ &= ((a_0x^{n-2} + a_1x^{n-3} + \ldots + a_{n-2})x + a_{n-1})x + a_n = \\ &= (\ldots((a_0x + a_1)x + a_2)x + \ldots + a_{n-2})x + a_{n-1})x + a_n \end{aligned}$$

Ten ostatni wzór można zapisać w następującej postaci algorytmicznej:

$y := a_0$
$y := yz + a_i$ dla $i = 1, 2, \ldots, n$.

Stąd otrzymujemy algorytm:

Algorytm: schemat Hornera
Dane: n – nieujemna liczba całkowita (stopień wielomianu);
$a_0, a_1, \ldots, a_n$ – n+1 współczynników wielomianu;
z – wartość argumentu.
Wynik: $y = w_n(z)$ – wartość wielomianu stopnia n dla wartości argumentu $x = z$.
Krok 1. $y := a_0$ – początkową wartością jest współczynnik przy najwyższej potędze.
Krok 2. Dla $i = 1, 2, \ldots, n$ oblicz wartość dwumianu $y := yz + a_i$.

Z postaci algorytmu wynika, że obliczenie wartości wielomianu stopnia n wymaga wykonania n mnożeń i n dodawań. Udowodniono, że jest to najszybszy sposób obliczania wartości wielomianu.

Schemat Hornera ma wiele zastosowań, m.in. do:
- obliczania dziesiętnej wartości liczby, danej w systemie o podstawie p – współczynniki wielomianu są w tym przypadku cyframi $\{0, 1, \ldots, p-1\}$, a argumentem p podstawa systemu,
- szybkiego obliczania wartości potęgi (patrz punkt 4.2).

4 DWA TRUDNE PROBLEMY

Warto wrócić tutaj do dwóch problemów przedstawionych w rozdziale 3. Jednym z nich jest sprawdzanie, czy dana liczba jest liczbą pierwszą, a drugim – szybkie podnoszenie do potęgi. Dla pierwszego z tych problemów nie mamy dobrej wiadomości, nie jest bowiem znana żadna metoda szybkiego testowania pierwszości liczb. Korzysta się z tego na przykład w szyfrze RSA, częścią kluczy w tej metodzie jest liczba będąca iloczynem dwóch dużych liczb pierwszych i brak znajomości tego rozkładu jest gwarancją bezpieczeństwa tego szyfru.

Drugim problemem jest podnoszenie do dużych potęg i tutaj mamy bardzo dobrą wiadomość – podnoszenie nawet do bardzo dużych potęg, zawierających setki cyfr, trwa ułamek sekundy.

4.1 BADANIE ZŁOŻONOŚCI LICZB

Dla problemu badania, czy dana liczba n jest liczbą pierwszą czy złożoną, dysponujemy prostym algorytmem, który polega na dzieleniu n przez kolejne liczby naturalne i wystarczy dzielić tylko przez liczby nie większe niż $\sqrt{n}$, gdyż liczby n nie można rozłożyć na iloczyn dwóch liczb większych od $\sqrt{n}$. Algorytm ten ma bardzo prostą postać:

```
var i,n:integer;
i:=2;
while i*i <= n do begin
   if n mod i = 0 then return 1;  {n jest podzielne przez i}
   i=i+1
end;
return 0    {n jest liczba pierwszą}
```

Ten fragment programu zwraca 0, jeśli n jest liczbą pierwszą, i 1, gdy n jest liczbą złożoną. W tym drugim przypadku znamy także podzielnik liczby n. Ten fragment programu można rozszerzyć do programu generującego wszystkie dzielniki liczby n.

Liczba operacji wykonywanych przez powyższy program jest w najgorszym przypadku (gdy n jest liczbą pierwszą) proporcjonalna do $\sqrt{n}$, a więc jeśli $n = 10^m$, to wykonywanych jest $10^{m/2}$ operacji. Zatem są niewielkie szanse, by tym algorytmem rozłożyć na czynniki pierwsze liczbę, która ma kilkaset cyfr lub stwierdzić, że się jej nie da rozłożyć.

Rozkładem liczby złożonej na czynniki pierwsze mogą być zainteresowani ci, którzy starają się złamać szyfr RSA. Wiadomo, że jedna z liczb tworzących klucz publiczny i prywatny jest iloczynem dwóch liczb pierwszych. Znajomość tych dwóch czynników umożliwia utworzenie klucza prywatnego (tajnego). Jednak ich wielkość – są to liczby pierwsze o kilkuset cyfrach (200-300) – stoi na przeszkodzie w rozkładzie n.

Istnieje wiele algorytmów, które służą do testowania pierwszości liczb oraz do rozkładania liczb na czynniki pierwsze. Niektóre z tych algorytmów mają charakter probabilistyczny – wynik ich działania jest poprawny z prawdopodobieństwem bliskim 1. Żaden jednak algorytm, przy obecnej mocy komputerów, nie umożliwia rozkładania na czynniki pierwsze liczb, które mają kilkaset cyfr. Szyfr RSA pozostaje więc nadal bezpiecznym sposobem utajniania wiadomości, w tym również przesyłanych w sieciach komputerowych.

4.2 SZYBKIE PODNOSZENIE DO POTĘGI

Wracamy tutaj do obliczania wartości potęgi x^m, dla dużych wartości wykładnika m[18]. W punkcie 3.3 podaliśmy przykład dla m = 12345678912345678912345678912345. Stosując „szkolny” algorytm, czyli kolejne mnożenia, i korzystając z naszego superkomputera, obliczenie potęgi dla tego wykładnika zajęłoby $3*10^8$ lat. Ten wykładnik jest faktycznie niewielką liczbą, w porównaniu z wykładnikami, jakie pojawiają się przy stosowaniu szyfru RSA, potrzebny jest więc znacznie efektywniejszy algorytm.

Do wyprowadzenia algorytmu szybkiego potęgowania posłużymy się rekurencyjnym zapisem operacji potęgowania:

$$x^m = \begin{cases} 1 & \text{jeśli } m = 0 \\ (x^{m/2})^2, & \text{jeśli } m \text{ jest liczbą parzystą} \\ (x^{(m-1)/2})^2 x & \text{jeśli } m \text{ jest liczbą nieparzystą} \end{cases}$$

Na przykład, jeśli chcemy obliczyć x^{22}, to powyższa zależność rekurencyjna prowadzi przez następujące odwołania rekurencyjne: $x^{22} = (x^{11})^2 = ((x^5)^2x)^2 = (((x^2)^2x)^2x)^2$.

Warto zauważyć, jaki jest związek kolejności wykonywania mnożeń, wynikającej z powyższej zależności rekurencyjnej, z postacią binarną wykładnika, zapisaną z użyciem schematu Hornera. Dla naszego przykładu mamy $m = (22)_{10} = (10110)_2$. Porównując kolejność mnożeń z postacią binarną widać, że podnoszenie do kwadratu odpowiada kolejnym pozycjom w rozwinięciu binarnym, a mnożenie przez x odpowiada cyfrze 1 w rozwinięciu binarnym. A zatem, liczba mnożeń jest nie większa niż dwa razy długość binarnego rozwinięcia wykładnika. Wiemy skądinąd, że długość rozwinięcia binarnego liczby m wynosi ok. $\log_2 m$. Mamy w przybliżeniu $\log_2 12345678912345678912345678912345 = 104$, w związku z tym podniesienie do tej potęgi wymaga wykonania nie więcej niż ok. 200 mnożeń.

W tabeli 2 zamieściliśmy wartości logarytmu przy podstawie 2 z rosnących wartości liczby m.

[18] Więcej na ten temat – punkt 4.1.1 w rozdziale Informatyka – klucz do zrozumienia, kariery, dobrobytu.

Tabela 2.
Przybliżona długość rozwinięcia binarnego liczby *m*

m	$\log_2 m$
10^4	10
10^6	20
10^9	30
10^{12}	40
10^{20}	66,5
10^{50}	166
10^{100}	332,2

Z tabeli 2 wynika, że obliczenie wartości potęgi dla wykładnika o stu cyfrach wymaga wykonania nie więcej niż 670 mnożeń, a to trwa nawet na komputerze osobistym ułamek sekundy.

Problem potęgowania jest najlepszą ilustracją powiedzenia Ralfa Gomory'ego, że prawdziwe przyspieszenie obliczeń osiągamy dzięki efektywnym algorytmom, a nie szybszym komputerom.

LITERATURA

1. Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów*, WNT, Warszawa 1997
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO*, WSiP, Warszawa 2002-2003
3. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
4. Knuth D.E., *Sztuka programowania*, tomy 1–3, WNT, Warszawa 2003
5. Steinhaus H., *Kalejdoskop matematyczny*, WSiP, Warszawa 1989
6. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
7. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69
8. Wirth N., *Algorytmy + struktury danych = programy*, WNT, Warszawa 1980

Dlaczego możemy się czuć bezpieczni w sieci, czyli o szyfrowaniu informacji

Maciej M. Sysło
Uniwersytet Wrocławski, UMK w Toruniu
syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl
http://mmsyslo.pl/

Streszczenie

Informacje ukrywano przed innymi osobami jeszcze przed rozpowszechnieniem się pisma. Obecnie szyfrowanie informacji ma zarówno zastosowania militarne, jak i cywilne, np. przy przesyłaniu w sieci danych, których właściwego znaczenia nie powinien poznać nikt poza nadawcą i odbiorcą, nawet jeśli wiadomość wpadnie w niepożądane ręce. Wykład jest wprowadzeniem do kryptografii, ze szczególnym uwzględnieniem kryptografii komputerowej. Na początku zostaną omówione ciekawe metody szyfrowania, którymi posługiwano się przed erą komputerów, a które są wykorzystywane jeszcze dzisiaj. W drugiej zaś części wykładu zostanie zaprezentowany algorytm szyfrowania z kluczem publicznym, który jest wykorzystywany na przykład przy wysyłaniu rozwiązań zadań z Olimpiady Informatycznej. Za początek ery kryptografii komputerowej uważa się osiągnięcia zespołu polskich kryptoanalityków pod kierunkiem Mariana Rejewskiego, którzy złamali szyfr niemieckiej maszyny Enigma. Wspomniane będzie o tym krótko w czasie wykładu.

Fragmenty tego opracowania pochodzą z podręcznika [3].

Spis treści

1 WPROWADZENIE

Człowiek od początku swoich dziejów porozumiewał się z innym człowiekiem. Z czasem, wynalazł różne sposoby komunikowania się na odległość. Później, rodzące się społeczności, państwa i ich wodzowie potrzebowali sprawnych systemów łączności, by skutecznie władać swoimi, często rozległymi posiadłościami. Sprzeczne interesy różnych władców i dowódców powodowały, że zaczęli oni strzec wysyłanych przez siebie wiadomości przed przechwyceniem ich przez inne osoby. Tak narodziła się potrzeba **szyfrowania**, czyli utajniania treści przesyłanych wiadomości, i doprowadziło to do powstania **kryptografii**, dziedziny wiedzy, a niektórzy twierdzą, że także dziedziny sztuki, zajmującej się pierwotnie szyfrowaniem.

Wielokrotnie w historii ludzkości szyfrowanie miało istotny wpływ na bieg wydarzeń. Najbardziej spektakularnym przykładem jest chyba historia rozpracowania niemieckiej maszyny szyfrującej Enigma, dzięki czemu – jak utrzymują historycy – II wojna światowa zakończyła się o 2-3 lata wcześniej. Dużą w tym rolę odegrali Polacy (patrz rozdz. 7).

Sposoby utajniania wiadomości i łamania zabezpieczeń stanowią ze swej natury wiedzę tajemną – wiele faktów z tej dziedziny jest długo utrzymywanych w tajemnicy lub wcale nieujawnianych.

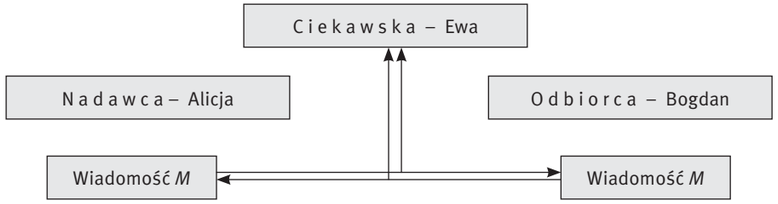
> „Ofiarami takiego stanu rzeczy stali się dwaj najbardziej zasłużeni dla informatyki Brytyjczycy, Charles Babbage i Alan Turing. Około 1854 roku, Babbage opracował sposób łamania szyfru Vigenère'a, który od połowy XVI wieku był powszechnie stosowany w łączności dyplomatycznej i wojskowej i uchodził za szyfr nie do złamania. O swoich pracach w tej dziedzinie Babbage nie wspomina nawet w swojej biografii – dowiedziano się o tym dopiero z jego korespondencji odczytanej sto lat później. Całe zasługi na tym polu przypisuje się Fridrichowi W. Kasiskiemu, który złamał szyfr Vigenère'a w 1863 roku. Historycy tłumaczą to dwojako: w zwyczaju Babbage'a było niekończenie prac, nie opublikował on więc swoich wyników, ale niektórzy twierdzą, że jego osiągnięcie utajnił wywiad brytyjski w związku z trwającą Wojną Krymską. Z kolei Alan Turing, w czasie II Wojny Światowej uczestniczył m.in. w pracach wywiadu brytyjskiego nad łamaniem szyfrów niemieckich, tworzonych za pomocą maszyny Enigma, bazował przy tym na osiągnięciach Polaków. Prawo w Wielkiej Brytanii chroni jednak informacje wywiadowcze przez przynajmniej 30 lat – świat dowiedział się więc o jego osiągnięciach dopiero w 1974 roku, gdy nie żył on już od ponad 20 lat. Okazało się wtedy również, że już w 1943 roku Brytyjczycy skonstruowali komputer Colossus oparty na przekaźnikach, by łamać najbardziej tajne szyfrogramy Hitlera. Colosuss powstał zatem dwa lata wcześniej niż maszyna ENIAC, uważana powszechnie za pierwszy komputer elektroniczny" [8].

Wraz z ekspansją komputerów i Internetu, dane i informacje są coraz powszechniej przechowywane i przekazywane w postaci elektronicznej. By nie miały do nich dostępu nieupoważnione osoby, szyfrowane są zarówno dane przechowywane w komputerach, jak i tym bardziej dane i informacje przekazywane drogą elektroniczną, np. poczta elektroniczna, rozmowy telefoniczne (rozmowy prowadzone za pomocą komórek są kierowane tylko do wybranych odbiorców, inni nie mogą ich usłyszeć) czy operacje bankowe wykonywane z automatów bankowych lub w formie płatności elektronicznych.

Obecnie kryptografia obejmuje znacznie szerszy zakres zagadnień niż tradycyjnie rozumiane szyfrowanie wiadomości i zajmuje się również identyfikacją użytkowników sieci oraz podpisem elektronicznym i znajduje zastosowanie przy kontroli dostępu użytkowników do stron i serwisów internetowych, do banków, w realizacji płatności elektronicznych przy zakupach internetowych, przy zabezpieczaniu prawa do własności intelektualnych i w wielu jeszcze innych obszarach komunikacji z wykorzystaniem sieci komputerowych, a ogólniej – technologii informacyjno-komunikacyjnych.

1.1 KOMUNIKACJA, SZYFRY I ICH RODZAJE

Ogólny schemat sytuacji, w której pojawia się konieczność szyfrowania, jest przedstawiony na rysunku 1. Alicja komunikuje się z Bogdanem lub Bogdan z Alicją, ale rozmowa może być podsłuchiwana przez ciekawską Ewę. Dalej będziemy używać tych imion w opisie sytuacji komunikacyjnej. Poczyńmy także uproszczenie, że interesować nas będzie bezpieczne przesyłanie wiadomości od Alicji do Bogdana. Przesyłanie wiadomości w drugą stronę odbywa się identycznie.



Rysunek 1.
Komunikowanie się Alicji z Bogdanem i ciekawska Ewa w pobliżu

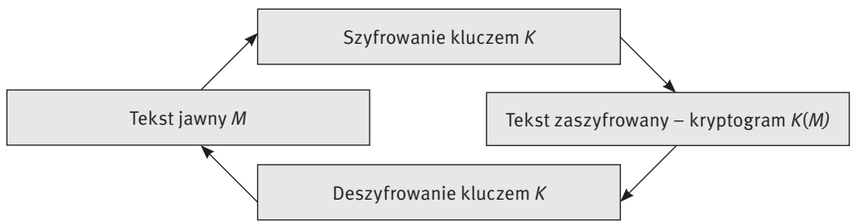
Aby Ewa nie mogła poznać wiadomości przesyłanych przez Alicję do Bogdana, muszą oni w jakiś sposób utajniać wysyłane wiadomości, czyli je **szyfrować**. **Szyfr**, to ustalony sposób utajniania (czyli **szyfrowania**) wiadomości, który na ogół polega na zastępowaniu tekstu wiadomości innym tekstem, z którego trudno domyśleć się znaczenia tekstu oryginalnego. Wiadomość, którą utajniamy określa się mianem **tekstu jawnego**, a jego zaszyfrowaną wersję – **kryptogramem**. Odczytywanie tekstu jawnego z szyfrogramu nazywa się **deszyfrowaniem**.

Tekst można szyfrować na dwa sposoby:

- przestawiając tylko znaki – jest to szyfrowanie **przez przestawianie**, czyli tzw. **szyfr przestawieniowy**;
- albo zastępując znaki tekstu innymi znakami – jest to szyfrowanie **przez podstawianie**, czyli tzw. **szyfr podstawieniowy**.

Najpowszechniej są stosowane **metody podstawieniowe**, które polegają na zamianie znaków (liter) innymi znakami (literami), czyli na podstawianiu za nie innych liter. A zatem, litery tworzące wiadomość nie zmieniają swojego miejsca w wiadomości, tylko zmieniają swoje znaczenie.

Istniejące i stosowane w praktyce metody szyfrowania są na ogół dobrze znane – na czym więc polega utajnianie wiadomości? Nieujawnianym elementem metod szyfrowania jest ich **klucz**, który służy do „otwarcia" utajnionej wiadomości. Na rysunku 2 przedstawiono bardzo ogólny schemat szyfrowania z kluczem *K*. Zauważmy, że ten sam klucz *K* jest wykorzystany do deszyfrowania wiadomości – jest to tak zwana **symetryczna metoda szyfrowania**. W dalszej części omówimy również **metodę asymetryczną**, w której inny klucz jest użyty do szyfrowania, a inny do deszyfrowania (patrz rozdz. 8).



Rysunek 2.
Schemat metody szyfrowania i deszyfrowania

1.2 KODOWANIE JAKO SZYFROWANIE

Jedną z odmian szyfrowania metodą podstawiania jest **kodowanie** wiadomości. Kodowanie polega na ogół na zastępowaniu znaków tworzących oryginalną wiadomość innymi znakami lub zbiorami znaków według jednoznacznego przepisu, danego zazwyczaj w postaci **książki kodowej**, którą można uznać za klucz w tej metodzie. Przykładami takiej „książki" jest kod ASCII, który służy do zamiany znaków na bajty, oraz alfabet Morse'a, w którym znaki są zastępowane ciągami kropek i kresek. Kodowanie może służyć ukrywaniu znaczenia wiadomości, a w kryptografii komputerowej przedstawianie wiadomości w kodzie ASCII jest pierwszym etapem szyfrowania wiadomości.

W czasie II wojny światowej w wojsku amerykańskim był stosowany kod Nawahów, w którym każdej literze alfabetu odpowiadało słowo z języka tego plemienia Indian. W oddziałach zostali również zatrudnieni łącznościowcy, wywodzący się tego plemienia, którzy jako jedyni byli w stanie szyfrować i deszyfrować wiadomości przesyłane za pomocą słów z ich języka.

2 POCZĄTKI KRYPTOGRAFII

Zanim przedstawimy wybrane aspekty współczesnej kryptografii, opiszemy kilka sposobów utajniania wiadomości, stosowanych w przeszłości. Osobom zainteresowanym szerszym omówieniem historii kryptografii polecamy bardzo ciekawie napisaną książkę Simona Singha, *Księga szyfrów* [8].

2.1 STEGANOGRAFIA

Terminem **steganografia** określa się ukrywanie przekazywanych wiadomości. Pierwsze wzmianki na ten temat znajdujemy w historii zmagań między Grecją a Persją w V w. p.n.e., opisanych przez sławnego historyka starożytności, Herodota. Steganografia przybierała różne formy, np. wiadomość zapisywano na cienkim jedwabiu, robiono z niego kulkę i po oblaniu jej woskiem goniec ją połykał. Inną formą steganografii jest stosowanie sympatycznego atramentu – tekst nim zapisany znika, ale pojawia się na ogół po podgrzaniu papieru lub potraktowaniu go specjalną substancją. Sławne stało się zapisanie wiadomości na ogolonej głowie wojownika i wysłanie go z tą wiadomością, gdy odrosły mu włosy.

Steganografia ma podstawową wadę – przechwycenie wiadomości jest równoznaczne z jej ujawnieniem. Od pojawienia się łączności drogą bezprzewodową za pomocą fal radiowych zaczęła mieć marginalne znaczenie w systemach łączności. Obecnie czasem odżywana dzięki możliwościom miniaturyzacji – możliwe jest ukrycie nawet znacznej wiadomości, dodatkowo wcześniej zaszyfrowanej, np. w... kropce, umieszczonej w innym tekście.

Polecamy bardzo ciekawy artykuł na temat steganografii, zamieszczony w czasopiśmie „Wiedza i Życie" 6/2002, s. 38-42, oraz strony w Internecie poświęcone ukrywaniu wiadomości. Obecnie wraca popularność steganografii wykorzystywanej do ukrywania wiadomości, np. znaków tekstu, w „wolnych miejscach" innych plików lub fragmentów wiadomości w postaci niewidocznego tła dokumentu.

W praktyce, dla zwiększenia bezpieczeństwa stosuje się steganografię w połączeniu z kryptografią, czyli ukrywa się przesyłanie utajnionych wiadomości.

2.2 SZYFR CEZARA

Jednym z najstarszych znanych szyfrów podstawieniowych jest **szyfr Cezara**, pochodzący z I w. p.n.e. Polega on na zastąpieniu każdej litery tekstu jawnego literą położoną w alfabecie o trzy miejsca dalej. Liczba 3 jest więc **kluczem** dla klasycznego szyfru Cezara. Kluczem w tym sposobie szyfrowania może być jednak dowolna liczba między 1 a długością alfabetu, o którą przesuwamy każdą literę alfabetu.

W przypadku szyfrów podstawieniowych można mówić o **alfabecie jawnym**, w którym są zapisywane wiadomości, i o **alfabecie szyfrowym**, w którym są zapisywane utajniane wiadomości. Dla ułatwienia umieszczamy te alfabety jeden pod drugim i tekst jawny będziemy pisać małymi literami, a zaszyfrowany – wielkimi. A zatem w przypadku szyfru Cezara mamy (oba alfabety zawierają również polskie litery):

Alfabet jawny **a ą b c ć d e ę f g h i j k l ł m n ń o ó p q r s ś t u v w x y z ż ź**
Alfabet szyfrowy **C Ć D E Ę F G H I J K L Ł M N Ń O Ó P Q R S Ś T U V W X Y Z Ż Ź A Ą B**

Polecamy zaszyfrowanie powiedzenia Cezara: **veni! vidi! vici!** stosując jego szyfr[19]. Jakie przysłowie łacińskie zaszyfrowano w ten sposób w następującym kryptogramie:

TGSGWLWLQ GUW OCWGT UWXFLQTXO – SQZWCTĄCÓLG ŁGUW OCWMĆ XEAĆEŹEK ULH

Wiedząc, że szyfrowano tekst z przesunięciem 3, łatwo można odczytać utajnioną wiadomość, gdyż jest to operacja odwrotna – litery szyfrogramu znajdujemy w alfabecie szyfrowym, a ich odpowiedniki w alfabecie jawnym. Szyfr Cezara jest więc szyfrowaniem z **kluczem symetrycznym**, gdyż znajomość klucza nadawcy wystarcza do określenia klucza odbiorcy. Niestety, tak zaszyfrowany tekst może rozszyfrować osoba nieuprawniona po przechwyceniu zaszyfrowanej wiadomości wiedząc tylko, że nadawca zastosował szyfr Cezara.

Klasyczny szyfr Cezara można uogólnić, dopuszczając jako alfabet szyfrowy przesunięcie alfabetu jawnego o dowolną liczbę znaków – mamy więc 34 możliwe alfabety w przypadku alfabetu języka polskiego. To jednak nadal niewielkie utrudnienie i nawet mało wprawiony **kryptolog**, czyli „łamacz szyfrów", po przechwyceniu kryptogramu zaszyfrowanego uogólnionym szyfrem Cezara łatwo określi, jaki był alfabet szyfrowy. Spróbuj i Ty swoich sił i rozszyfruj przysłowie łacińskie, które zostało ukryte w następującym kryptogramie:

RSBCXASK ESCKO VKQSBCAK OBC

otrzymanym za pomocą uogólnionego szyfru Cezara dla alfabetu łacińskiego. *Uwaga*. Alfabet łaciński nie zawiera polskich liter, nie zawiera również litery J, a więc składa się z 25 liter.

2.3 SZYFR PLAYFAIRA

Przedstawimy tutaj szyfr, zwany **szyfrem Playfaira**, który w połowie XIX wieku wymyślił Charles Wheatstone, jeden z pionierów telegrafu, a spopularyzował Lyon Playfair. Jest to przykład szyfru podstawieniowego, w którym pary różnych liter tekstu jawnego są zastępowane inną parą liter. Aby móc stosować ten szyfr, nadawca i odbiorca muszą jedynie wybrać słowo kluczowe. Przypuśćmy, że jest nim słowo **WYPAD**. Na podstawie słowa kluczowego jest tworzona tabliczka – patrz tabela 1 – złożona z 25 (5 na 5) pól, służąca do szyfrowania i deszyfrowania wiadomości. Umieszczamy w niej najpierw słowo kluczowe, a następnie pozostałe litery alfabetu łacińskiego (**I** oraz **J** w jednym polu; i w tekstach zaniedbujemy znaki diakrytyczne w polskich literach).

Tabela 1.
Przykładowa tabliczka-klucz dla metody szyfrowania Playfaira

W	Y	P	A	**D**
B	C	E	F	G
H	I/J	K	L	M
N	**O**	Q	R	S
T	U	V	X	Z

Tekst jawny przed zaszyfrowaniem dzielimy na pary różnych liter – takie same litery przedzielamy np. literą **x**, i ostatnią pojedynczą literę również uzupełniamy do pary literą **x**.

Tekst jawny: **do zobaczenia o szóstej** przyjmuje więc przed szyfrowaniem postać: **do-zo-ba-cz-en-ia--os-zo-st-ej**. A oto reguły zastępowania par liter:

- jeśli obie litery znajdują się w tym samy wierszu, np. **os**, to zastępujemy je sąsiednimi literami z prawej strony; jeśli jedna z liter znajduje się na końcu wiersza, w naszym przypadku jest to **s**, to zastępujemy ją

[19] Dla uproszczenia, znaki inne niż litery (np. cyfry, znaki interpunkcyjne) pomijamy w kryptogramach, z wyjątkiem odstępów w dłuższych tekstach.

pierwszą literą wiersza; a zatem w tym przypadku **os** zastępujemy parą **QN**;
- jeśli obie litery znajdują się w tej samej kolumnie, to zastępujemy je literami leżącymi bezpośrednio poniżej; jeśli jedna z liter znajduje się na końcu kolumny, to zastępujemy ją pierwszą literą kolumny;
- jeśli obie litery nie znajdują się ani w tym samym wierszu, ani w tej samej kolumnie, to zastępujemy je parą według schematu pokazanego powyżej, w odniesieniu do pary **do** – zastępujemy ją parą **YS**; w obu przypadkach, polega to na wzięciu litery znajdującej się w tym samym wierszu i w kolumnie, w której znajduje się druga litera.

Ostatecznie, tekst **do zobaczenia o szóstej** zostanie zaszyfrowany jako:

YS-US-FW-GU-BQ-LY-QN-US-NZ-CK

A jaki tekst jawny został zaszyfrowany dla tego samego klucza w następującym kryptogramie:

KCRMKHDRUGNBKMBVNYCOGDRWQDYFFQVDQGLZ

3 SZYFROWANIE PRZEZ PRZESTAWIANIE

Na chwilę tylko zatrzymamy się przy szyfrowaniu przez przestawianie, gdyż jest to mało popularny sposób utajniania wiadomości, chociaż przestawienie liter w tekście wiadomości jest jednym z najprostszych sposobów zmiany znaczenia wiadomości. Przestawiając litery w słowie możemy otrzymać inne słowo, zwane jego **anagramem**. A zatem taki sposób szyfrowania dłuższego tekstu jest uogólnieniem anagramu.

Znajdź dwa anagramy „słowa" **AGLMORTY**[20], które wielokrotnie pojawiają się na zajęciach z tego działu informatyki w projekcie Informatyka +. Czy nie dziwi Cię, że są one swoimi anagramami?

Jeśli w danym tekście możemy dowolnie przestawiać litery, to otrzymujemy olbrzymią liczbę możliwych kryptogramów. Przypuśćmy, że chcemy zaszyfrować słowo **szyfrowanie** przestawiając tylko litery tego słowa. Wszystkie możliwe przestawienia liter tego słowa można otrzymać budując słowa litera po literze. W przypadku słowa **szyfrowanie** mamy do dyspozycji litery ze zbioru {**a, e, f, i, n, o, r, s, w, y, z**}. Aby utworzyć słowo z liter tego zbioru (słowo to nie musi mieć znaczenia), najpierw wybieramy pierwszą literę (spośród 11), następnie określamy drugą literę, wybierając ją spośród 10 pozostałych, dla trzeciej litery w słowie mamy 9 możliwości itd. Zatem wszystkich możliwości jest: 11*10*9*...*2*1 = 39916800. Ten iloczyn oznaczamy przez 11! – jest to funkcja zwaną **silnią**. Ogólnie mamy:

$$n! = 1*2*3* \ldots*(n-1)*n.$$

Tabela 2.
Wartości funkcji *n*! dla przykładowych *n*

n	*n*!
5	120
15	$1{,}3*10^{12}$
25	$1{,}5*10^{25}$
35	$1{,}0*10^{40}$
50	$3{,}0*10^{64}$

W tabeli 2 są podane wartości funkcji silnia dla kilku wartości *n*. Dla porównania, liczba protonów we wszechświecie jest szacowana na 10^{79}, a najszybsze obecnie komputery są w stanie wykonać w ciągu sekundy około 10^{15} operacji, a więc ta funkcja przyjmuje bardzo duże wartości i szybko rośnie ze wzrostem *n*.

[20] Zgodnie z przyjętą umową, wiadomości utajnione zapisujemy wielkimi literami, a wiadomości jawne – małymi literami.

Aby odczytać kryptogram, będący anagramem tekstu jawnego, należałoby wziąć pod uwagę wszystkie możliwe przestawienia liter. Jak pokazują liczby w tabeli 2, nawet dla tekstów o niewielkiej długości, bez pewnych reguł przestawiania będzie to szyfr tak samo trudny dla osoby przechwytującej kryptogram, jak i dla odbiorcy zaszyfrowanej wiadomości. Jedną z reguł tworzenia anagramu może być np. przestawianie liter na ustalonej pozycji.

Poniższy kryptogram został otrzymany za pomocą przestawienia każdych dwóch kolejnych liter w tekście jawnym poczynając do drugiej litery (zaniedbano odstępy między słowami).

WAHITILEKODNIBGSEITNSTOIHGN

Odczytaj tekst jawny (jest po angielsku). Czy pamiętasz, kto wypowiedział te słowa?

Przestawienia mogą dotyczyć bitów w dwójkowej reprezentacji tekstu jawnego. Można zaproponować na przykład następujący algorytm szyfrowania:

1. Zapisz wiadomość w kodzie ASCII.
2. Przestaw ze sobą co ósmą parę bitów poczynając od pary na pozycjach 2 i 3, a więc zamieniając ze sobą miejscami bity 2 i 3, 10 i 11, 18 i 19 itd. – w tym kroku definiuje się klucz tego sposobu szyfrowania.
3. Powróć, stosując kod ASCII, do wiadomości w postaci ciągu znaków.

Metoda płotu

Bardzo prosty szyfr przestawieniowy otrzymujemy stosując tzw. **metodę płotu**. W tej metodzie, najpierw kolejne litery tekstu jawnego są zapisywane na zmianę w dwóch rzędach, a następnie za kryptogram przyjmuje się ciąg kolejnych liter z pierwszego rzędu, po którym następuje ciąg kolejnych liter z drugiego rzędu. Na przykład, metoda płotu zastosowana do słowa **szyfrowanie** daje następujący kryptogram:

s y r w n e
z f o a i
SYRWNEZFOAI

„Płot” w tej metodzie szyfrowania może się składać z więcej niż dwóch rzędów, np. kryptogram powyższej wiadomości jawnej, otrzymany z użyciem trzech rzędów w „płocie”, ma postać: **SFWIZRAEYON**,

Dla dociekliwych i wytrwałych uczniów mamy następujące zadanie: Wiadomo, że poniższy kryptogram (powiedzenie premiera brytyjskiego Winstona Churchilla do szefa Secret Intelligence Service, 1941):

PIIŚIYOLSŁWETSIOŁŻZŁLŻEĄEESAŁ
ELEERCKNNDMŹZKOEMBOOOAIZŻMTDNCCYBTMAEIEIOON

powstał z tekstu jawnego za pomocą metody płotu. Nie wiadomo jednak, z ilu rzędów składał się „płot”. Rozszyfruj tę wiadomość. Podpowiemy tylko: zauważ, że umieszczając tekst jawny w kilku rzędach płotu, każdy rząd płotu zawiera taką samą liczbę liter, z wyjątkiem kilku ostatnich rzędów, krótszych o jeden znak.

4 SZYFROWANIE Z ALFABETEM SZYFROWYM

Wracamy tutaj do szyfrowania z alfabetem szyfrowym – taką metodą jest opisany wcześniej szyfr Cezara. Za alfabet szyfrowy, zamiast przesuwania alfabetu o ustaloną liczbę liter, można wziąć jakiekolwiek uporządkowanie liter alfabetu. Takich uporządkowań jest jednak 35!, co jest olbrzymią liczbą. Niepowołana osoba ma więc małe szanse natrafić przypadkowo na wybrany z tej liczby alfabet szyfrowy. Jednak wybór losowego uporządkowania liter w alfabecie szyfrowym ma wadę. Alfabet ten trzeba w jakiś tajny sposób przekazać osobie, która ma odczytywać tworzone nim kryptogramy. Dlatego zamiast losowego wyboru alfabetu szyfrowego stosuje się różne rodzaje kluczy, które precyzyjnie określają alfabet szyfrowy.

Jednym ze sposobów określania alfabetu szyfrowego jest podanie **słowa** (lub **powiedzenia**) **kluczowego**, na podstawie którego tworzy się alfabet szyfrowy. Dla przykładu użyjmy **Cappadocia** jako słowo kluczowe do utworzenia alfabetu szyfrowego. Najpierw umieszczamy słowo kluczowe na początku alfabetu szyfrowego i usuwamy z niego powtarzające się litery – w naszym przykładzie pozostaje więc **CAPDOI**. Następnie, dopisujemy pozostałe litery w porządku alfabetycznym, zaczynając od litery następnej po ostatniej literze w pozostałej części słowa kluczowego (w przykładzie od litery **J**) i kontynuując od początku alfabetu. Otrzymujemy:

Alfabet jawny	**a ą b c ć d e ę f g h i j k l ł m n ń o ó p q r s ś t u v w x y z ż ź**
Alfabet szyfrowy	**C A P D O I J K L Ł M N Ń Ó Q R S Ś T U V W X Y Z Ż Ź Ą B Ć E Ę F G H**

Spróbuj odczytać, jaką wiadomość zaszyfrowano w poniższym tekście, posługując się szyfrem z kluczem **Cappadocia** (dla ułatwienia pozostawiliśmy w tekście odstępy i znaki interpunkcyjne):

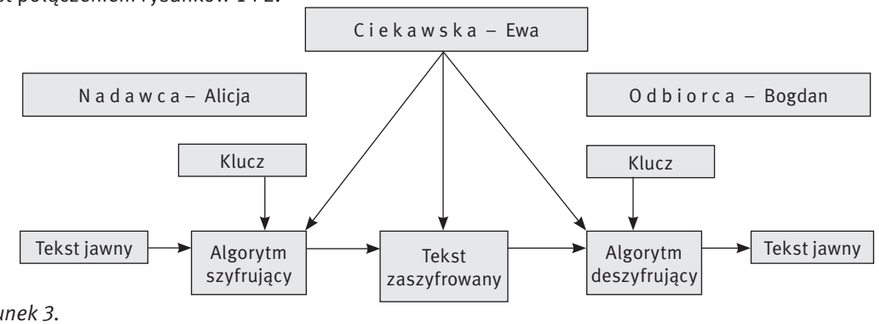
ÓCWCIUDŃC ŹU WYFJWNKÓŚC ÓYCNŚC ŚC ĆĘGĘŚNJ CŚCŹUQNŃZÓNJŃ Ć ŻYUIÓUĆJŃ
ŹĄYDŃN, UPLNŹĄŃADC Ć LCŚŹCZŹĘDFŚJ LUYSĘ ZÓCQŚJ, ĆĘIYAGUŚJ Ć ZÓCRCDM
ÓUŻDNURĘ N ÓQCZFŹUYĘ, IŃSĘ SNJZFÓCQŚJ, DCRJ WUIFNJSŚE SNCZŹC. ŹCS WUĆZŹCR
ŹJŚ LYCŁSJŚŹ WUIYKDFŚNÓC U ZFĘLYUĆCŚNĄ.

W czasie II wojny światowej w polskim ruchu oporu stosowano alfabety szyfrowe, określane na podstawie ustalonych fragmentów wybranych książek. Na przykład za klucz można przyjąć siedem pierwszych słów z pierwszego akapitu na 267 stronie *Potopu*, wydanego w 1923 roku we Lwowie (obie komunikujące się strony musiały posiadać dokładnie tę samą książkę), a w naszym przypadku może to być na przykład pierwsze zdanie ze streszczenia tych materiałów – podaj, jaki to będzie klucz.

Tworzenie alfabetów szyfrowych na podstawie słów kluczowych znacznie utrudnia deszyfrację kryptogramów, gdyż możliwych jest bardzo wiele słów (a ogólniej tekstów) kluczowych.

5 SCHEMAT KOMUNIKACJI Z SZYFROWANIEM

Podsumujemy tutaj krótko to, co dotychczas powiedzieliśmy o utajnianiu przesyłanych wiadomości. Rysunek 3 jest połączeniem rysunków 1 i 2.



Rysunek 3.
Schemat utajnionego przekazywania wiadomości

Przykładowe metody szyfrowania i deszyfrowania wiadomości, przedstawione w poprzednich punktach, można opisać schematycznie jak na rysunku 3. W tych metodach można wyróżnić następujące elementy:

- **algorytm szyfrowania**, który służy do zamiany tekstu jawnego na tekst zaszyfrowany (kryptogram) i **algorytm deszyfrujący**, służący do odszyfrowania kryptogramu na tekst jawny;

- **klucz**, który jest w pewnym sensie parametrem algorytmu szyfrującego i deszyfrującego, niezbędną informacją, w jaki sposób mają działać algorytmy szyfrujący i deszyfrujący.

We współczesnej kryptografii przyjmuje się założenie, że algorytmy szyfrowania są znane, natomiast ukryty jest klucz do algorytmów szyfrowania i deszyfrowania, a zatem Ciekawska Ewa zna sposób utajniania wiadomości, które przesyłają między sobą Alicja i Bogdan, ale nie zna klucza, który jest używany przez nich w algorytmach szyfrujących i deszyfrujących. Jest to odzwierciedleniem podstawowego założenia, jakie przyjmuje się obecnie w kryptografii:

> bezpieczeństwo systemu szyfrowania jest oparte na kluczach, a nie na sposobie szyfrowania.

Przypomnijmy więc sobie, jakie były klucze w poznanych algorytmach szyfrowania:

- szyfr Cezara – 3 lub inna liczba, o którą przesuwa się alfabet jawny, by otrzymać alfabet szyfrowy;
- metoda płotu – liczba rzędów w płocie;
- szyfr Playfaira – słowo kluczowe, które służy do zbudowania tabliczki zamiany par liter;
- szyfr z jednym alfabetem szyfrującym, tzw. **szyfr monoalfabetyczny** – słowo kluczowe (lub tekst kluczowy), które służą do zbudowania alfabetu szyfrowego.

Zauważmy, że we wszystkich tych przypadkach ten sam klucz służy do szyfrowania wiadomości i do deszyfrowania kryptogramów. Stwarza to pewne problemy z przekazywaniem klucza, co musi być czynione z wielką ostrożnością, by klucz nie wpadł w ręce Ewy.

Podane przez nas przykłady użycia powyższych algorytmów pokazują, że znalezienie klucza w przypadku pierwszego i drugiego algorytmu nie jest specjalnie trudne, zwłaszcza jeśli dysponujemy komputerem. W dwóch pozostałych przypadkach nie jest to już jednak takie proste, gdyż istnieją nieograniczone możliwości doboru słów kluczowych. Jednak i te szyfry udało się złamać.

Łamaniem szyfrów, czyli odczytywaniem utajnionych wiadomości bez znajomości wszystkich elementów sposobu szyfrowania (np. bez znajomości klucza), zajmuje się **kryptoanaliza**. Kryptografia i kryptoanaliza to dwa główne działy **kryptologii**, nauki o utajnionej komunikacji. Kryptografia i kryptoanaliza rywalizują ze sobą od początków utajniania wiadomości – kryptografia tworzy coraz doskonalsze (bezpieczniejsze) szyfry, a kryptoanaliza dostarcza metod ich łamania. Jak pokazaliśmy, złamanie uogólnionego szyfru Cezara jest dość proste – należy jedynie ustalić, o ile pozycji przesuwają się wszystkie litery. Podobnie jest ze złamaniem szyfru płotowego – należy określić, jak wysoki jest płot, czyli ile ma rzędów. W następnym punkcie opowiemy o łamaniu szyfrów z jednym alfabetem szyfrowym i o pewnym uogólnieniu tej metody.

6 SZYFR POLIALFABETYCZNY

Wracamy tutaj do sposobu szyfrowania z alfabetem szyfrującym, gdyż interesuje nas, czy można złamać taki szyfr z alfabetem szyfrowania, utworzonym przez dowolne słowo kluczowe. Jeśli tak, to na pewno nie jest to metoda sprawdzająca wszystkie możliwe alfabety szyfrowania, gdyż jest ich zbyt dużo. Pierwszy zapis o skutecznym sposobie łamania szyfru z alfabetem szyfrowania pochodzi z IX wieku. Jego autorem był Al-Kindi, zwany „filozofem Arabów". Jako pierwszy wykorzystał on różnice w częstości występowania liter w tekstach i zastosował tzw. **metodę częstościową**. Wyjaśnimy krótko, na czym polega ta metoda.

W tabeli 3 są podane częstości występowania liter w tekstach w języku angielskim i w języku polskim. W tekstach angielskich zdecydowanie najczęściej występuje litera **e**, a następnie litery **t**, **a**, **o**, **i**. W tekstach w języku polskim natomiast żadna litera nie występuje tak często jak litera **e** w tekstach angielskich, a z częstością ponad 7% występują litery: **a**, **i**, **o**, **e**. Stąd można wywnioskować, że litera, która występuje najczęściej w kryptogramie w języku polskim powinna odpowiadać literze **a** w tekście jawnym. Oczywiście odnosi się to do nieco dłuższych tekstów, a więc w krótszych kryptogramach ta najczęściej występująca w nich litera może odpowiadać którejś z następnych co do częstości pojawiania się w tekstach liter. Ponadto, szukając par odpowiadających sobie liter uwzględnia się charakterystyczne dla danego języka połączenia dwóch lub więcej liter, np. w języku polskim dość często występują pary liter: **rz**, **sz**, **cz**, **ść**, a w języku angielskim – **qu**, **th**. Sposób łamania szyfru, czyli deszyfracji wiadomości, polegający na wykorzystaniu częstości występowania liter i ich kombinacji w tekście nazywa się **analizą częstości**.

Tabela 3.
Częstości występowania liter w tekstach w języku angielskim i polskim

Litery	Częstość występowania liter w tekstach (w %)	
	w języku angielskim	w języku polskim
A	8,1	8,71
B	1,5	1,29
C	2,8	3,91
D	4,3	3,45
E	12,6	7,64
F	2,2	0,38
G	2,0	1,42
H	6,1	1,22
I	7,0	8,48
J	0,2	2,38
K	0,8	3,10
L	4,0	2,09
M	2,4	2,64
N	6,7	5,60
O	7,5	7,90
P	1,9	3,01
Q	0,1	–
R	6,0	4,63
S	6,3	4,64
T	9,0	3,63
U	2,8	1,92
V	1,0	–
W	2,4	4,62
X	0,2	–
Y	2,0	3,88
Z	0,1	5,95

Jeśli policzymy częstości poszczególnych liter w kryptogramie podanym powyżej dla słowa kluczowego Cappadocia to się okaże, że odpowiedniki pięciu najczęściej występujących liter w tym kryptogramie plasują się na pierwszych (pod względem częstości) sześciu pozycjach w ostatniej kolumnie tabeli 3.

Posłużenie się analizą częstości przy deszyfrowaniu tekstu nie jest tylko prostym skorzystaniem z tabeli częstości liter, ale żmudną analizą możliwych przypadków, w której nierzadko trzeba postępować metodą prób i błędów, czyli próbować różnych przyporządkowań liter i nieraz się z nich wycofywać. Procesu deszyfracji, wykorzystującego analizę częstości, nie daje się prosto zautomatyzować za pomocą komputera. Jest to postępowanie w dużym stopniu interaktywne.

Po złamaniu monoalfabetycznego szyfru podstawieniowego, czyli z jednym alfabetem szyfrowym, nastąpił ruch kryptografów, którzy zaproponowali szyfr z wieloma alfabetami szyfrowymi – taki szyfr nazywa się **polialfabetycznym**. Uniemożliwia on użycie prostej analizy częstości, gdyż w tworzonych kryptogramach za daną literę może być podstawianych wiele różnych liter. Słowo kluczowe w tym przypadku służy do określenia alfabetów – są nimi alfabety Cezara wyznaczone przez kolejne litery słowa kluczowego. Dla przykładu, niech słowem kluczowym będzie **BRIDE** (dla utrudnienia kryptoanalizy, słowa kluczowe są wybierane z innych języków), wtedy mamy pięć alfabetów szyfrowych, zaczynające się od liter **B**, **R**, **I**, **D** i **E**.

Alfabet jawny	**a ą b c ć d e ę f g h i j k l ł m n ń o ó p q r s ś t u v w x y z ż ź**
Alfabety szyfrowe	**B C Ć D E Ę F G H I J K L Ł M N Ń O Ó P Q R S Ś T U V W X Y Z Ż Ź A Ą**
	R S Ś T U V W X Y Z Ż Ź A Ą B C Ć D E Ę F G H I J K L Ł M N Ń O Ó P Q
	I J K L Ł M N Ń O Ó P Q R S Ś T U V W X Y Z Ż Ź A Ą B C Ć D E Ę F G H
	D E Ę F G H I J K L Ł M N Ń O Ó P Q R S Ś T U V W X Y Z Ż Ź A Ą B C Ć
	E Ę F G H I J K L Ł M N Ń O Ó P Q R S Ś T U V W X Y Z Ż Ź A Ą B C Ć D

Zaszyfrujmy teraz wiadomość **panna w opałach!**: pierwszą literę jawnego tekstu szyfrujemy posługując się pierwszym alfabetem szyfrowym, drugą – drugim, trzecią – trzecim, czwartą – czwartym, piątą – piątym, szóstą – pierwszym itd. Aby nie zagubić się w liczeniu, którego kolejnego alfabetu użyć, dobrze jest umieścić powtarzające się słowo kluczowe nad tekstem do zaszyfrowania. W naszym przypadku mamy:

B R I D E B R I D E B R I
p a n n a w o p a ł a c h !

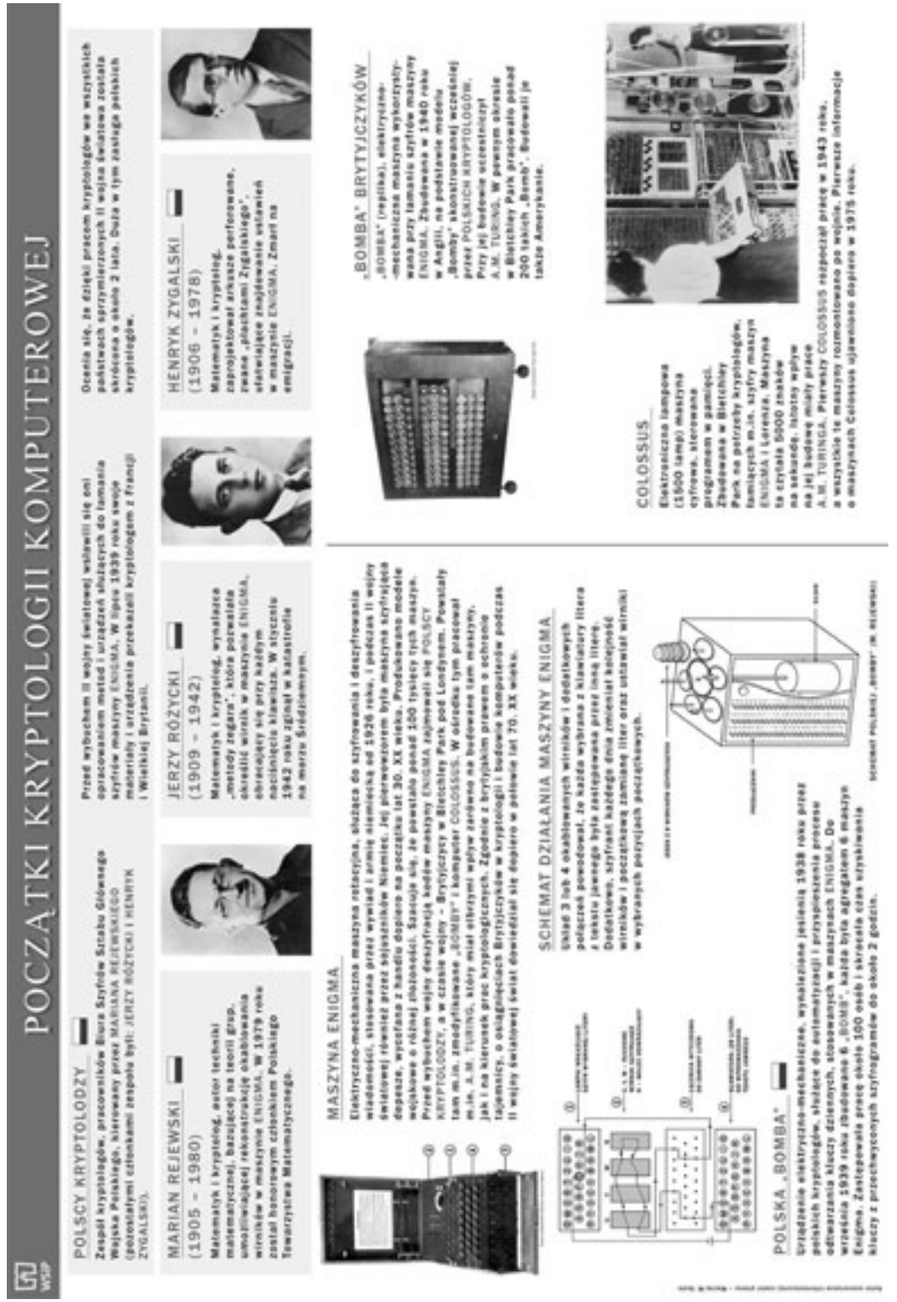
Otrzymujemy więc kryptogram: **RRVQE Y ĘZDPBTP!**, w którym każda z powtarzających się liter w tekście jawnym (**a** i **n**) jest za każdym razem szyfrowana przez inną literę.

Taki sposób szyfrowania został zaproponowany w XVI wieku przez dyplomatę francuskiego Blaise de Vigenère'a i nazywa się **szyfrem Vigenère'a**. Przekonaj się, że szyfrowanie tym sposobem nie przenosi częstości liter z tekstu jawnego na kryptogram. W tym celu zastosuj szyfr Vigenère'a z kluczem **BRIDE** do zaszyfrowania wiadomości jawnej, otrzymanej ze słowem kluczowym Cappadocia i oblicz częstości występowania liter w otrzymanym kryptogramie. Porównaj te częstości z częstościami liter w kryptogramie otrzymanym za pomocą jednego alfabetu z tym samy słowem kluczowym **BRIDE** oraz z częstościami ich odpowiedników w tekście jawnym (patrz tabela 3).

Dopiero w połowie XIX wieku kryptoanalitycy złamali szyfr Vigenère'a – jednym z ich był Charles Babbage, o czym wspominamy na początku tych materiałów. W odpowiedzi na złamanie szyfru Vigenère'a, kryptografowie zaproponowali jednoczesne szyfrowanie par liter – bardzo prosty szyfr tego rodzaju przedstawiliśmy już wcześniej, to **szyfr Playfaira**.

7 PRZEŁOM W KRYPTOGRAFII – ENIGMA

W latach dwudziestych pojawiła się pierwsza maszyna szyfrująca – **Enigma** – zastąpiła ona algorytm szyfrujący i deszyfrujący ze schematu na rysunku 3, a kluczem w tej maszynie było początkowe ustawienie jej elementów mechanicznych. Na początku maszynę tę można było nawet kupić na wolnym rynku. Szyfr Enigmy został złamany przez zespół polskich matematyków, którym kierował **Marian Rejewski**. Zbudowali oni maszynę deszyfrującą, zwaną **Bombą**, która składała się z wielu kopii Enigmy. Swoje osiągnięcia i pomysły wraz z tą maszyną Polacy przekazali jeszcze przed wybuchem II wojny światowej Brytyjczykom, którzy w Bletchley Park koło Londynu zapoczątkowali erę komputerowej kryptografii i kryptoanalizy. Na rysunku 4 przedstawiono krótką historię początków kryptografii komputerowej. Więcej na ten pasjonujący temat szyfrów maszyny Enigma można znaleźć w książkach: [2], [5], [6], [8].



Rysunek 4.
Początki kryptografii komputerowej w zarysie [ta i inne plansze dostępne są na stronie http://mmsyslo.pl/]

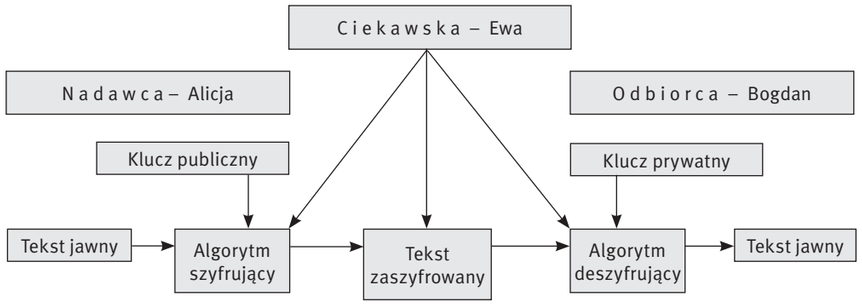
8 KRYPTOGRAFIA Z JAWNYM KLUCZEM

Pojawianie się coraz silniejszych komputerów powoduje realne zagrożenie dla przesyłania utajnionych wiadomości. Kryptoanalityk może skorzystać z mocy komputera, by sprawdzić bardzo dużą liczbę możliwych alfabetów i kluczy oraz prowadzić analizę kryptogramów metodą prób i błędów. Co więcej, z ekspansją komunikacji najpierw telefonicznej, a obecnie – internetowej wzrosła do olbrzymich rozmiarów liczba przesyłanych wiadomości. Państwa, instytucje, a także pojedynczy obywatele chcieliby mieć gwarancję, że system wymiany wiadomości może zapewnić im bezpieczeństwo i prywatność komunikacji.

W 1976 roku przyjęto w USA szyfr **Lucifer** jako standard komputerowego szyfrowania danych **DES** (ang. *Data Encryption Standard*). Liczba możliwych kluczy dla systemu DES jest gwarancją, że praktycznie jest to bezpieczny szyfr – powszechnie dostępne komputery są zbyt słabe, by go złamać. Pozostaje jednak nadal nierozwiązany tzw. **problem dystrybucji klucza** – w jaki sposób dostarczyć klucz odbiorcy utajnionej nim wiadomości. Problem ten dotyczył większości szyfrów w historii kryptografii. Na przykład Marian Rejewski osiągnął pierwsze sukcesy przy deszyfracji Enigmy, korzystając m.in. z faktu, że klucz do zaszyfrowanej wiadomości był dwa razy powtarzany na początku wiadomości, co okazało się słabą stroną kryptogramów Enigmy.

W połowie lat siedemdziesiątych pojawiła się sugestia, że wymiana klucza między komunikującymi się stronami być może nie jest konieczna. Tak zrodził się pomysł szyfru z jawnym kluczem, którego schemat jest przedstawiony na rysunku 5. Od schematu na rysunku 3 różni się tym, że zamiast jednego klucza dla nadawcy i odbiorcy mamy parę kluczy: **klucz publiczny**, zwanym **kluczem jawnym**, i **klucz prywatny**, zwany również **kluczem tajnym**. Jest to więc **szyfr asymetryczny**. Działanie odbiorcy i nadawcy utajnionych wiadomości w tym przypadku jest następujące:

1. Odbiorca wiadomości tworzy parę kluczy: publiczny i prywatny i ujawnia klucz publiczny, np. zamieszcza go na swojej stronie internetowej.
2. Jeśli nadawca chce wysłać zaszyfrowaną wiadomość do odbiorcy, to szyfruje ją jego kluczem publicznym i tak utworzony kryptogram może odczytać jedynie odbiorca posługując się kluczem prywatnym.



Rysunek 5.
Schemat przesyłania wiadomości w kryptografii z jawnym kluczem

Para kluczy – publiczny i prywatny – ma jeszcze tę własność, że znajomość klucza publicznego nie wystarcza nie tylko do odszyfrowania wiadomości nim zaszyfrowanej, ale również nie umożliwia utworzenia klucza prywatnego, który jest niezbędny do odszyfrowania wiadomości. Utworzenie takiej pary kluczy było możliwe dopiero w erze komputerów. Odbiorca może łatwo określić oba klucze z pary, ale nikt poza nim, przy obecnym stanie wiedzy i mocy komputerów nie jest w stanie odtworzyć klucza prywatnego na podstawie klucza publicznego. Ta trudność w odgadnięciu klucza prywatnego polega na tym, że jego znalezienie przez osobę postronną jest związane ze znalezieniem rozwiązania tak trudnego problemu, że żaden istniejący obecnie komputer nie jest w stanie w tym pomóc. Jeśli jednak pojawiłby się taki komputer, to łatwo można zwiększyć

trudność rozwiązania tego problemu przez niewielką modyfikację danych. Nieco szczegółów na ten temat zamieszczamy w rozdziale 10.

Najbardziej znany szyfr z kluczem publicznym opracowali w 1977 roku Ronald **Rivest**, Adi **Shamir** i Leonard **Adleman** z MIT i od inicjałów ich nazwisk pochodzi jego nazwa – **szyfr RSA**. Zilustrujemy jego działanie na niewielkim przykładzie komentując jednocześnie, jak wygląda jego pełna wersja (zob. również rys. 5). W poniższym opisie przyjmujemy, że Bogdan chce otrzymywać utajnione wiadomości od Alicji (patrz rys. 5 i 6), dlatego przygotowuje odpowiednie klucze dla siebie i dla Alicji.

Etap przygotowania kluczy

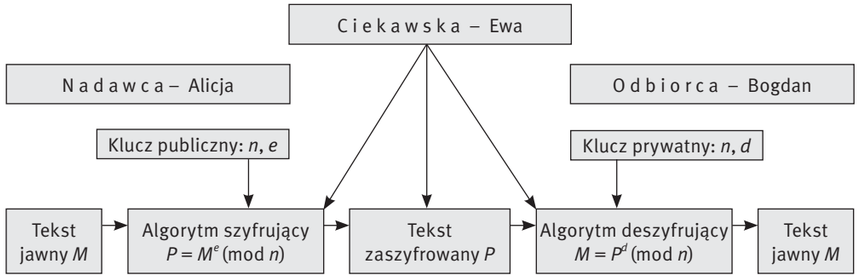
1. Bogdan wybiera dwie duże, możliwie bliskie sobie liczby pierwsze p i q. Liczby te powinny mieć przynajmniej po 200 cyfr. Liczby te Bogdan trzyma w tajemnicy.
   Przykład. Przyjmijmy $p = 11$ i $q = 13$.
2. Bogdan oblicza $n = p*q$ i wybiera dowolną liczbę naturalną e, która jest względnie pierwsza z liczbą $(p - 1)*(q - 1)$, czyli te dwie liczby nie mają wspólnych dzielników poza liczbą 1.
   Przykład. Mamy $n = 11*13 = 143$. Ponieważ $(p - 1)*(q - 1) = 10*12 = 120 = 2^3*3*5$, więc możemy przyjąć $e = 7$.
3. Bogdan znajduje liczbę naturalną d taką, że $e*d = 1 \pmod{(p - 1)*(q - 1)}$, czyli reszta z dzielenia $e*d$ przez $(p - 1)*(q - 1)$ jest równa 1. Liczbę d można znaleźć, posługując się algorytmem Euklidesa (patrz rozdz. 10). [Operacja mod k oznacza wzięcie reszty z dzielenia przez k.]
   Przykład. Mamy znaleźć liczbą naturalną d, spełniającą równość $e*d = 1 \pmod{(p - 1)*(q - 1)}$, czyli $7*d = 1 \pmod{120}$. Taką liczbą jest np. $d = 103$, gdyż $7*103 = 721$ i 721 podzielone przez 120 daje resztę 1.
4. Bogdan ogłasza parę liczb (n, e) jako swój klucz publiczny, np. zamieszcza go na swojej stronie w Internecie, a parę (n, d) zachowuje w tajemnicy jako klucz prywatny. Jednocześnie niszczy liczby p i q, by nie wpadły w niczyje ręce, co mogłoby umożliwić odtworzenie klucza prywatnego. Jeśli p i q są dostatecznie dużymi liczbami pierwszymi, to znajomość n i e nie wystarcza, by obliczyć wartość d posługując się nawet najpotężniejszymi dzisiaj komputerami (patrz rozdz. 10).

Szyfrowanie wiadomości

1. Jeśli Alicja chce wysłać do Bogdana jakąś wiadomość, to najpierw musi ją przedstawić w postaci liczby naturalnej M, nie większej niż n. Tekst można zamienić na liczbę posługując się kodem ASCII. Jeśli wiadomość jest zbyt długa, to należy ją szyfrować blokami odpowiedniej wielkości.
   Przykład. Jako wiadomość wybierzmy literę Q, która w kodzie ASCII ma kod 81. Przyjmujemy więc za wiadomość do wysłania $M = 81$.
2. Wiadomość, jaką Alicja wysyła do Bogdana, jest liczbą: $P = M^e \pmod{n}$. Obliczenie wartości P może wydawać bardzo złożone, w gruncie rzeczy można szybko wykonać przy wykorzystaniu jednej z szybkich metod potęgowania (patrz rozdz. 10) oraz własności operacji na resztach (zob. nasze przykładowe obliczenia).
   Przykład. Mamy obliczyć $P = 81^7 \pmod{143}$. Korzystając z rozkładu wykładnika na postać binarną mamy $81^7 = 81^1*81^2*81^4$. Z każdej z tych potęg obliczamy tylko resztę z dzielenia przez 143. Otrzymujemy stąd: $81^1*81^2*81^4 = 81*126*3 = 16 \pmod{143}$.
3. Alicja wysyła do Bogdana wiadomość $P = 16$.

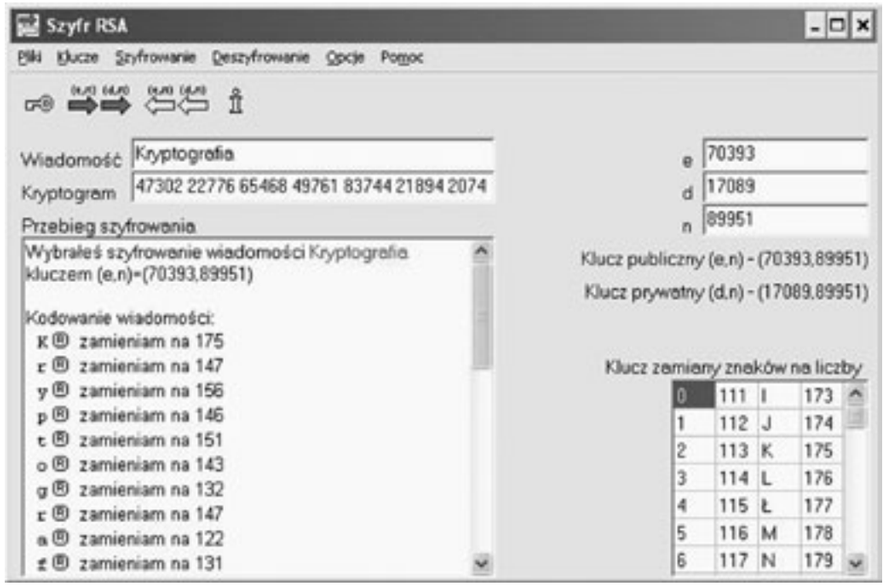
Odszyfrowanie kryptogramu

Bogdan otrzymał od Alicji zaszyfrowaną wiadomość P i aby ją odszyfrować oblicza $M = P^d \pmod{n}$.
Przykład. Mamy obliczyć $M = 16^{103} \pmod{143}$. Postępujemy podobnie, jak w punkcie 2 przy szyfrowaniu wiadomości. Mamy $16^{103} = 16*16^2*16^4*16^{32}*16^{64}$ i z każdej z tych potęg obliczamy resztę z dzielenia przez 143. Otrzymujemy: $16*16^2*16^4*16^{32}*16^{64} = 16*113*42*113*42 = 81 \pmod{143}$.
Bogdan wie, że Alicja chciała jemu przekazać w tajemnicy wiadomość brzmiącą: Q.



Rysunek 6.
Schemat użycia szyfru RSA

Działanie szyfru RSA można prześledzić na przykładach, posługując się w tym celu programem edukacyjnym **Szyfr RSA** (patrz rys. 7). Program ten jest dostępny na stronie programu Informatyka+. Najpierw ustalamy w nim klucz publiczny i klucz prywatny, następnie szyfrujemy wybraną wiadomość, a na końcu odszyfrujemy ją. W głównym oknie programu można śledzić kolejne kroki i obliczenia służące do otrzymywania kluczy, szyfrowania i deszyfrowania wiadomości.



Rysunek 7.
Okno programu edukacyjnego **Szyfr RSA**

Szyfrowanie z kluczem jawnym ma ciekawe zastosowanie. Uczniowie biorący udział w Olimpiadzie Informatycznej w pierwszym etapie zawodów przesyłają rozwiązania na serwer organizatorów konkursu. Organizatorzy chcą zapewnić wszystkich uczniów, że ich rozwiązania dotrą bezpiecznie, to znaczy między innymi, że nie zostaną przechwycone i podpatrzone przez innych uczestników zawodów. Organizatorzy nie wiedzą jednak,

kto będzie startował. W tej sytuacji stosowany jest szyfr RSA – klucz publiczny jest zamieszczony na stronie olimpiady, by mógł z niego skorzystać każdy, kto chce wziąć udział w zawodach i wysłać do organizatorów zadania pierwszego etapu, natomiast klucz prywatny jest przechowywany przez organizatorów, by tylko jury olimpiady mogło zdeszyfrować rozwiązania. A zatem jest możliwe, by ten sam klucz publiczny mógł być stosowany przez wielu nadawców i każdy z nich może być pewien, że po zaszyfrowaniu nim swojej wiadomości będzie mogła ona być odszyfrowana tylko przez osoby, które dysponują prywatnym kluczem z pary – takimi osobami są członkowie jury olimpiady.

Szyfr RSA został wykorzystany w komputerowej realizacji szyfrowania z jawnym kluczem, zwanej **szyfrem PGP** (ang. *Pretty Good Privacy*), który jest powszechnie stosowany w Internecie.

9 PODPIS ELEKTRONICZNY

Jednym z produktów kryptografii jest **podpis elektroniczny** (zwany również **podpisem cyfrowym**), czyli podpis na dokumentach elektronicznych, a raczej – towarzyszący takim dokumentom. O jego znaczeniu może świadczyć fakt, że państwa Unii Europejskiej wprowadzają przepisy, uznające podpis elektroniczny za równorzędny z odręcznym. W Polsce również uchwalono w sejmie odpowiednią ustawę. A więc w przyszłości, zamiast iść do banku, by podpisać umowę o kredyt wystarczy taką umowę zaszyfrować, dołączyć do niej podpis elektroniczny i przesłać pocztą elektroniczną.

Podpis na dokumencie (tradycyjnym lub elektronicznym) jest odpowiedni, jeśli:
- zapewnia jednoznaczną identyfikację autora – nikt inny nie posługuje się takim samym podpisem;
- nie można go podrobić;
- nie można go skopiować na inny dokument – mechanicznie lub elektronicznie;
- gwarantuje, że po podpisaniu nim dokumentu nikt nie może wprowadzić żadnych zmian do tego dokumentu.

Opiszemy teraz, co to jest podpis elektroniczny i jak się nim posługiwać.

Sam podpis elektroniczny jest umownym ciągiem znaków, nierozerwalnie związanych z podpisywanym dokumentem. Jeśli chcemy posługiwać się takim podpisem, to najpierw należy skontaktować się z **centrum certyfikacji**. Tam otrzymuje się parę kluczy, publiczny i prywatny oraz osobisty **certyfikat elektroniczny**, który będzie zawierał m.in. osobiste dane i własny klucz publiczny. Osobisty certyfikat będzie dostępny w Internecie i może być wykorzystany przez odbiorcę dokumentu podpisanego elektronicznie do sprawdzenia tożsamości nadawcy.

Jeszcze jedno pojęcie jest ważne przy posługiwaniu się podpisem elektronicznym, to tzw. **skrót dokumentu**. Skrót powstaje przez zastosowanie do dokumentu odpowiedniego algorytmu (będącego realizacją tzw. **funkcji mieszającej**), np. SHA-1 lub MD-5. Skrót jest jednoznaczną reprezentacją dokumentu, tj. jakakolwiek zmiana w treści dokumentu powoduje zmianę w jego skrócie. Skrót zostaje również zaszyfrowany, jest na stałe związany z dokumentem, na podstawie którego powstał, i identyfikuje podpisującego.

Teraz możemy już opisać, na czym polega podpis elektroniczny dokumentu. Przypuśćmy, że chcę wysłać do Pana A dokument i podpisać go elektronicznie. Dokument znajduje się w komputerze, tam również jest dostępny mój certyfikat elektroniczny. Naciskam więc przycisk **Podpisz**. Wtedy są wykonywane następujące czynności (patrz rys. 8):
1. Tworzony jest skrót z dokumentu przeznaczonego do wysyłki.
2. Skrót ten zostaje zaszyfrowany moim kluczem prywatnym.
3. Do dokumentu zostaje dołączony jego zaszyfrowany skrót i mój certyfikat z kluczem publicznym.
4. Tak podpisany dokument jest przesyłany do odbiorcy, np. pocztą elektroniczną.

Rysunek 8.
Schemat składania podpisu elektronicznego pod dokumentem elektronicznym wysyłanym w sieci
[źródło: http://www.proinfo.com.pl/produkty/podpis-elektroniczny/]

Rysunek 9.
Weryfikacja podpisu elektronicznego pod dokumentem elektronicznym wysyłanym w sieci
[źródło: http://www.proinfo.com.pl/produkty/podpis-elektroniczny/]

Odbiorca, po otrzymaniu dokumentu podpisanego elektronicznie, aby zweryfikować jego autentyczność, naciska przycisk **Weryfikuj**. Wtedy (patrz rys. *9*):

1. Tworzony jest skrót z otrzymanego dokumentu.
2. Dołączony do dokumentu skrót zostaje rozszyfrowany moim kluczem publicznym, zawartym w moim certyfikacie elektronicznym.
3. Utworzony skrót z otrzymanego dokumentu i odszyfrowany skrót zostają porównane. Jeśli są zgodne, to oznacza, że od momentu podpisania dokument nie był modyfikowany oraz że ja, czyli osoba widniejąca na certyfikacie, jestem jego autorem.

Sam dokument może być również zaszyfrowany, np. kluczem publicznym odbiorcy.

Klucz prywatny nadawcy, najbardziej newralgiczna część całego procesu, może być umieszczony na karcie mikroprocesorowej, wtedy, by stosować podpis elektroniczny, należy wyposażyć się w specjalny czytnik. Klucz prywatny, raz umieszczony na karcie, pozostaje na niej cały czas. W takim przypadku, tworzenie skrótu dokumentu z udziałem tego klucza odbywa się również na tej karcie.

Przekonajmy się, że tak tworzony podpis ma wymienione na początku cechy:

- autentyczność nadawcy potwierdza jego certyfikat, do którego ma dostęp odbiorca;
- takiego podpisu nie można podrobić, bo klucz prywatny, pasujący do klucza publicznego znajdującego się w certyfikacie, ma tylko nadawca;
- podpisu nie można związać z innym dokumentem, gdyż nie będzie pasował do jego skrótu;
- skrót dokumentu jest również gwarancją, że dokument nie uległ zmianie po jego podpisaniu, gdyż inaczej skrót odszyfrowany przez odbiorcę nie zgadzałby się ze skrótem, utworzonym przez odbiorcę na podstawie odszyfrowanego listu.

Tworzy się **centra certyfikacji**, które wydają klucze wraz z niezbędnym oprogramowaniem. Co więcej, jeśli odbiorca naszych listów będzie się chciał dodatkowo upewnić, że koresponduje rzeczywiście z daną osobą, to będzie to mógł zrobić, kontaktując się z centrum, które wydało klucze nadawcy. Szczegółowy opis działania i tworzenia podpisu elektronicznego jest opisany na przykład na stronie pod adresem http://www.podpiselektroniczny.pl. Można tam również znaleźć informacje o centrach certyfikacji, jak również utworzyć swój osobisty certyfikat.

Na zakończenie wspomnijmy, że kryptografia ma także swoją ciemną stronę. Szyframi mogą bowiem posługiwać się osoby, które chcą ukryć przed innymi swoje nieczyste zamiary. W ten sposób państwo, które konstytucyjnie ma gwarantować swoim obywatelom bezpieczeństwo, może nie być w stanie wypełnić swojego zobowiązania, gdyby na przykład grupy przestępcze korzystały z szyfrów z kluczami, które zapewniają, że są to szyfry nie do złamania. Dlatego na przykład standardowy szyfr DES, wprowadzony w Stanach Zjednoczonych, ma ograniczoną długość klucza.

Ochrona i bezpieczeństwo przesyłanych wiadomości i przechowywanych danych ma fundamentalne znaczenie dla bezpieczeństwa człowieka i ochrony transakcji, wykonywanych elektronicznie. Z tego powodu kryptografia jest obecnie jednym z najaktywniej rozwijających się działów informatyki i wiele jej osiągnięć zapewne nawet nie jest ujawnianych. Duże nadzieje pokłada się w wykorzystaniu efektów fizyki kwantowej w tworzeniu nowych systemów kryptograficznych. Zainteresowanych szerszymi rozważaniami odsyłamy do wspomnianej już książki Simona Singha, *Księga szyfrów* [8], w której można przeczytać wiele fascynujących historii wiążących się z kryptografią.

10 ALGORYTMY WYKORZYSTYWANE W METODACH SZYFROWANIA

Omawiamy tutaj krótko algorytmy, a faktycznie wymieniamy je tylko, które są wykorzystywane w metodzie szyfrowania RSA. Chcemy pokazać, że algorytmy szyfrowania i deszyfrowania w tej metodzie są bardzo proste i sprowadzają się do wykonania działań, które są przedmiotem zajęć z informatyki w szkole. Niezbędne jest jednak poczynienie pewnych zastrzeżeń:

- uzasadnienie (dowody), że podane algorytmy szyfrowania i deszyfrowania działają poprawnie wykracza poza poziom matematyki szkolnej, nie jesteśmy więc w stanie wykazać, że wiadomość, jaką otrzymuje Bogdan jest dokładnie tą wiadomością, którą wysłała Alicja;
- działania w obu algorytmach, szyfrowania i deszyfrowania, są wykonywane na bardzo dużych liczbach, niezbędne jest więc posłużenie się w obliczeniach komputerowych specjalną arytmetyką długich liczb – nie piszemy o tym tutaj;
- bezpieczeństwo szyfru RSA jest oparte na aktualnym stanie wiedzy informatycznej, zarówno odnośnie problemów, które potrafimy rozwiązywać szybko, jak i problemów, których nie potrafimy rozwiązywać dysponując nawet najszybszymi komputerami.

Skomentujmy więc sposób realizacji poszczególnych kroków algorytmów szyfrowania i deszyfrowania. Komentowane fragmenty tych kroków są poniżej wyróżnione pismem pochyłym.

Przygotowanie kluczy

1. *Bogdan wybiera dwie duże, możliwie bliskie sobie liczby pierwsze p i q. Liczby te powinny mieć przynajmniej po 200 cyfr.*
   Znanych jest wiele metod generowania dużych liczb i sprawdzania, czy są one liczbami pierwszymi. Stosuje się przy tym często metody probabilistyczne, które z prawdopodobieństwem niemal 1 gwarantują, że są to liczby pierwsze.
2. *Bogdan oblicza n = p*q i wybiera dowolną liczbę naturalną e, która jest względnie pierwsza z liczbą (p – 1)*(q – 1), czyli te dwie liczby nie mają wspólnych dzielników poza liczbą 1.*
   Podobnie jak w punkcie 1.
3. *Bogdan znajduje liczbę naturalną d taką, że e*d = 1 (mod (p – 1)*(q – 1)), czyli reszta z dzielenia e*d przez (p – 1)*(q – 1) jest równa 1.*
   Liczbę *d* można znaleźć, posługując się rozszerzonym algorytmem Euklidesa. Algorytm Euklidesa, nawet dla dużych liczb działa bardzo szybko.
4. *Bogdan ogłasza parę liczb (n, e) jako swój klucz publiczny, a parę (n, d) zachowuje w tajemnicy jako klucz prywatny. Jednocześnie niszczy liczby p i q, by nie wpadły w niczyje ręce, co mogłoby umożliwić odtworzenie jej klucza prywatnego.*
   Jeśli *p* i *q* są dostatecznie dużymi liczbami pierwszymi, to znajomość *n* i *e* nie wystarcza, by obliczyć wartość *d* posługując się nawet najpotężniejszymi obecnie komputerami.

Szyfrowanie wiadomości

1. *Jeśli Alicja chce wysłać do Bogdana jakąś wiadomość, to najpierw musi ją przedstawić w postaci liczby naturalnej M, nie większej niż n. Tekst można zamienić na liczbę posługując się kodem ASCII.*
   Jeśli wiadomość jest zbyt długa, to należy ją szyfrować blokami odpowiedniej wielkości.
   To są proste rachunki na reprezentacjach liczb.
2. *Wiadomość, jaką Alicja wysyła do Bogdana, jest liczbą: $P = M^e \pmod n$.*
   Obliczenie wartości *P* można wykonać szybko przy wykorzystaniu szybkiej metody potęgowania, która w pierwszym kroku polega na przedstawieniu wykładnika w postaci binarnej. Wszystkie obliczenia są wykonywane na liczbach nie większych niż *n*. Te obliczenia ilustrujemy w naszym przykładzie przy opisie algorytmu.
3. *Alicja wysyła do Bogdana wiadomość P = 16.*

Odszyfrowanie kryptogramu

Bogdan odszyfrowuje otrzymaną wiadomość P wykonując obliczenia $M = P^d \pmod n$.
Obliczanie wartości *M* jest wykonywane takim samym algorytmem jak *P* w kroku 6.

Na zakończenie naszych rozważań przytoczmy jeden przykład efektywnych obliczeń na dużych liczbach, by uzmysłowić sobie, że obliczeniowa moc komputerów ma swoje ograniczenia i faktycznie cała nadzieja w szybkich algorytmach:

Przypuśćmy, że mamy obliczyć wartość potęgi[21]:

$$x^{12345678912345678912345678912345}$$

której wykładnik ma 32 cyfry. Jeśli zastosujemy szkolną metodę, która polega na kolejnym wykonywaniu mnożeń, to będziemy musieli wykonać ich o jedno mniej niż wynosi wykładnik. Przypuśćmy, że dysponujemy superkomputerem, który wykonuje 10^{15} mnożeń na sekundę. Wtedy ta potęga byłaby obliczona dopiero po $8*10^8$ latach.

A teraz zastosujmy algorytm „binarny", podobny do tego, którego użyliśmy w przykładowych obliczeniach. Wtedy wartość powyższej potęgi zostanie obliczona za pomocą około… 200 mnożeń i będzie to trwało niewielki ułamek sekundy.

LITERATURA

1. Ferguson N., Schneider B., *Kryptografia w praktyce*, Helion, Gliwice 2004
2. Grajek M., *Enigma. Bliżej prawdy*, Rebis, Poznań 2007
3. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO*, WSiP, Warszawa 2002-2003
4. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
5. Hodges A., *Enigma. Życie i śmierć Alana Turinga*, Prószyński i S-ka, Warszawa 2002
6. Knuth D.E., *Sztuka programowania*, tomy 1–3, WNT, Warszawa 2003
7. Kozaczuk W., *W kręgu Enigmy*, Książka i Wiedza, Warszawa 1986
8. Singh S., *Księga szyfrów. Od starożytnego Egiptu do kryptografii kwantowej*, Albatros, Warszawa 2001
9. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
10. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69
11. Wobst R., *Kryptologia. Budowa i łamanie zabezpieczeń*, Wydawnictwo RM, Warszawa 2002

[21] Więcej na ten temat – punkt 4.2 w rozdziale Czy wszystko można policzyć na komputerze.

Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew

Maciej M. Sysło
Uniwersytet Wrocławski, UMK w Toruniu
syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl
http://mmsyslo.pl/

Streszczenie

Wykład jest poświęcony elementom teorii grafów i obliczeń na grafach. Grafy odgrywają podwójną rolę w informatyce. Z jednej strony, są modelami obliczeń – w tej roli najczęściej występują drzewa – lub odzwierciedlają strukturę połączeń komunikacyjnych, a z drugiej – wiele problemów o praktycznych zastosowaniach jest definiowanych na grafach jako strukturach połączeń (zależności) między elementami. W pierwszej części wykładu zostaną przedstawione problemy, które miały wpływ na rozwój teorii grafów oraz dużą ich popularność. Druga część jest poświęcona wykorzystaniu drzew jako schematów obliczeń i algorytmów, a w trzeciej części zostaną przedstawione klasyczne problemy obliczeniowe na grafach, takie jak: znajdowanie najkrótszych dróg i znajdowanie najkrótszego drzewa rozpinającego w grafie z obciążonymi połączeniami. Rozwiązania problemów będą prezentowane w specjalnym oprogramowaniu.

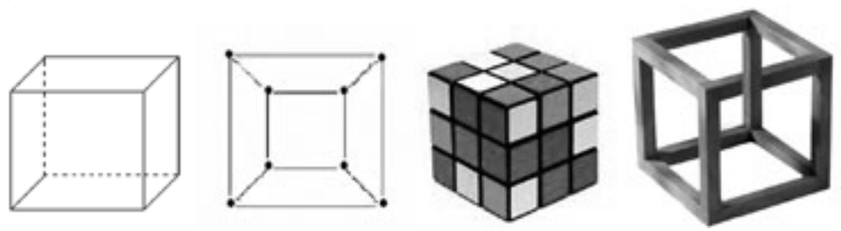
Spis treści

1 GRAFY JAKO MODELE RÓŻNYCH SYTUACJI

W potocznym sensie grafem jest pewien zbiór elementów, między którymi istnieją połączenia. Na przykład, miasta i sieć połączeń między nimi tworzą graf. Połączeniami z kolei mogą być drogi dla samochodów, sieć połączeń kolejowych lub sieć połączeń lotniczych. Na rysunku 1 przedstawiono fragmenty połączeń lotniczych. W rozważaniach informatycznych (algorytmicznych) na ogół nie interesuje nas, czemu odpowiadają elementy w takich sieciach połączeń (miasta, stacje, lotniska) i jaki jest charakter połączeń (drogowy, kolejowy, lotniczy). Elementy w takich sieciach połączeń nazywamy **wierzchołkami**, połączenia – **krawędziami** (jeśli są symetryczne, czyli nieskierowane) lub **łukami** (jeśli są skierowane), a sieć – **grafem** (gdy zawiera symetryczne połączenia) lub **digrafem** (gdy zawiera skierowane połączenia). Graf jest **modelem** sytuacji, w której mamy zbiór elementów i połączenia między nimi.

Rysunek 1.
Przykład sieci połączeń lotniczych z Bydgoszczy

Na rysunku 2 przedstawiono inny przykład pojawiania się grafów. Jest on związany z Eulerem (patrz następny rozdział), który rozważał siatki wielościanów, czyli grafy utworzone z wierzchołków i krawędzi wielościanów – stąd pochodzą nazwy wierzchołek i krawędź.



Rysunek 2.
Sześcian i graf jego siatki. Wierzchołki sześcianu są wierzchołkami grafu, a krawędzie sześcianu są krawędziami grafu. Dwa inne przykłady sześcianów są pokazane obok

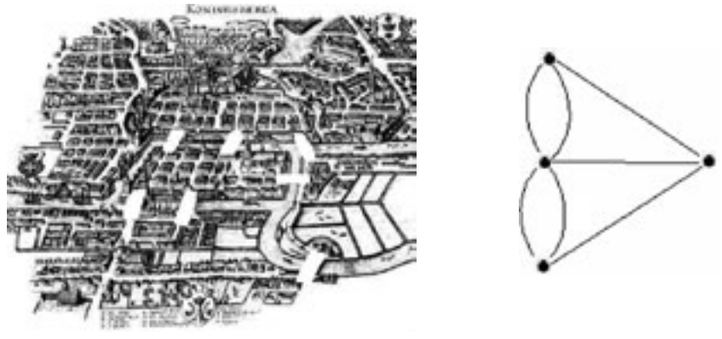
Elementarnym wprowadzeniem do teorii grafów jest książka [8]. Zastosowania grafów w informatyce zilustrowano w książkach [2], [5], [6]. Grafy związane z zagadkami można znaleźć w [6]. Głębsze rozważania na temat grafów i ich algorytmów zamieszczono w książkach [1], [3], [7].

2 TRZY KLASYCZNE ZAGADNIENIA

Przedstawiamy tutaj trzy przykłady klasycznych zagadnień, które uważa się, że najbardziej przyczyniły się do olbrzymiej popularności grafów jako modeli różnych sytuacji.

2.1 GRAFY EULERA

Wspomniany już wcześniej Leonhard Euler (1707-1783) jest uważany za ojca teorii grafów. W roku 1736, gdy mieszkał w obecnym Kaliningradzie (Rosja), zainteresował się, czy można zorganizować wycieczkę po tym mieście w taki sposób, aby przejść każdy most na rzece Pregoła, każdy dokładnie jeden raz. Zagadka ta nosi nazwę **mostów królewieckich**, gdyż Kaliningrad przez jakiś czas od połowy XV wieku nosił nazwę Królewiec; za czasów Eulera zaś nazywał się Königsberg. Na rysunku 3, na starej mapie tego miasta z czasów Eulera, naniesiono wspomniane w zagadce mosty, było ich siedem.



Rysunek 3.
Rzeka Pregoła w Königsbergu z zaznaczonymi na niej mostami (białe grube kreski), a obok graf (dokładniej – multigraf) odpowiadający tej sytuacji

Zagadkę Eulera można przetłumaczyć na język innej zagadki, w której pytamy, czy daną figurę można narysować na kartce papieru bez odrywania ołówka przechodząc każdą linię dokładnie jeden raz. Taka figura nazywa się **jednobieżną** lub **unikursalną**. Łatwo zauważyć, że w każdym wierzchołku takiej figury liczba połączeń, którymi wchodzimy do wierzchołka musi być równa liczbie połączeń, którymi wychodzimy z tego wierzchołka, a więc liczba połączeń w każdym wierzchołku – tę liczbę nazywamy **stopniem wierzchołka** – musi być parzysta, jeśli mamy powrócić do punktu startu, lub graf może zawierać dokładnie tylko dwa wierzchołki o stopniach nieparzystych, wtedy droga zawierająca wszystkie połączenia musi się zaczynać i kończyć w tych wierzchołkach. W pierwszym przypadku, graf zawiera **cykl Eulera**, a w drugim – **drogę Eulera**. Graf zawierający cykl Eulera nazywamy **grafem Eulera**. Łatwo spostrzec, że graf mostów królewieckich na rysunku 3 nie jest jednobieżny, gdyż wszystkie jego cztery wierzchołki mają stopnie nieparzyste.

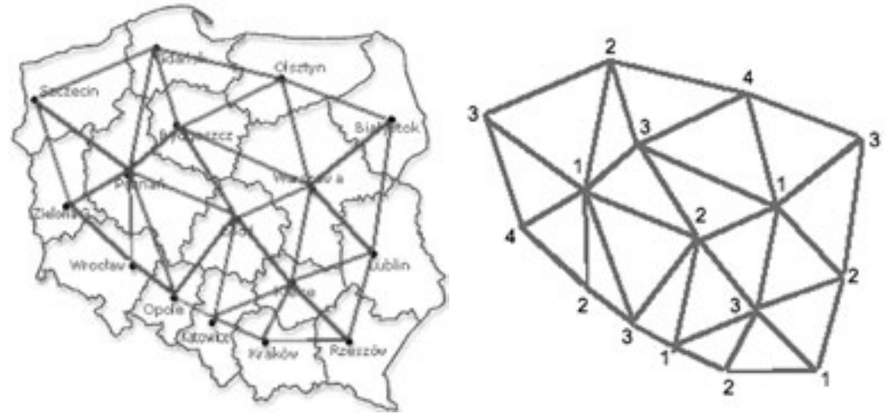
Rysunek 4.
Odpowiedz, która z figur jest jednobieżna?

2.2 MALOWANIE MAP

W połowie XIX wieku w Anglii duże zainteresowanie wzbudził **Problem Czerech Kolorów**, który przez ponad sto lat był **Przypuszczeniem Czterech Kolorów**. Ten problem, jak większość problemów dotyczących grafów, ma bardzo proste sformułowanie. Bierzemy mapę na przykład mapę Polski z podziałem administracyjnym i chcemy tak pomalować województwa kolorami, by żadne dwa sąsiadujące ze sobą województwa nie zostały pomalowane tym samym kolorem, staramy się przy tym użyć możliwie jak najmniejszej liczby kolorów. Francis Guthrie, student De Morgana, w 1852 roku postawił przypuszczenie, że zawsze cztery kolory wystarczą.

Związek Problemu Czterech Kolorów z grafami jest zilustrowany na rysunku 5. Najpierw tworzymy graf, który będzie odzwierciedlał sąsiedztwo województw. A zatem wierzchołki w tym grafie mogą odpowiadać stolicom województw i dwa wierzchołki połączymy krawędzią, jeśli odpowiadające im województwa bezpośrednio graniczą ze sobą. Zauważmy, że w tak utworzonym grafie, żadne dwie krawędzie nie przecinają się poza wierzchołkami – graf, który można tak narysować, nazywa się **grafem planarnym**, a jego rysunek – **grafem płaskim**. Specyfikacja Problemu Czterech Kolorów dla grafów ma teraz następującą postać:

Problem Czterech Kolorów dla grafów
Dane: Graf planarny *G*.
Wynik: Pomalowanie wierzchołków grafu *G* co najwyżej 4 kolorami w taki sposób, że żadne dwa wierzchołki połączone krawędzią nie zostały pomalowane tym samym kolorem.



Rysunek 5.
Mapa polski z podziałem na województwa. Na mapie naniesiono połączenia odpowiadające sąsiedztwu województw. Obok graf województw i jego pokolorowanie czterema kolorami (kolory są oznaczone liczbami 1, 2, 3, 4)

Wróćmy do malowania grafu województw (patrz rys. 5). Wierzchołki tego grafu pomalowaliśmy 4 kolorami (oznaczonymi liczbami 1, 2, 3, 4) stosując algorytm, w którym wierzchołki są malowane w kolejności stopni od największego, i malowanemu wierzchołkowi dajemy kolor o możliwie najmniejszej liczbie. W taki sposób, najpierw Poznań (sąsiaduje z 7 województwami) otrzymał kolor 1, później Łódź (sąsiaduje z 6 województwami) otrzymała kolor 2, następnie Warszawa otrzymała kolor 1, Kielce – kolor 3, Bydgoszcz – kolor 3, Gdańsk – kolor 2, Olsztyn – kolor 4, Lublin – kolor 2, Katowice – kolor 1, Opole – kolor 3, Szczecin – kolor 3, Białystok – kolor 3, Rzeszów – kolor 1, Kraków – kolor 2, Wrocław – kolor 2,

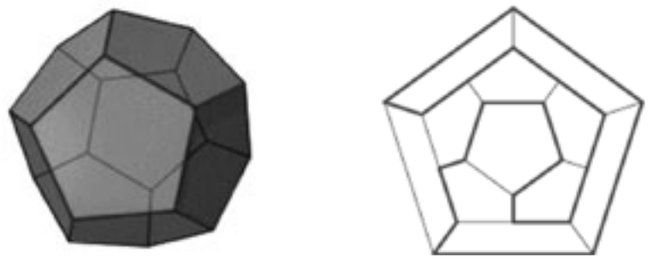
Zielona Góra – kolor 4. Jako ćwiczenie proponujemy sprawdzenie, czy tego grafu nie można pomalować trzema kolorami.

Zastosowany przez nas algorytm malowania grafu to **algorytm sekwencyjny**, który może być używany do malowania wierzchołków dowolnego grafu. Stosowana jest w nim **strategia zachłanna** – na każdym kroku jest używany możliwie najmniejszy kolor. Problem kolorowania grafów jest bardzo trudny algorytmicznie – więcej szczegółów na temat tego problemu można znaleźć w podanych w literaturze pozycjach [8] i [7].
Wróćmy jeszcze do historii Problemu Czterech Kolorów. W 1879 roku Alfred B. Kempe opublikował dowód, że każdy graf planarny można pomalować co najwyżej 4 kolorami. Jednak w 1890 roku Percy J. Heawood znalazł błąd w dowodzie Kempego, ale był w stanie udowodnić, że 5 kolorów wystarczy dla grafów planarnych (dowód jest bardzo prosty – patrz [8]). Przez niemal 100 lat tym problemem interesowali się niemal wszyscy matematycy, rozwijając wiele różnych metod, aż dopiero w 1976 roku Przypuszczenie Czterech Kolorów zostało potwierdzone przez... **komputer**. Dokonali tego Kenneth Appel i Wolfgang Haken przy współpracy z programistą Johnem Kochem. Komputer pracował przez 1200 godzin. Dotychczas nie udało się podać dowodu tego twierdzenia, który nie korzystałby z komputera.

2.3 GRAFY HAMILTONA

Bardzo podobnie do problemu Eulera z punktu 2.1 brzmi problem postawiony w 1859 roku przez Williama R. Hamiltona (1805-1865). W tym przypadku, zagadka dotyczyła dwunastościanu formalnego i należało znaleźć drogę zamkniętą przechodzącą po krawędziach tej bryły, która zawiera każdy jej wierzchołek dokładnie jeden raz (patrz rys. 6). Taką drogę nazywamy **cyklem Hamiltona**, a graf zawierający taki cykl – **grafem Hamiltona**.

Problem Cyklu Hamiltona
Dane: Dowolny *G*.
Wynik: Znaleźć cykl Hamiltona w grafie *G* albo stwierdzić, że takiego cyklu nie ma w grafie *G*.



Rysunek 6.
Dwunastościan foremny i jego graf z zaznaczonym cyklem Hamiltona

Różnica między problemami Eulera i Hamiltona tkwi w tym, że w pierwszym przypadku poszukujemy cyklu zawierającego wszystkie połączenia, każde dokładnie raz, a w drugim – cyklu zawierającego wszystkie wierzchołki, również każdy dokładnie raz. Różnica wydaje się być niewielka, jednak obliczeniowo te dwa problemy należą do diametralnie różnych klas złożoności. Problem Eulera ma łatwe rozwiązanie – wystarczy sprawdzić, czy stopnie wszystkich wierzchołków są parzyste i czy graf jest spójny (tzn. jest w „jednym kawałku"). Natomiast dla problemu Hamiltona nie podano dotychczas efektywnego algorytmu rozwiązywania. Jest to zapewne powodem, dlaczego praktyczna wersja problemu Hamiltona, znana pod nazwą problem komiwojażera, jest jednym z najtrudniejszych problemów kombinatoryjnych – patrz p. 5.2.1 w rozdziale Informatyka – klucz do zrozumienia, kariery, dobrobytu.

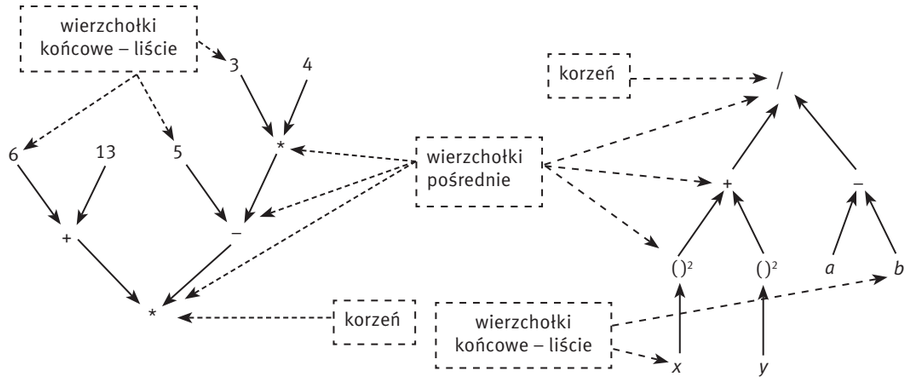
3 PRZYKŁADY ZASTOSOWAŃ GRAFÓW W INFORMATYCE

Grafy – szczególnie drzewa – mają zastosowania w informatyce. **Drzewo** jest grafem, który **nie zawiera cyklu**, czyli drogi zamkniętej, i jest **spójny**, czyli jest w „jednym kawałku". Ze spójności wynika, że każde dwa wierzchołki w drzewie są połączone drogą, a z braku cyklu – że dla każdych dwóch wierzchołków istnieje dokładnie jedna droga, która je łączy. Na ogół w zastosowaniach informatycznych, drzewa mają wyróżniony wierzchołek, który nazywamy **korzeniem**. Na rysunku 7 są przedstawione przykładowe drzewa.

Rysunek 7.
Przykładowe drzewa

3.1 DRZEWA WYRAŻEŃ

Na początku nauki matematyki w szkole podstawowej spotkaliście się z drzewem jako schematem obliczania wartości wyrażenia. W takim drzewie, zwanym **drzewem wyrażenia**, argumenty wyrażenia (liczby lub zmienne) znajdują się w **wierzchołkach końcowych** (zwanych **wiszącymi,** a także **liśćmi**), a działania są umieszczone w **wierzchołkach pośrednich**. Przykłady takich drzew są pokazane na rysunku 8. Wyróżniony wierzchołek drzewa nazywa się **korzeniem**. W tekstach informatycznych, drzewo jest na ogół rysowane z korzeniem u góry, jak po prawej stronie rysunku. Do elementów drzewa często stosuje się nazewnictwo wzięte z dendrologii – korzeń, liście.



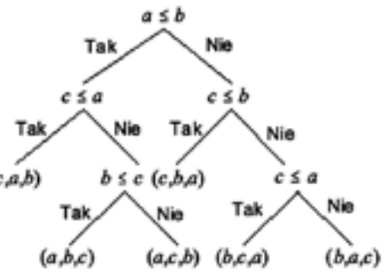
Rysunek 8.
Drzewa wyrażeń: (6 + 3)*(5 – 3*4) i $(x^2 + y^2)/(a - b)$

Wykonanie obliczeń zapisanych w postaci drzewa wyrażenia polega na „zwinięciu" tego drzewa, począwszy od liści, czyli przejściu od liści do korzenia, w którym jest otrzymywana wartość wyrażenia. Stosujemy przy tym oczywistą zasadę: aby móc wykonać dane działanie (umieszczone w wierzchołku pośrednim) musimy znać jego argumenty. A zatem, w wyrażeniu po lewej stronie na rysunku nie można wykonać odejmowania, zanim nie będzie znana wartość odjemnika w wierzchołku powyżej, czyli najpierw należy wykonać mnożenie umieszczone w tym wierzchołku. Proponujemy prześledzenie kolejności wykonania działań podczas obliczania wartości wyrażeń reprezentowanych przez drzewa na rysunku 8.

Drzewa wyrażenia są przydatne przy interpretacji i tworzeniu postaci wyrażenia w tak zwanej **odwrotnej notacji polskiej** (**ONP**), w której zbędne są nawiasy. Nazwa tej notacji pochodzi od twórcy notacji polskiej, polskiego logika Jana Łukasiewicza (1878-1956). Ta notacja jest stosowana w wielu językach programowania, była wykorzystana również w kalkulatorach Hewlett-Packard i Sinclair. Postać ONP wyrażenia tworzy się stosując rekurencyjnie następującą zasadę: Zacznij w korzeniu i wypisz to, co jest w wierzchołku dopiero po wypisaniu tego co jest w lewym poddrzewie, a później tego, co jest w prawym poddrzewie. Dla przykładu, zastosujmy tę zasadę do drzewa po lewej stronie na rysunku 8. Przed wypisaniem znaku mnożenia musimy najpierw wypisać to, co jest w lewym poddrzewie i w prawym poddrzewie. Przechodzimy więc do lewego poddrzewa i stosujemy w nim tę samą zasadę: aby wypisać znak dodawania najpierw wypisujemy oba argumenty tego działania, czyli dla lewego poddrzewa wyrażenia ma postać 6 13 +. Podobnie, dla prawego poddrzewa otrzymujemy 5 3 4 * – . A zatem całe wyrażenia ma następującą postać ONP: 6 13 + 5 3 4 * – *.

3.2 DRZEWA ALGORYTMÓW

Drzewo algorytmu jest jednym ze sposobów zapisywania algorytmów, przypominającym schematy blokowe. Na rysunku 9 przedstawiono drzewo algorytmu porządkowania trzech liczb *a*, *b*, *c*.



Rysunek 9.
Drzewo porządkowania trzech liczb

Wierzchołki końcowe drzewa algorytmu zawierają wszystkie możliwe rozwiązania – w naszym przykładzie są to wszystkie możliwe uporządkowania trzech elementów. Takich uporządkowań jest 6, a wykonywanie algorytmu zapisanego w postaci drzewa rozpoczyna się w korzeniu tego drzewa. Dzieje się zatem nieco inaczej niż w drzewie wyrażenia. Zapisanie tego algorytmu w postaci drzewa jest przejrzystym opisem wykonywanych operacji i ich kolejności. Ponadto drzewo algorytmu może służyć do konstruowania i analizowania algorytmów.

Drzewo na rysunku 9 może być łatwo rozszerzone do drzewa algorytmu, który służy do porządkowania czterech liczb *a*, *b*, *c* i *d*. Wystarczy wstawić czwarty element *d* do uporządkowanych ciągów trzech pierwszych elementów – w tym celu należy wykonać dwa dodatkowe porównania zaczynając od środkowego elementu.

Drzewo algorytmu może służyć do określenia liczby operacji wykonywanych przez algorytm. Zilustrujemy to na przykładzie drzewa przedstawionego na rysunku 9. Dla uzyskania konkretnego wyniku tego algorytmu, który znajduje się w wierzchołku wiszącym tego drzewa, liczba wykonanych porównań równa się liczbie wierzchołków pośrednich na drodze od korzenia (licząc również korzeń) do tego wierzchołka – tę liczbę nazywamy **długością** takiej drogi. Na przykład, jeśli dane trzy liczby *a*, *b* i *c* spełniają nierówności $c \le a \le b$ lub $c \le b \le a$, to algorytm wykonuje 2 porównania, a w pozostałych przypadkach – wykonuje 3 porównania. Największa długość drogi z korzenia do wierzchołka końcowego w drzewie nazywa się **wysokością drzewa**. Drzewo na rysunku 9 ma wysokość 3. Wysokość drzewa to największa liczba operacji wykonywanych w algorytmie dla

jakiegokolwiek układu danych. W algorytmice wielkość ta nazywa się **złożonością obliczeniową algorytmu**. Jak ilustruje to nasz przykład, jest to **złożoność pesymistyczna** lub **złożoność najgorszego przypadku**, gdyż w niektórych przypadkach algorytm może działać szybciej. Dla danego problemu mamy tym lepszy (szybszy) algorytm, im mniejszą wysokość ma jego drzewo.

Drzewo algorytmu może być wykorzystane do wykazania, ile operacji musi wykonać jakikolwiek algorytm rozwiązywania danego problemu, co może być wykorzystane przy dowodzeniu optymalności algorytmów (patrz [5]).

3.3 DRZEWA HUFFMANA – KRÓTKIE KODY

Pojemność pamięci komputerów rośnie w jeszcze większym tempie niż szybkość procesorów. Możliwość zapisania w pamięci komputera całej książki spowodowała chęć zapisania całych bibliotek. Możliwość pokazania na ekranie monitora dobrej jakości obrazu rozwinęła się do rozmiarów całego filmu. Alternatywą dla powiększenia pamięci jest **kompresja danych**, czyli minimalizowanie ich objętości przez reprezentowanie w zwięzłej postaci. Gwałtowny rozwój metod i form komunikowania się nie byłby możliwy bez ciągłego ulepszania metod kompresji danych. Odnosi się to zarówno do tradycyjnych form wymiany informacji, takich jak: faks, modem czy telefonia komórkowa, jak i do wymiany informacji za pośrednictwem sieci Internet, w tym zwłaszcza do wymiany informacji multimedialnych. Kompresja danych jest możliwa dzięki wykorzystaniu pewnych własności danych, na przykład często powtarzające się fragmenty można zastępować umownym symbolem lub im częściej jakiś fragment występuje tym mniejsza (krótsza) powinna być jego reprezentacja. W dalszej części tego punktu zajmiemy się jedynie kompresją tekstu, która polega na odpowiednim kodowaniu znaków.

Trudno uwierzyć, ale historia kompresji informacji ma swój początek… w połowie XIX wieku. 24 maja 1844 roku Samuel Morse po raz pierwszy posłużył się zbudowanym przez siebie telegrafem i przesłał na odległość 37 mil (z Kapitolu w Waszyngtonie do Baltimore) depeszę, w której zacytował Biblię „To, co uczynił Bóg” (ang. *What Hath God Wrought*!). Jego osiągnięciem było nie tylko zbudowanie telegrafu, ale również opracowanie specjalnego alfabetu, zwanego **alfabetem Morse’a**, umożliwiającego kodowanie znaków w tekście za pomocą dwóch symboli – kropki i kreski, którym wtedy odpowiadały krótsze lub dłuższe impulsy elektryczne, a dzisiaj mogłyby to być cyfry 0 i 1. Morse przy konstrukcji swojego alfabetu przyjął założenie, że im częściej znak występuje w informacji, tym krótszy powinien mieć kod. Dlatego w jego alfabecie litera E ma kod • (kropka), a litera T ma kod – (kreska), gdyż są to najczęściej występujące litery w tekstach języka angielskiego. W języku polskim byłyby to odpowiednio litery A oraz I. Wadą alfabetu Morse’a jest konieczność stosowania znaku oddzielającego litery, gdyż kod danej litery może być początkową częścią kodu innej litery. Na przykład zapis •• może oznaczać dwie litery EE, a także literę I, która ma taki kod.

Zauważmy, że stosując kod ASCII wszystkie znaki są kodowane za pomocą jednakowej liczby bitów.

Przedstawimy teraz krótko kod Huffmana, który jest zbudowany na podobnych zasadach co alfabet Morse’a, ale nie ma wspomnianej wady tego alfabetu, czyli nie muszą być stosowane znaki do rozdzielania kodów liter tekstu. W **kodzie Huffmana**:

- znaki są zapisywane również za pomocą dwóch znaków (cyfr 0 i 1);
- kod żadnego znaku nie jest początkiem kodu żadnego innego znaku – nie trzeba więc stosować znaków rozdzielających;
- średnia długość kodu znaku w tekście jest możliwie najmniejsza.

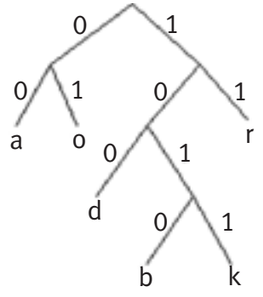
Kod Huffmana tworzy się za pomocą specjalnego drzewa, **drzewa Huffmana**, w którym powyższe własności są realizowane w następujący sposób:

- na gałęziach drzewa są umieszczone cyfry 0 i 1;
- znaki, dla których jest tworzony kod, znajdują się w wierzchołkach wiszących drzewa;
- wierzchołki wiszące ze znakami o większej częstości występowania w tekstach znajdują się wyżej niż wierzchołki ze znakami o mniejszych częstościach.

Drzewo Hoffmana dla danego zestawu znaków o podanych częstotliwościach jest budowane sukcesywnie przez łączenie ze sobą na każdym etapie dwóch poddrzew o najmniejszych częstotliwościach i zastępowaniu ich poddrzewem o sumie ich częstotliwości. Szczegółowy opis algorytmu Huffmana można znaleźć w podręczniku [2] i w książce [6]. Na rysunku 10 przedstawiamy drzewo Huffmana otrzymane tą metodą dla następujących znaków i ich częstotliwości (to są rzeczywiste częstotliwości występowania tych liter w tekstach w języku polskim):

a	8,71
b	1,29
d	3,45
k	3,10
o	7,90
r	4,63

W pierwszym kroku tworzenia tego drzewa, zgodnie z naszkicowanym algorytmem, łączone są dwie najmniejsze częstotliwości odpowiadające literom b oraz k i zastępowane są przez częstotliwość 1,29 + 3,10 = 4,39. Następnie jest łączona częstotliwość 3,35 (litera d) z otrzymaną przed chwilą częstotliwością itd.



Rysunek 10.
Drzewo Huffmana dla sześciu liter

Na podstawie drzewa z rysunku 10 otrzymujemy następujące kody liter:

a	00
b	1010
d	100
k	1011
o	01
r	11

Słowo ABRAKADABRA ma w otrzymanym kodzie postać 00101011001011001000010101100, złożoną z 29 cyfr, a stosując kod ASCII – postać tego słowa miałaby długość 88 znaków. Odpowiada to kompresji o wielkości 60%.

Algorytm Huffmana jest wykorzystywany w wielu profesjonalnych metodach kompresji tekstu, obrazów i dźwięków, również w połączeniu z innymi metodami. Redukcja wielkości danych przy stosowaniu tego algorytmu wynosi około 50% (w przypadku obrazów i dźwięków kodowane są nie same znaki, ale różnice między kolejnymi znakami).

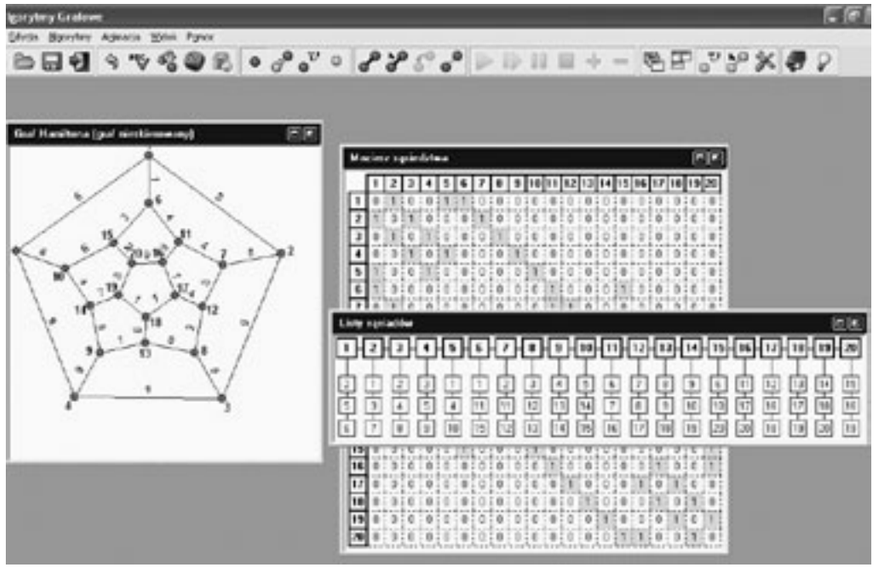
4 ALGORYTMY NA GRAFACH

W tej części wykładu przedstawiamy algorytmy na grafach, które mają wiele zastosowań praktycznych, są jednocześnie ilustracją typowych sposobów rozwiązywania problemów definiowanych na grafach. Szczegółowe rozważania dotyczące przedstawionych tutaj algorytmów można znaleźć w książce [7].

W czasie wykładu, działanie prezentowanych algorytmów będzie ilustrowane za pomocą specjalnego programu edukacyjnego **Algorytmy Grafowe**, w którym można:

- budować i modyfikować grafy;
- zapoznać się z podstawowymi sposobami reprezentowania grafów w komputerze;
- śledzić przebieg działania podstawowych algorytmów grafowych, w tym m.in. związanych z przeszukiwaniem grafów, znajdowaniem najkrótszych dróg w grafach i znajdowaniem najkrótszego drzewa rozpinającego grafu; podczas działania algorytmu, jego przebieg jest zsynchronizowany z demonstracją wykonywania programu w pseudojęzyku programowania, pokazywane są również zmieniające się stany struktur danych wykorzystywanych w algorytmach.

Na rysunku 11 jest przedstawione okno programu Algorytmy Grafowe[22], w którym widać graf i jego komputerowe reprezentacje.



Rysunek 11.
Okno w programie **Algorytmy Grafowe** z grafem dwunastościanu i jego komputerowymi reprezentacjami

[22] Program ten jest dostępny na stronie http://mmsyslo.pl/materialy/oprogramowanie.

4.4 GRAFY I ICH KOMPUTEROWE REPREZENTACJE

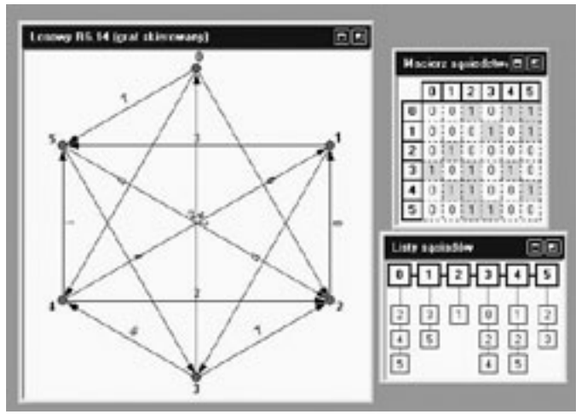
Jak już pisaliśmy, każdy graf składa się ze zbioru wierzchołków i zbioru połączeń. Wyróżniamy dwa rodzaje grafów: nieskierowane i skierowane.

Grafy nieskierowane nazywamy po prostu **grafami**. Wszystkie połączenia w nich są nieskierowane, a to oznacza, że mogą być przechodzone w obie strony. Połączenia w grafach nieskierowanych nazywamy **krawędziami**. Przykładowy graf jest pokazany w oknie programu na rysunku 11.

Z kolei grafy skierowane nazywamy **digrafami**. Wszystkie w nich połączenia mają zaznaczony kierunek przechodzenia. Połączenia w digrafach nazywamy **łukami**. Przykładowy digraf jest pokazany na rysunku 12.

Grafy i digrafy można reprezentować w komputerze na wiele sposobów. Na początku przyjmijmy, że wierzchołki są ponumerowane kolejnymi liczbami 1, 2, 3,... W każdym ze sposobów reprezentowania digrafów i grafów w komputerze zapisywano **sąsiedztwo wierzchołków**, czyli dla każdego wierzchołka jest podane, z którymi innymi wierzchołkami jest on połączony. W przypadku grafów symetrycznych, dla każdego wierzchołka podajemy numery wierzchołków, z którymi jest on połączony, a dla digrafów – podajemy numery wierzchołków, do których prowadzi skierowane połączenie.

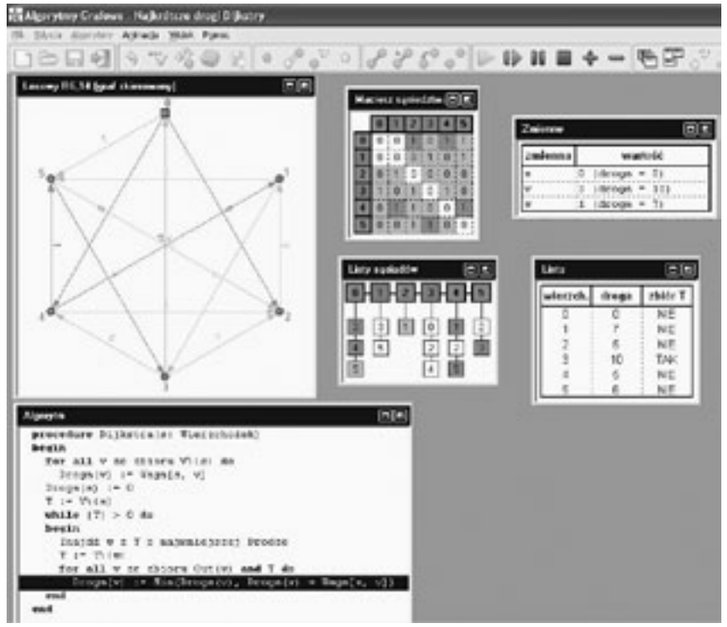
Najpopularniejsze są dwie reprezentacje: macierzowa i w postaci list sąsiadów. W reprezentacji macierzowej, jeśli graf zawiera połączenie z wierzchołka *j* do wierzchołka *k*, to w **macierzy sąsiedztwa** na przecięciu wiersza *j* z kolumną *k* jest 1, a w przeciwnym razie jest 0. Z kolei w **listach sąsiedztwa**, w liście o numerze *j* występują wszystkie wierzchołki, które są sąsiednie do wierzchołka *k*. Na rysunkach 11 i 12 ilustrujemy te komputerowe reprezentacje dla grafów przedstawionych na tych rysunkach. Sąsiedztwo w grafach należy traktować jako relację symetryczną, natomiast w digrafach sąsiedztwo jest wskazywane przez zwrot łuków.



Rysunek 12.
Digraf i jego komputerowe reprezentacje

4.5 ZNAJDOWANIE NAJKRÓTSZYCH DRÓG W GRAFACH

Drogą w grafie nazywamy ciąg wierzchołków, dla których graf zawiera połączenia między kolejnymi wierzchołkami tego ciągu. Zauważmy na rysunkach 11 i 12, że połączenia mają przypisane im liczby – są to **wagi połączeń**, które mogą oznaczać na przykład rzeczywistą długość połączenia. **Długość drogi** definiujemy jako sumę długości połączeń, które tworzą tę drogę. Na przykład, droga (2, 3, 8, 13, 9, 4) w grafie na rysunku 11 ma długość 9+5+8+1+9 = 32, a droga (1, 3, 4, 5, 2) w digrafie na rysunku 12 ma długość 5+5+1+4 = 15.



Rysunek 13.
Demonstracja działania algorytmu Dijkstry w programie **Algorytmy Grafowe**, zastosowana do digrafu z rysunku 12

Istnieje wiele różnych problemów, w których celem jest znalezienie najkrótszej drogi. Jeden z takich problemów był przedmiotem naszych rozważań w punkcie 2.3, gdzie zastanawialiśmy się, jak znaleźć trasę dla komiwojażera, czyli najkrótszy cykl przechodzący przez każdy wierzchołek w grafie dokładnie jeden raz. Klasyczne problemy najkrótszych dróg można podzielić na następujące grupy:

- znaleźć najkrótszą drogę między wybraną parą wierzchołków w grafie;
- znaleźć najkrótsze drogi z wybranego wierzchołka w grafie do wszystkich pozostałych wierzchołków w grafie;
- znaleźć najkrótsze drogi między każdą parą wierzchołków w grafie.

W tych sformułowaniach problemów „znaleźć najkrótszą drogę" oznacza znaleźć długość najkrótszej drogi oraz ciąg wierzchołków drogi o najkrótszej długości.

Łatwo zauważyć, że rozwiązanie problemu typu 1 może być wykorzystane w rozwiązaniu problemów typu 2 i 3. Podobnie, rozwiązanie problemu typu 2 może być wykorzystane do rozwiązania problemu typu 3. Na ogół, rozwiązywanie problemu typu 2 może być przerwane, gdy mamy już rozwiązanie problemu typu 1. Dlatego zatrzymamy się jedynie nad problemem typu 2 i dodatkowo założymy, że wszystkie wagi połączeń są nieujemne (w ogólności nie muszą być). A zatem, rozważymy następujący problem, przyjmując dodatkowo, że szukamy najkrótszych dróg w obciążonych digrafach. Jeśli chcemy znaleźć najkrótsze drogi w obciążonym grafie symetrycznym, to każdą krawędź traktujemy jako parę łuków przeciwnie skierowanych.

Problem najkrótszych dróg z ustalonego wierzchołka w digrafie obciążonym
Dane: Digraf *G* z łukami obciążonymi nieujemnymi wagami; wybrany wierzchołek *s*.
Wynik: Najkrótsze drogi w *G* z *s* do wszystkich pozostałych wierzchołków w digrafie *G*.

Do rozwiązania tego problemu posłużymy się algorytmem Dijkstry. Nie zamieszczamy tutaj jednak szczegółowego opisu tego algorytmu, tylko opiszemy jego główną ideę, a następnie w czasie wykładu zilustrujemy działanie tego algorytmu posługując się programem Algorytmy Grafowe, (patrz rysunek 13).

Algorytm Dijkstry ma charakter metody zachłannej. Startujemy z danego wierzchołka *s*, zaliczamy go do rozwiązania i w pierwszym kroku obliczamy długości dróg z *s* do wszystkich jego wierzchołów sąsiednich (te drogi składają się z pojedynczych łuków), a następnie wybieramy wierzchołek sąsiedni *w*, który jest najbliżej *s* – wierzchołek *w* zaliczamy do rozwiązania i dla każdego wierzchołka *v*, który nie należy jeszcze do rozwiązania, sprawdzamy, czy droga z *s* do *v* przez *w* nie jest krótsza od dotychczasowej drogi najkrótszej z *s* do *v*. Jeśli tak jest, to poprawiamy długości dróg. Następnie, ponownie wybieramy wierzchołek spośród tych, które nie należą jeszcze do rozwiązania, a który jest najbliższy *s* i powtarzamy to postępowanie. Liczba iteracji w tej metodzie wynosi *n* – 1, bo tyle wierzchołków trzeba dołączyć do rozwiązania w kolejnych krokach.

Polecamy prześledzić działania algorytmu Dijkstry w programie Algorytmy Grafowe na wybranych przykładach digrafów i grafów.

4.6 ZNAJDOWANIE NAJKRÓTSZYCH DRZEW ROZPINAJĄCYCH W GRAFACH

Jak pamiętamy, drzewo jest grafem, który jest spójny i nie zawiera cykli, a graf jest spójny, jeśli każda para jego wierzchołków jest połączona drogą. Zakładamy tutaj że graf jest symetryczny. Jeśli graf jest spójny, to można znaleźć w nim podgraf, który jest drzewem – takie drzewo w grafie nazywamy **drzewem rozpinającym**. Drzewo rozpinające w grafie spójnym można znajdować na dwa sposoby. Drzewo o *n* wierzchołkach ma dokładnie *n* – 1 krawędzi:

- usuwać krawędzie, które nie powodują rozpojenia grafu, tak długo jak długo graf ma więcej niż *n* – 1 krawędzi, gdzie *n* jest liczbą wierzchołków w grafie;
- wybierać krawędzie w grafie, ale tylko takie, które nie powodują powstania cyklu wśród wybranych krawędzi; postępować tak długo, aż wybranych zostanie *n* – 1 krawędzi, gdzie *n* jest liczbą wierzchołków w grafie.

Rozważany przez nas problem ma następującą postać:

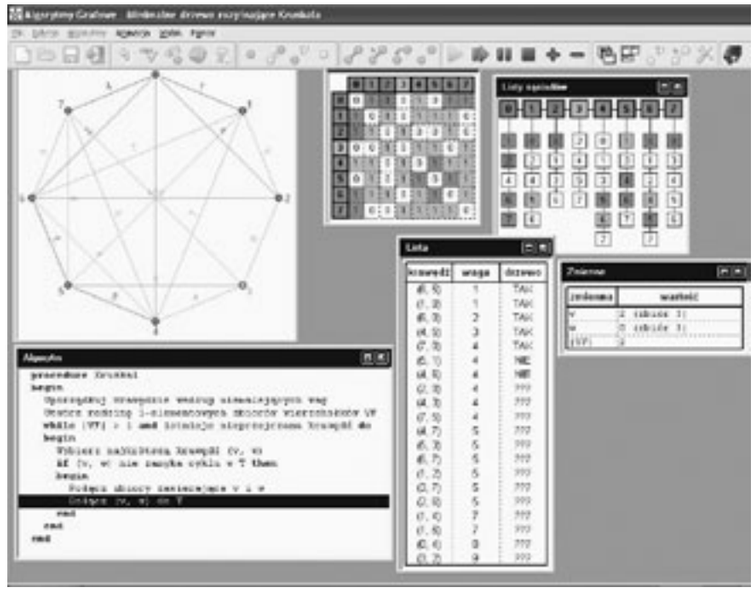
Problem najkrótszego drzewa rozpinającego w grafie obciążonym
Dane: Graf *G* z krawędziami obciążonymi wagami.
Wynik: Najkrótsze drzewo rozpinające w grafie *G*. Za długość drzewa przyjmujemy sumę wag (długości) jego krawędzi.

Najbardziej znanym algorytmem rozwiązywania tego problemu jest **zachłanny algorytm Kruskala**. Polega on na tworzeniu drzewa rozpinającego metodą nr 2 przy założeniu, że krawędzie są rozpatrywane w kolejności od najlżejszych (najkrótszych). Opiszemy ten algorytm w pseudo-języku programowania. Zakładamy, że rozważany graf jest spójny.

```
algorytm Kruskala;
begin
   T:=pusty;  {T – zbiór krawędzi tworzonego drzewa}
   E – zbiór krawędzi grafu G;
   while T ma mniej elementów niż n – 1 do begin
      wybierz najkrótszą krawędź e w zbiorze E;
      usuń e z E;
      if e nie tworzy cyklu z krawędziami w T then
         dołącz krawędź e do zbioru T
   end
end
```


Na rysunku 14 jest przedstawione okno z demonstracją algorytmu Kruskala na losowym grafie.

Ciekawą modyfikacją algorytmu Kruskala jest algorytm Prima-Dijkstry, który jest również algorytmem zachłannym, ale podobnie jak algorytm Dijkstry dla znajdowania najkrótszych dróg może rozpocząć budowanie najkrótszego drzewa rozpinającego zaczynając w dowolnym wierzchołku grafu, jako korzeniu. Proponujemy prześledzić działanie tego algorytmu w programie Algorytmy Grafowe.



Rysunek 14.
Demonstracja działania algorytmu Kruskala w programie **Algorytmy Grafowe**

LITERATURA

1. Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów*, WNT, Warszawa 1997
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO*, WSiP, Warszawa 2002-2003
3. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
4. Knuth D.E., *Sztuka programowania*, tomy 1–3, WNT, Warszawa 2003
5. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
6. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69
7. Sysło M.M., Deo N., Kowalik J.S., *Algorytmy optymalizacji dyskretnej z programami w języku Pascal*, WN PWN, Warszawa 1997
8. Wilson R.J., *Wprowadzenie do teorii grafów*, WN PWN, Warszawa 1998

O relacjach i algorytmach

Zenon Gniazdowski

Warszawska Wyższa Szkoła Informatyki

zgniazdowski@wwsi.edu.pl

Streszczenie

Podczas wykładu zostaną omówione relacje dwuczłonowe, a także sposoby ich reprezentacji w postaci macierzy lub grafu. Grafy zostaną użyte do pokazania relacji w postaci przemawiającej do wyobraźni. Tymczasem postać macierzowa umożliwi konstrukcję algorytmów do badania własności relacji. Zostaną także pokazane przykłady relacji wraz z przedstawieniem ich własności i określeniem ich typów.

Spis treści

1 WSTĘP

Relacja jest podstawowym pojęciem matematycznym [2, 3], także użytecznym w informatyce. Na przykład, w językach programowania występują operatory relacji $=, \neq, <, \leq, >, \geq$. Od tego, czy pewna relacja zachodzi, jest uzależnione wykonywanie instrukcji warunkowej.

Jako inny przykład można podać relacyjne bazy danych, gdzie dane są grupowane w relacje, które są reprezentowane, jako wiersze w tabelach. Relacja, o której tu mowa, jest uogólnieniem relacji dwuczłonowej, która zostanie przedstawiona w dalszej części wykładu.

Z kolei w eksploracji danych mówi się o danych otrzymywanych w wyniku pomiarów. Z pomiarem związana jest skala pomiarowa. Skale pomiarowe są definiowane przez różne typy relacji. Również w eksploracji danych wykorzystuje się teorię zbiorów przybliżonych (ang. *rough sets*), a w niej relację nierozróżnialności.

Relację dwuczłonową można przedstawić w postaci macierzy kwadratowej, która jednocześnie jest macierzą sąsiedztwa grafu zdefiniowanego przez tę relację. Macierzowa reprezentacja relacji umożliwia tworzenie algorytmów do badania własności relacji. Takie podejście ma istotne znaczenie z punktu widzenia dydaktyki informatyki. Informatyk powinien umieć skojarzyć pojęcia teoretyczne z praktycznym ich zastosowaniem. W szczególności, powinien umieć przedstawiać modele matematyczne w postaci zalgorytmizowanej.

W niniejszym wykładzie zostaną omówione relacje dwuczłonowe, ich własności i typy, a także algorytmy służące do ich badania. Ponieważ kluczem do zrozumienia niektórych algorytmów jest znajomość pewnych operacji na macierzach zostaną one krótko przedstawione w dodatku na końcu wykładu.

2 ILOCZYN KARTEZJAŃSKI ZBIORÓW

Rozważa się dwa zbiory X i Y. Zbiór wszystkich uporządkowanych par (x, y) elementów należących odpowiednio do tych zbiorów, nazywa się **iloczynem (produktem) kartezjańskim** zbiorów X i Y. Iloczyn kartezjański oznacza się jako $X \times Y$. Można to zapisać w następujący sposób:

$$X \times Y = \{(x, y): x \in X \text{ i } y \in Y\} . \tag{1}$$

Jeżeli dodatkowo zachodzi równość $X = Y$, to zamiast $X \times Y$ można napisać X^2.

Niech za przykład posłuży iloczyn kartezjański dwóch zbiorów X i Y, z których pierwszy zawiera cztery cyfry: $X = \{1, 2, 3, 4\}$, zaś drugi zawiera trzy litery: $Y = \{a, b, c\}$. Iloczynem kartezjańskim jest zbiór wszystkich par (*cyfra*, *litera*):

$$X \times Y = \{1, 2, 3, 4\} \times \{a, b, c\} = \begin{Bmatrix} (1, a)\ (1, b)\ (1, c) \\ (2, a)\ (2, b)\ (2, c) \\ (3, a)\ (3, b)\ (3, c) \\ (4, a)\ (4, b)\ (4, c) \end{Bmatrix} . \tag{2}$$

Przykład ten można także zinterpretować graficznie, jak na rysunku 1. Jeżeli w tabeli wiersze oznaczymy elementami jednego zbioru, a kolumny – elementami drugiego zbioru, to punkt na przecięciu odpowiedniego wiersza i kolumny reprezentuje parę (*cyfra*, *litera*).

	a	b	c
1	■	■	■
2	■	■	■
3	■	■	■
4	■	■	■

Rysunek 1.
Graficzna interpretacja iloczynu kartezjańskiego

Innym przykładem iloczynu kartezjańskiego jest zbiór punktów na płaszczyźnie *OXY*, oznaczany jako R^2. Pojęcie iloczynu kartezjańskiego można rozszerzyć na większą liczbę wymiarów. Dla przykładu, iloczyn $R \times R \times R = R^3$ oznacza zbiór punktów w przestrzeni trójwymiarowej.

Kolejnym przykładem iloczynu kartezjańskiego jest zbiór indeksów elementów macierzy. Macierz jest skończonym zbiorem elementów, zapisywanym w postaci prostokątnej tablicy o *m* wierszach i *n* kolumnach:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} . \qquad (3)$$

Adres elementu w macierzy składa się z dwóch liczb, z których pierwsza wskazuje na numer wiersza, a druga na numer kolumny, w których ten element się znajduje. Na przykład $a_{24} = 1$ oznacza, że element znajdujący się w drugim wierszu i czwartej kolumnie macierzy jest równy 1. Dokładniej rzecz ujmując, macierz można zdefiniować jako funkcję określoną na iloczynie kartezjańskim zbiorów $\{1, 2,\dots, m\}$ oraz $\{1, 2,\dots, n\}$:

$$f : \{1, 2,\dots, m\} \times \{1, 2,\dots, n\} \to R . \qquad (4)$$

3 RELACJA DWUCZŁONOWA

Dla danych zbiorów *X* i *Y* relacją dwuczłonową na iloczynie kartezjańskim $X \times Y$ jest dowolny podzbiór tego iloczynu. Przykładem relacji może być przydział przedmiotów, których uczą nauczyciele w pewnej szkole artystycznej. W przykładzie tym można wyodrębnić dwa zbiory. Pierwszy z nich, to zbiór nazwisk nauczycieli: $X = \{$*Kowalski*, *Kwiatkowski*, *Malinowski*, *Nowak*, *Wiśniewski*$\}$. W zbiorze drugim znajdą się przedmioty, które są w szkole nauczane: $Y = \{$*gimnastyka*, *rytmika*, *śpiew*, *taniec*$\}$. Oto przykładowy przydział nauczycieli do przedmiotów: {(*Nowak*, *rytmika*), (*Nowak*, *śpiew*), (*Nowak*, *taniec*), (*Kwiatkowski*, *gimnastyka*), (*Malinowski*, *rytmika*), (*Malinowski*, *taniec*), (*Kowalski*, *taniec*), (*Wiśniewski*, *gimnastyka*), (*Wiśniewski*, *śpiew*)}. Przydział ten jest przykładem relacji. Widać, że jest to tylko pewien podzbiór iloczynu kartezjańskiego $X \times Y$. Iloczyn kartezjański zawierałby dwadzieścia par (*nazwisko*, *przedmiot*), tymczasem przedstawiona relacja zawiera tylko dziewięć par.

3.1 REPREZENTACJA RELACJI

Rozważa się dwa zbiory: zbiór $X = \{x_1, x_2,\dots, x_m\}$ składający się z *m* elementów oraz zbiór $Y = \{y_1, y_2,\dots, y_n\}$ zawierający *n* elementów. Niech na ich iloczynie kartezjańskim $X \times Y$ będzie określona pewna relacja ρ. Relacja ta może być reprezentowana w postaci macierzy. Wiersze macierzy odpowiadają kolejnym elementom zbioru *X*, zaś kolumny – elementom zbioru *Y*. Macierzą relacji ρ jest zerojedynkowa macierz $[R_{ij}]$, zawierająca *m* wierszy i *n* kolumn. Jej elementy określone są następującym wzorem:

$$R_{ij} = \begin{Bmatrix} 1, & gdy & x_i \rho y_j \\ 0, & gdy & \neg(x_i \rho y_j) \end{Bmatrix} . \qquad (5)$$

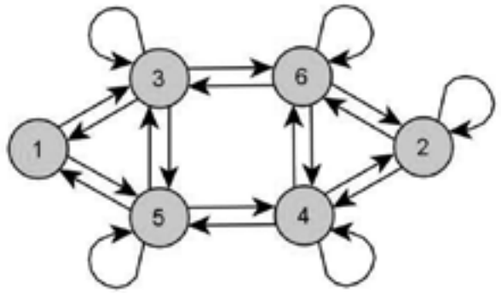
gdzie $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$. W powyższej formule wyrażenie $x_i \rho y_j$ czyta się: element x_i jest w relacji z elementem y_j, zaś wyrażenie $\neg(x_i \rho y_j)$ czyta się: nie jest prawdą, że element x_i jest w relacji z elementem y_j.

Jeżeli $X=Y$, to relacją w zbiorze *X* jest pewien podzbiór iloczynu kartezjańskiego $X \times X$. W dalszej części wykładu będą rozważane wyłącznie relacje w *n*-elementowym zbiorze *X*, reprezentowane przez kwadratową macierz o rozmiarze $n \times n$.

Przykład 1: Jako pierwszy przykład, zostanie pokazana relacja ρ określona na sześcioelementowym zbiorze $X = \{1, 2, 3, 4, 5, 6\}$. Dla dowolnych dwóch elementów $x, y \in X$, relacja jest zdefiniowana w następujący sposób: $x \rho y \Leftrightarrow x + y$ jest *liczbą złożoną*, a zatem element *x* jest w relacji z elementem *y* wtedy i tylko wtedy, gdy ich suma $x + y$ jest liczbą złożoną. Macierz tej relacji (ozn. *R*) ma następującą postać:

$$\mathrm{R} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} . \qquad (6)$$

Jeżeli elementy zbioru potraktować jako węzły grafu, to zdarzenie, iż element *i*-ty jest w relacji z elementem *j*-tym, można na grafie przedstawić przy pomocy łuku skierowanego od węzła *i* do węzła *j*. Otrzymany graf jest grafem skierowanym, zaś macierz relacji jest jednocześnie macierzą sąsiedztwa grafu. Na rysunku 2 jest przedstawiony graf relacji opisanej macierzą (6).



Rysunek 2.
Graf relacji opisanej macierzą (6)

4 WŁASNOŚCI RELACJI

Relacja dwuczłonowa może mieć następujące własności: może ona być zwrotna, przeciwzwrotna, symetryczna, antysymetryczna, przechodnia, spójna. Własności relacji ujawnią się w macierzy lub w grafie. Korzystając

z reprezentacji macierzowej, można algorytmizować sprawdzanie, czy relacja ma daną własność. Reprezentacja grafowa relacji wpływa na wyobraźnię i ułatwia proces tworzenia algorytmu.

Do tworzenia algorytmów zostanie użyty język C++, w którym całkowite wartości 1 albo 0 można traktować jako logiczne wartości *prawda* albo *fałsz*. Wobec tego, dla zerojedynkowych elementów macierzy relacji mogą być stosowane operacje arytmetyczne lub operacje logiczne. Kryterium wyboru będzie prostota i zwięzłość algorytmu.

W przedstawionych algorytmach pojawi się statyczna tablica `R[SIZE][SIZE]`. Stała `SIZE` oznacza maksymalną liczność zbioru, na którym jest zdefiniowana relacja. Kwadrat tej wielkości określa złożoność pamięciową przedstawianych algorytmów. Z drugiej strony, złożoność czasowa będzie funkcją liczby naturalnej n, będącej aktualną licznością zbioru, na którym jest definiowana relacja. Przy czym aktualny rozmiar zbioru jest nie większy niż rozmiar maksymalny: $n \leq$ `SIZE`.

W algorytmach zostanie w sposób milczący wykorzystana jeszcze jedna własność języka C++. W opisie matematycznym numery wierszy lub kolumn macierzy są indeksowane liczbami od 1 do n. Tymczasem w języku C++ n wierszy i n kolumn tablicy dwuwymiarowej są indeksowane liczbami od 0 do $n - 1$.

4.1 ZWROTNOŚĆ

Relacja ρ określona w zbiorze X jest zwrotna, gdy dla każdego elementu $x \in X$ element ten pozostaje w relacji z samym sobą. Symbolicznie można to przedstawić w następujący sposób:

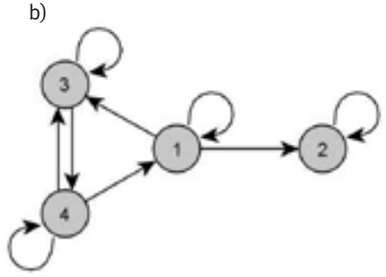
$$\forall x \in X : x \rho x. \tag{7}$$

W zapisie macierzowym zwrotność przejawia się tym, że wszystkie elementy znajdujące się na przekątnej macierzy R są równe 1. Oznacza to, że w każdym węźle grafu relacji znajduje się pętla. Na rysunku 3 przedstawiono przykład macierzy relacji zwrotnej oraz odpowiadający mu graf.

a)

$$R = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

b)



Rysunek 3.
Przykład relacji zwrotnej: a) macierz relacji, b) graf relacji

Algorytm badania zwrotności powinien sprawdzać, czy na przekątnej macierzy znajdują się same jedynki. Jeżeli pojawi się co najmniej jedno zero, to relacja nie jest relacją zwrotną. Algorytm badania zwrotności zapisany w postaci funkcji w języku C++ ma następującą postać:

```
int zwrotna(int R[SIZE][SIZE],int n)
{  int i;
   for (i=0; i<n; i++)
      if (R[i][i] == 0) return 0;
   return 1;
}
```


Dominującą operacją w algorytmie jest poszukiwanie zer na przekątnej macierzy relacji. Gdy relacja jest zwrotna, liczba przeszukiwań będzie równa aktualnej liczności zbioru, w którym jest definiowana relacja. Liczba ta wynosi n i jest górnym oszacowaniem złożoności obliczeniowej algorytmu.

4.2 PRZECIWZWROTNOŚĆ

Relacja ρ określona w zbiorze X jest przeciwzwrotna, gdy żaden element $x \in X$ nie jest w relacji z samym sobą. Symbolicznie przeciwzwrotność można wyrazić w następujący sposób:

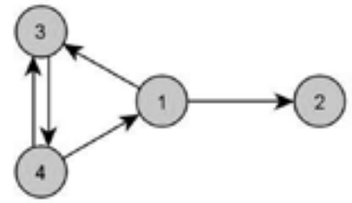
$$\forall x \in X : \neg(x \rho x). \tag{8}$$

W zapisie macierzowym przeciwzwrotność przejawia się tym, że na głównej przekątnej macierzy relacji znajdują się same zera. Na grafie ujawni się to w ten sposób, że żaden jego węzeł nie będzie miał pętli. Przykład relacji przeciwzwrotnej jest pokazany na rysunku 4.

a)

$$R = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

b)



Rysunek 4.
Przykład relacji przeciwzwrotnej: a) macierz relacji, b) graf relacji

Algorytm badania przeciwzwrotności powinien poszukiwać jedynek na przekątnej macierzy. Jeżeli pojawi się co najmniej jedna jedynka, to relacja nie jest relacją przeciwzwrotną. Funkcja badająca przeciwzwrotność ma następującą postać:

```
int przeciwzwrotna(int R[SIZE][SIZE],int n)
{  int i;
   for (i=0; i<n; i++)
      if (R[i][i]) return 0;
   return 1;
}
```

W algorytmie badania przeciwzwrotności dominującą operacją jest poszukiwanie jedynki na przekątnej macierzy relacji. Gdy relacja jest przeciwzwrotna, liczba przeszukiwań będzie równa aktualnej liczności zbioru, na którym jest definiowana relacja. Liczba ta wynosi n i jest górnym oszacowaniem złożoności obliczeniowej.

4.3 SYMETRIA

Relacja ρ określona w zbiorze X jest symetryczna, gdy dla każdych dwóch elementów x, $y \in X$ z faktu, że x jest w relacji z elementem y – wynika, że y jest w relacji z elementem x. Można to przedstawić tak:

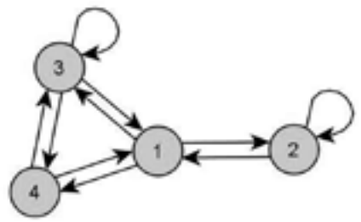
$$\forall x, y \in X : x \rho y \Rightarrow y \rho x. \tag{9}$$

W zapisie macierzowym symetria relacji przejawia się w symetrii macierzy relacji. Macierz R jest symetryczna, gdy jest równa swojej transpozycji: $R = R^T$. Obraz grafu takiej relacji charakteryzuje się tym, że wszystkie łuki między dwoma różnymi węzłami grafu biegną w dwóch kierunkach. Rysunek 5 przedstawia przykład relacji symetrycznej.

a)

$$R = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

b)



Rysunek 5.
Przykład relacji symetrycznej: a) macierz relacji, b) graf relacji

Algorytm badania symetrii relacji polega na przechodzeniu wszystkich elementów R_{ij} znajdujących się ponad główną przekątną macierzy relacji i sprawdzaniu, czy są im równe elementy R_{ji} leżące pod przekątną. Jeżeli pojawi się sytuacja, że $R_{ij} \neq R_{ji}$, to relacja nie jest symetryczna. Algorytm ten ma następującą postać:

```
int symetryczna(int R[SIZE][SIZE],int n)
{  int i,j;
   for (i=0; i<n–1; i++)
      for (j=i+1; j<n; j++)
         if (R[i][j] != R[j][i]) return 0;
   return 1;
}
```

W algorytmie badania symetrii operacją dominującą jest porównywanie elementu leżącego nad przekątną macierzy z odpowiadającym mu elementem leżącym pod przekątną. Nad przekątną macierzy znajdują się wiersze od pierwszego do przedostatniego. Przy czym w wierszu pierwszym jest $n - 1$ elementów, w wierszu kolejnym o jeden mniej, a w ostatnim wierszu nad przekątną jest już tylko jeden element. W skrajnym przypadku, gdy relacja będzie symetryczna, wszystkie elementy nadprzekątniowe biorą udział w porównywaniu. Liczba porównań jest równa liczbie tych elementów, a więc jest równa sumie ciągu arytmetycznego składającego się z kolejnych liczb od 1 do $n - 1$. Suma ta jest równa $n(n - 1)/2$ i jest górnym oszacowaniem złożoności obliczeniowej algorytmu badania symetrii relacji.

4.4 ANTYSYMETRIA

Relacja ρ określona w zbiorze X jest antysymetryczna, jeżeli dla każdych dwóch elementów $x, y \in X$ z faktu, że x jest w relacji z elementem y i y jest w relacji z elementem x wynika, że elementy x i y są identyczne. Symbolicznie zapisujemy to w następujący sposób:

$$\forall x, y \in X : x \rho y \wedge y \rho x \Rightarrow x = y. \tag{10}$$

W macierzy relacji antysymetrycznej każdemu elementowi $R_{ij} = 1$ spoza przekątnej towarzyszy element $R_{ji} = 0$. Jest to równoważne następującemu warunkowi:

$$\forall (i, j)_{i \neq j} \, R_{ij} \wedge R_{ji} = 0. \tag{11}$$

Na grafie ujawni się to w ten sposób, że jeżeli między dwoma różnymi węzłami istnieje łuk, to między tymi węzłami nie ma łuku biegnącego w przeciwnym kierunku. Na rysunku 6 pokazano przykład relacji antysymetrycznej.

Algorytm badania antysymetrii polega na przechodzeniu przez wszystkie elementy macierzy znajdujące się ponad przekątną i sprawdzaniu warunku (11). Jeżeli ten warunek nie jest spełniony co najmniej jeden raz, to relacja nie jest antysymetryczna. Algorytm badania antysymetrii zapisany w języku C++ ma postać:

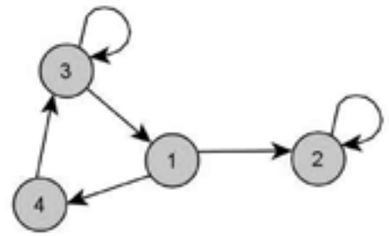
```
int antysymetryczna(int R[SIZE][SIZE],int n)
{  int i,j;
   for (i=0; i<n–1; i++)
      for (j=i+1; j<n; j++)
         if (R[i][j] && R[j][i]) return 0;
   return 1;
}
```

W algorytmie badania antysymetrii dominującą operacją jest sprawdzanie, czy jest równy zero iloczyn boolowski elementów R_{ij} leżących nad przekątną macierzy i odpowiadających im elementów R_{ji} leżących pod przekątną. Gdy relacja jest antysymetryczna, to podobnie jak w przypadku algorytmu badania symetrii, liczba sprawdzeń wynosi $n(n - 1)/2$. Liczba ta jest górnym oszacowaniem złożoności obliczeniowej algorytmu.

a)

$$R = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

b)



Rysunek 6.
Przykład relacji antysymetrycznej: a) macierz relacji, b) graf relacji

4.5 PRZECIWSYMETRIA

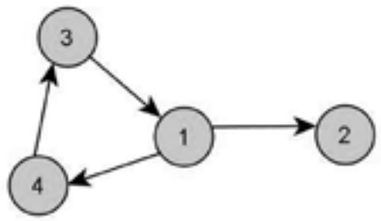
Relacja ρ określona w zbiorze X jest przeciwsymetryczna, gdy dla każdych dwóch elementów $x, y \in X$ z faktu, że x jest w relacji z y wynika, że y nie jest w relacji z x. Można to przedstawić w następujący sposób:

$$\forall x, y \in X : x \rho \, y \Rightarrow \neg (y \rho \, x). \tag{12}$$

a)

$$R = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

b)



Rysunek 7.
Przykład relacji przeciwsymetrycznej: a) macierz relacji, b) graf relacji

Przeciwsymetria relacji jest równoważna występowaniu dwóch innych wcześniej zdefiniowanych własności relacji: antysymetrii i przeciwzwrotności. Jeżeli relacja jest przeciwzwrotna i antysymetryczna, to jest przeciwsymetryczna. Oznacza to, że dla badania przeciwsymetrii nie ma potrzeby tworzenia specjalnego algorytmu. Relacja przeciwsymetryczna bywa także nazywana relacją asymetryczną [2]. Na rysunku 7 jest przedstawiony przykład relacji przeciwsymetrycznej.

4.6 PRZECHODNIOŚĆ

Relacja ρ określona w zbiorze X jest przechodnia, jeżeli dla dowolnych elementów $x, y, z \in X$ z faktu, że x jest w relacji z elementem y i y jest w relacji z elementem z wynika, że x jest w relacji z elementem z. Korzystając z symboliki matematycznej, przechodniość opisuje się następująco:

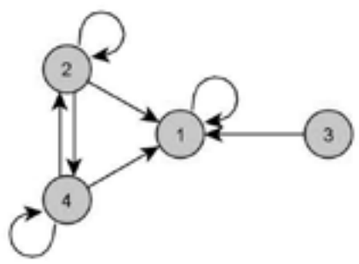
$$\forall x, y, z \in X : x \rho y \wedge y \rho z \Rightarrow x \rho z. \tag{13}$$

Graf relacji przechodniej charakteryzuje się tym, że jeżeli istnieje łuk od węzła x do węzła y i od węzła y do węzła z, to istnieje także łuk idący „na skróty” od węzła x do węzła z. Rysunek 8 przedstawia przykład relacji przechodniej.

a)

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

b)



Rysunek 8.
Przykład relacji przechodniej: a) macierz relacji, b) graf relacji

Warunek przechodniości można także sformułować inaczej: *jeśli pomiędzy dwoma różnymi węzłami grafu relacji istnieje ścieżka o długości dwóch łuków, to w grafie relacji przechodniej będzie między nimi istniała ścieżka o długości jednego łuku*. Powyższa obserwacja może pomóc w znalezieniu algorytmu badania przechodniości. Element R_{ij} macierzy sąsiedztwa grafu jest równy liczbie ścieżek o długości jednego łuku, biegnących od węzła i do węzła j. W macierzy sąsiedztwa grafu relacji będzie to co najwyżej jedna ścieżka. Kwadrat macierzy sąsiedztwa zlicza wszystkie ścieżki o długości dwóch łuków [3]. Warunek przechodniości relacji można teraz wyrazić następująco: *jeżeli* $(R^2)_{ij} > 0$, to $R_{ij} = 1$.

Macierz relacji jest macierzą zerojedynkową. Jej kwadrat – zliczający ścieżki o długości dwóch łuków – może zawierać zera lub liczby większe od zera. Ponieważ dla badania przechodniości nie jest istotne, ile jest ścieżek o długości dwóch łuków, ale czy takie ścieżki istnieją, to zamiast niezerowej liczby ścieżek o długości dwóch łuków w odpowiednich miejscach wynikowej macierzy R^2 wystarczy wstawić jedynkę informującą, że takie ścieżki istnieją, a następnie sprawdzać, czy jedynkom w wynikowej macierzy R^2 towarzyszą jedynki w macierzy R. Oznacza to, że operację podnoszenia macierzy R do kwadratu, można zastąpić operacją boolowskiego mnożenia macierzy. Wtedy warunek przechodniości relacji będzie spełniony, gdy:

$$R * R \leq R. \tag{14}$$

W wyrażeniu (14) operacja (*) oznacza mnożenie boolowskie macierzy [3]. W zwykłym mnożeniu macierzy znanym z algebry liniowej, wynikowy element mnożenia znajdujący się w i-tym wierszu i j-tej kolumnie jest liczony jako iloczyn skalarny i-tego wiersza i j-tej kolumny w macierzy R, co można zapisać w następującej postaci:

$$(R^2)_{ij} = \sum_{k=1}^{n} R_{ik} R_{kj} \ . \tag{15}$$

W mnożeniu boolowskim zamiast iloczynu i sumy w wyrażeniu (15) używa się odpowiednio iloczynu i sumy logicznej. Oznaczając wynik boolowskiego mnożenia macierzy jako B, pojedynczy element tej macierzy można wyrazić w następujący sposób: $B_{ij} = R_{i1} \wedge R_{1j} \vee R_{i2} \wedge R_{2j} \vee \ldots \vee R_{in} \wedge R_{nj}$. W zapisie zwartym wyrażenie ma postać:

$$B_{ij} = \bigcup_{k=1}^{n} R_{ik} \wedge R_{kj} \ . \tag{16}$$

Nierówność w wyrażeniu (14) oznacza, że w zerojedynkowej macierzy B jedynki mogą się znajdować co najwyżej w tych miejscach, w których znajdują się jedynki w macierzy R. Algorytm badania przechodniości składa się z dwóch faz. W fazie pierwszej wykonywana jest operacja (16), zaś w fazie drugiej sprawdzany jest warunek (14):

```
int przechodnia(int R[SIZE][SIZE], int n)
{  int B[SIZE][SIZE];
   int i,j,k;
//mnożenie boolowskie: B = R*R – wz. (16)
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
      {  B[i][j] = 0;
         for (k=0; k<n; k++)
            B[i][j] = B[i][j] || (R[i][k]&&R[k][j]);
      }
//sprawdzenie warunku (14)
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
         if (R[i][j] <B[i][j]) return 0;
   return 1;
}
```

Przedstawiony algorytm można dalej uprościć:

- Ponieważ znajdowanie kolejnych elementów macierzy *B* odbywa się niezależnie od innych elementów tej macierzy, nie ma potrzeby wyliczać całej macierzy przed przystąpieniem do sprawdzania warunku (14). Zamiast tego można wyliczać kolejny element macierzy *B* i sprawdzić, czy jest równy 1. Jeżeli tak jest, to także trzeba sprawdzić, czy odpowiadający mu element macierzy *R* jest równy 1. Jeżeli równość nie zachodzi, to można zakończyć działanie algorytmu, gdyż relacja nie jest relacją przechodnią.
- W wyrażeniu (16) można zrezygnować z sumowania. Jeżeli kolejny iloczyn wynosi zero, sumowanie nic nie zmieni. Jeżeli iloczyn jest równy jeden, należy przerwać pętlę liczącą iloczyny.
- Ponieważ cała macierz *B* nigdy nie jest potrzebna, a tylko jej kolejne elementy są lokalnie wyliczane, to można także zrezygnować z macierzy *B* na rzecz zmiennej lokalnej *B*, która przechowuje boolowską sumę iloczynów (16).

Po uproszczeniach, ostateczna wersja algorytmu ma postać:

```
int przechodnia(int n, int R[SIZE][SIZE])
{  int i,j,k;
   int B;
   for (i=0; i<n; i++) for (j=0; j<n; j++)
      {  B = 0;
         for (k=0; (!B) && (k<n); k++) B = R[i][k] && R[k][j];
         if (R[i][j] < B) return 0;
      }
   return 1;
}
```

W algorytmie badania przechodniości dominującą operacją jest obliczanie wartości co najwyżej n^2 wartości *B*. Dodatkowo, dla znalezienia pojedynczego elementu *B* potrzeba obliczyć maksymalnie *n* iloczynów. Stąd szacowana górna złożoność algorytmu wynosi n^3 iloczynów boolowskich.

4.7 SPÓJNOŚĆ

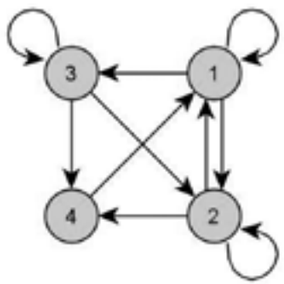
Relacja ρ określona w zbiorze *X* jest spójna, jeżeli dla dowolnych dwóch elementów $x, y \in X$ element *x* pozostaje w relacji z elementem *y* lub element *y* pozostaje w relacji z elementem *x*. Spójność wyrażona symbolicznie ma postać:

$$\forall x, y \in X: x \rho y \vee y \rho x \vee x = y. \tag{17}$$

a)

$$R = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

b)



Rysunek 9.
Przykład relacji spójnej: a) macierz relacji, b) graf relacji

W zapisie macierzowym spójność przejawia się tym, że jeżeli poza przekątną w macierzy relacji zachodzi $R_{ij} = 0$, to odpowiednio $R_{ji} = 1$. Oznacza to, że dla relacji spójnej zawsze zachodzi następujący warunek:

$$\forall (i, j)_{i \neq j} R_{ij} \vee R_{ji} = 1. \tag{18}$$

Graf relacji spójnej charakteryzuje się tym, że pomiędzy dwoma różnymi jego węzłami istnieje łuk co najmniej w jednym kierunku. Na rysunek 9 pokazano przykład relacji spójnej.

Algorytm badania spójności relacji polega na przebiegu przez wszystkie elementy macierzy znajdujące się ponad przekątną i sprawdzaniu, czy spełniony jest warunek (18):

```
int spojna(int R[SIZE][SIZE],int n)
{  int i,j;
   for (i=0; i<n–1; i++)
      for (j=i+1; j<n; j++)
         if (!(R[i][j] || R[j][i])) return 0;
   return 1;
}
```

W algorytmie badania spójności operacją dominującą jest sprawdzanie, czy logiczna suma elementów leżących nad przekątną macierzy (R_{ij}) i odpowiadających im elementów spod przekątnej (odpowiednio R_{ji}) jest równa 1. Gdy relacja jest spójna, liczba sprawdzeń – będąca jednocześnie górnym oszacowaniem złożoności algorytmu – wynosi (podobnie jak w przypadku algorytmów badania symetrii i antysymetrii) co najwyżej $n(n-1)/2$.

5 TYPY RELACJI

Własności relacji dwuczłonowej mogą konstytuować jej typ:

- Jeżeli relacja jest zwrotna, symetryczna i przechodnia, to jest relacją równoważności. Relacja równoważności dzieli zbiór na rozłączne klasy abstrakcji (klasy równoważności). Jako przykład można przedstawić następującą relację równoważności: dwa samochody na pobliskim parkingu są ze sobą w relacji, gdy mają ten sam kolor. Można sprawdzić, że tak zdefiniowana relacja jest zwrotna, symetryczna i przechodnia. Klas abstrakcji jest tyle, na ile różnych kolorów są pomalowane samochody: jedną z klas abstrakcji stanowią samochody czarne, inną czerwone, jeszcze inną srebrne itd.
- Kolejnym typem jest relacja częściowego porządku. Jest to relacja zwrotna, antysymetryczna i przechodnia. Zbiór z relacją częściowego porządku jest zbiorem częściowo uporządkowanym. W zbiorze częściowo uporządkowanym porządkowanie (sortowanie w sensie danej relacji) jest możliwe tylko w ramach pewnych podzbiorów. Przykładem relacji częściowego porządku jest relacja podzielności w zbiorze liczb całkowitych dodatnich nie większych niż 100.

  Formą przedstawienia relacji jest jej graf. Tymczasem skończony zbiór uporządkowany można także przedstawić w postaci tzw. diagramu Hassego [1, 2, 3]. Dla danej relacji porządkującej diagram ten powstaje przez zredukowanie grafu relacji. Najpierw należy zredukować wszystkie pętle charakteryzujące zwrotność. W drugiej kolejności należy odrzucić wszystkie łuki charakteryzujące przechodniość. Otrzymany po redukcji graf należy narysować tak, aby wszystkie jego strzałki były skierowane do góry. Po tej operacji należy zlikwidować wszystkie groty strzałek na końcach łuków [1, 3].

- Z kolei mocniejsze wymagania spełnia relacja porządku liniowego. Jeżeli relacja jest relacją porządku częściowego (a więc jest relacją zwrotną, antysymetryczną i przechodnią) i jednocześnie jest spójna, to jest

to relacja porządku liniowego. Porządek liniowy jest własnością silniejszą niż porządek częściowy. Umożliwia on porządkowanie (sortowanie w sensie danej relacji) całego zbioru. Przykładem relacji porządku liniowego może być relacja ≥ w zbiorze liczb całkowitych dodatnich nie większych niż 100.

Podobnie jak w przypadku relacji porządku częściowego, relację porządku liniowego można również przedstawić w postaci diagramu Hassego, który tym razem ma postać linii.

Relacja równoważności umożliwia badanie, czy dwa elementy w zbiorze są równe (należą do tej samej klasy abstrakcji), czy też są różne (należą do różnych klas abstrakcji). W ramach tej relacji nie ma możliwości porządkowania elementów np. w sensie sprawdzania, czy jeden element poprzedza drugi element (w sensie rozważanej relacji). Tymczasem relacje porządku dają takie możliwości.

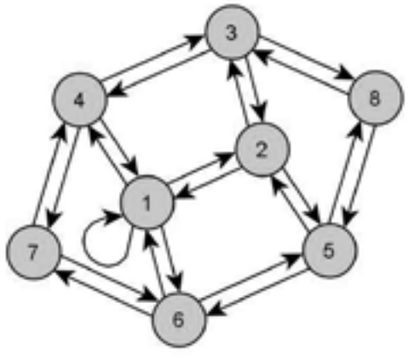
6 PRZYKŁADY RELACJI

Dla pokazania różnych własności i typów relacji zostaną przedstawione dalsze przykłady kilku relacji. W każdym z przykładów zostaną wskazane własności relacji oraz (o ile istnieje) jej typ.

Przykład 2: Dany jest zbiór $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Relacja w zbiorze X jest zdefiniowana w następujący sposób: $x \rho y \Leftrightarrow x+y$ jest *liczbą pierwszą*, $x, y \in X$. Macierz tej relacji ma postać:

$$R = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \quad (19)$$

Na rysunku 10 przedstawiono graf relacji (19). Relacja ta jest relacją symetryczną: dla macierzy zachodzi związek $R = R^T$, zaś w grafie pomiędzy węzłami biegną łuki w obie strony. Symetria jest jedyną własnością omawianej relacji, ponieważ ta relacja nie jest zwrotna (7), przeciwzwrotna (8), antysymetryczna (10), przechodnia (13) i spójna (17).



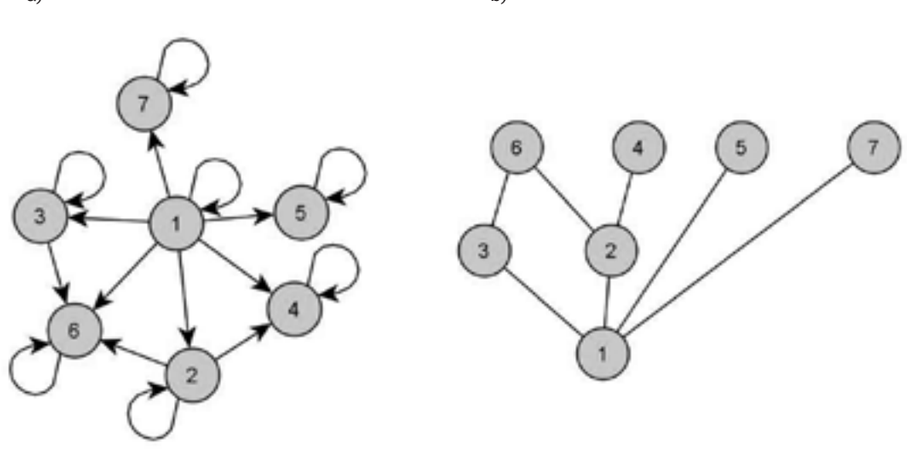
Rysunek 10.
Graf relacji opisanej macierzą (19)

Przykład 3: Dla danego zbioru $X = \{1, 2, 3, 4, 5, 6, 7\}$, relacja jest zdefiniowana w następujący sposób: $x \rho y \Leftrightarrow x$ *jest dzielnikiem liczby* y, $x, y \in X$. Macierz relacji ma postać:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (20)$$

Rysunek 11a przedstawia graf relacji (20). Pokazana w przykładzie relacja jest zwrotna – każda z liczb jest swoim własnym dzielnikiem. Jest to także relacja antysymetryczna – jeżeli liczba x jest dzielnikiem różnej od siebie liczby y, to liczba y nie jest dzielnikiem liczby x. W grafie będzie to widoczne w ten sposób, że pomiędzy węzłami biegną łuki tylko w jedną stronę. Relacja jest także przechodnia – zachodzi zależność (14). W ten sposób spełnione są warunki definiujące relację częściowego porządku.

Na rysunku 11b przedstawiono diagram Hassego dla relacji (20), otrzymany z redukcji grafu pokazanego na rysunku 11a. Odrzucenie łuków przedstawiających zwrotność nie niesie żadnych trudności. W przypadku redukcji linii charakteryzujących przechodniość usunięto łuki będące „drogami na skróty" od węzła 1 do węzła 4, a także od węzła 1 do węzła 6. Po usunięciu grotów strzałek pozostał diagram Hassego.



Rysunek 11.
Reprezentacja relacji (20): a) graf relacji, b) diagram Hassego

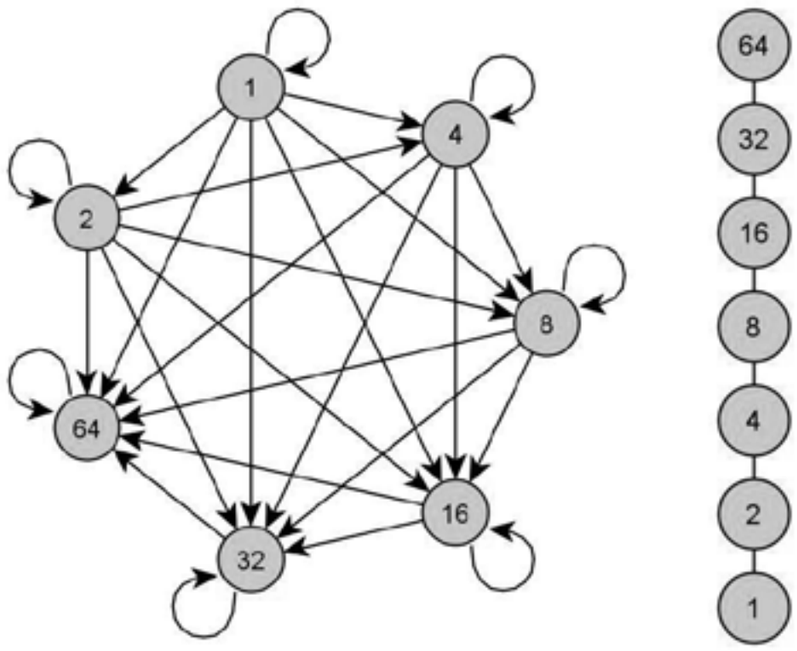
Tam, gdzie istnieje porządek, tam pewne elementy poprzedzają inne elementy. Gdy relacja jest relacją częściowego porządku, jedne elementy poprzedzają inne tylko w ramach podzbiorów. Oznacza to, że pewne podzbiory danego zbioru mogą być porządkowane (sortowane) w sensie tej relacji. Z diagramu Hassego można odczytać, które elementy poprzedzają się wzajemnie, a więc które podzbiory można porządkować w sensie omawianej relacji: {1,3,6}, {1,2,6}, {1,2,4}, {1,5} oraz {1,7}.

Przykład 4: Dla danego zbioru $X = \{1, 2, 4, 8, 16, 32, 64\}$, relacja jest zdefiniowana w następujący sposób: $x \rho y \Leftrightarrow x$ *jest dzielnikiem liczby* y, $x, y \in X$. W stosunku do poprzedniego przykładu różnica polega jedynie na rozpatrywaniu różnych zbiorów. Przyjmując, że kolejnym elementom zbioru X odpowiadają kolejne numery wierszy i kolumn w macierzy relacji, macierz ta ma następującą postać:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (21)$$

Rysunek 12a pokazuje graf relacji (21). Przedstawiona relacja jest zwrotna, antysymetryczna, przechodnia i spójna. W przeciwieństwie do relacji z przykładu 3, która była relacją porządku częściowego, jest to relacja porządku liniowego. Oznacza to, że cały zbiór można uporządkować (posortować) w sensie danej relacji, co widać na diagramie Hassego (rys. 12b).

a) b)



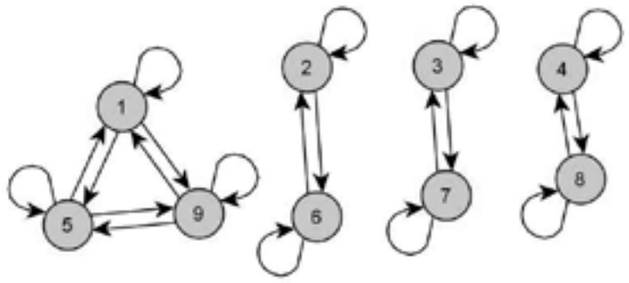
Rysunek 12.
Reprezentacja relacji (21), a) graf relacji, b) diagram Hassego

Przykład 5: Dla danego zbioru $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, relacja jest zdefiniowana w następujący sposób: $x \rho y \Leftrightarrow (x \equiv y) \ mod\ 4$, $x, y \in X$. Tak zdefiniowaną relację nazywa się niekiedy relacją kongruencji modulo 4.

Inaczej mówiąc, zmienne x i y są ze sobą w relacji, gdy mają jednakowe reszty z dzielenia przez 4. Macierz tej relacji wygląda tak:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (22)$$

Na rysunku 13 przedstawiono graf relacji (22). Z analizy macierzy oraz grafu wynika, że jest to relacja zwrotna – każdy element jest w relacji z samym sobą, na przekątnej macierzy znajdują się same jedynki, a w grafie każdy węzeł ma pętlę. Jest to także relacja symetryczna – jeżeli liczba x ma resztę z dzielenia przez 4 identyczną z resztą z dzielenia liczby y przez 4, to także liczba y ma resztę z dzielenia przez 4 identyczną z resztą z dzielenia liczby x przez 4. W grafie pomiędzy różnymi węzłami łuki biegną parami i są skierowane w przeciwnych kierunkach. Relacja jest także przechodnia – zachodzi związek (14): w grafie każda ścieżka o długości dwóch łuków może być zastąpiona przez ścieżkę „na skróty" mającą długość jednego łuku. Jest to więc relacja równoważności. Relacja ta dzieli zbiór na cztery podzbiory będące klasami abstrakcji. Każda z klas zawiera elementy będące ze sobą w relacji, czyli takie, które mają identyczne reszty z dzielenia przez 4: {1,5,9}, {2,6}, {3,7}, {4,8}. Na rysunku 13 klasy abstrakcji są widoczne jako cztery podgrafy spójne.



Rysunek 13.
Graf relacji opisanej macierzą (22)

7 PROGRAM DO BADANIA RELACJI

Macierzowa reprezentacja relacji umożliwia algorytmizację badania własności relacji, a w konsekwencji umożliwia utworzenie programu do badania własności i typów relacji. Taki program mógłby być dobrym treningiem programistycznym, przygotowującym do tworzenia w przyszłości bardziej złożonych programów. Sformułowanie odpowiedniego zadania programistycznego przedstawiono w postaci zbliżonej do formy zadań występujących na zawodach algorytmicznych.

Zadanie:
Koledzy Jasia postanowili zagrać w grę planszową, która składa się z wielu etapów. W każdym z etapów odbywa się rozgrywka, po zakończeniu której, aby przejść na wyższy poziom, trzeba rozwiązać pewne dziwne zadanie matematyczne. Jaś chętnie by wziął udział w grze, ale nie potrafi rozwiązać zadania, gdyż nie uważał w czasie spotkań kółka matematycznego. Potrzebuje pomocy. Pomóż mu, pisząc odpowiedni program.

Rozwiązując zadanie, Jasio powinien odpowiedzieć na pytanie, jakie własności i jaki typ ma pewna relacja dwuczłonowa w zbiorze składającym się z nie więcej niż stu elementów. Za każdym razem relacja jest zdefiniowana w następujący sposób: jeżeli *i*-ty element jest w relacji z elementem *j*-tym, to na wejściu w osobnym wierszu pojawi się rozdzielona spacjami para liczb *i* oraz *j*. Liczba wierszy w zestawie danych jest równa liczbie par (*i*,*j*) dla których zachodzi relacja. O wielkości zbioru, na którym określona jest ta relacja można wnioskować na podstawie największej z liczb *i* albo *j* w parach.

Napisz program, którego wynikiem działania będzie wiersz opisujący przy pomocy odpowiednich skrótów oddzielonych spacjami własności relacji (o ile występują) oraz typ relacji (również, o ile występuje), w następującej kolejności:

- *Z* – zwrotna;
- *PZ* – przeciwzwrotna;
- *S* – symetryczna;
- *AS* – antysymetryczna;
- *PS* – przeciwsymetryczna;
- *P* – przechodnia;
- *SP* – spójna;
- *RR* – relacja równoważności;
- *RCP* – relacja częściowego porządku;
- *RLP* – relacja liniowego porządku;
- *X* – żadna z powyższych.

Przykład:
Wejście:
2 3
2 9
3 4
5 7
5 9
6 7
7 8
8 3
Wyjście:
PZ AS

PODSUMOWANIE

Relacja dwuczłonowa w zbiorze może być przedstawiona przy pomocy macierzy zerojedynkowej, która jest jednocześnie macierzą sąsiedztwa grafu relacji. Graf relacji przez odwoływanie się do wyobraźni, daje możliwość badania relacji w prosty i intuicyjny sposób. Tymczasem reprezentacja macierzowa umożliwia algorytmizację badania własności relacji, a w konsekwencji pozwala na napisanie programu do badania własności i typów relacji. Takie podejście daje możliwość przejścia od abstrakcyjnego modelu matematycznego do konkretnego modelu w postaci algorytmu komputerowego. W wykładzie omówiono algorytmy wykorzystujące tę macierz do badania własności relacji takich jak zwrotność, przeciwzwrotność, symetria, antysymetria, przechodniość oraz spójność relacji. Przedstawiono także przykłady różnych relacji, wraz z określeniem ich własności oraz typów. Stwierdzono, że algorytmy do badania relacji, wykorzystujące operacje na macierzach, mogłyby być dobrym treningiem programistycznym, przygotowującym do tworzenia bardziej złożonych programów. Wobec tego zaproponowano zadanie programistyczne w postaci zbliżonej do zadań występujących na zawodach algorytmicznych. Rozwiązaniem tego zadania byłby program wykorzystujący omówione w wykładzie algorytmy badania relacji.

INFORMACJA

W trakcie przygotowywania niniejszego wykładu do rysowania grafów zastosowano program yEd Graph Editor w wersji 3.6.1.1, dostępny na stronie internetowej http://www.yworks.com, należącej do firmy yWorks GmbH.

LITERATURA

1. Bronsztejn I.N., Siemiendiajew K.A., Musiol G., Mühlig H., *Nowoczesne kompendium matematyki*, WN PWN, Warszawa 2007
2. Rasiowa H., *Wstęp do matematyki współczesnej*, WN PWN, Warszawa 2003
3. Ross K.A., Wright C.R.B., *Matematyka dyskretna*, WN PWN, Warszawa 2003

DODATEK

W wykładzie pojawiły się pojęcia i operacje z obszaru algebry liniowej. Zagadnienia te mogą nie być znane adresatom wykładu. Niniejszy dodatek zawiera kilka podstawowych faktów dotyczących tych zagadnień. Wykład był pisany przy założeniu, że jego odbiorcy znają pojęcie tablicy jednowymiarowej i dwuwymiarowej w języku C++. Tablica jednowymiarowa jest tożsama z wektorem w matematyce. Podobnie, tablica dwuwymiarowa w języku C++ jest tożsama z macierzą. Trzeba tylko pamiętać o tym, że jeżeli w opisie matematycznym składniki n-elementowego wektora oraz n wierszy lub kolumn w macierzy są indeksowane liczbami od 1 do n, to w języku C++ odpowiednie elementy tablicy jednowymiarowej lub wiersze i kolumny tablicy dwuwymiarowej są indeksowane liczbami od 0 do $n - 1$. Wobec powyższego, samo pojęcie wektora oraz macierzy nie będzie tu szerzej omawiane. Natomiast takie pojęcia jak iloczyn skalarny dwóch wektorów, transpozycja macierzy, macierz symetryczna albo algorytm mnożenia macierzy zostaną potraktowane nieco szerzej.

Iloczyn skalarny: operacja przyporządkowująca dwóm wektorom liczbę rzeczywistą. Jeżeli dany jest wektor $a=(a_1, a_2, ..., a_n)$ i wektor $b=(b_1, b_2, ..., b_n)$, to iloczyn skalarny $a \cdot b$ definiuje się jako sumę wszystkich iloczynów $a_i \cdot b_i$:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n .$$

Macierz kwadratowa: macierz, która ma taką samą liczbę wierszy i kolumn. Macierz A przedstawiona niżej jest przykładem macierzy kwadratowej o rozmiarze 3×3:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Tymczasem pokazane niżej macierze B i C nie są macierzami kwadratowymi, gdyż mają różną liczbę wierszy i kolumn:

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 0 & 1 & 2 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 4 & 7 & 0 \\ 2 & 5 & 8 & 1 \\ 3 & 6 & 9 & 2 \end{bmatrix}.$$

Transpozycja macierzy: przekształcenie macierzy polegające na wzajemnej zamianie (wzajemnym przestawieniu) wierszy z kolumnami. Macierz transponowaną względem macierzy X oznacza się jako X^T. Pokazane wyżej macierze B i C są wzajemnymi transpozycjami: $B = C^T$ oraz $C = B^T$. Dla macierzy A jej transpozycja A^T ma postać:

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}.$$

W wyniku transpozycji elementy na głównej przekątnej macierzy nie ulegają zmianie.

Macierz symetryczna: kwadratowa macierz S o rozmiarze $n \times n$ jest symetryczna (względem głównej przekątnej), jeżeli dla każdego $i, j = 1, 2, \ldots, n$ zachodzi związek $S_{ij} = S_{ji}$:

$$S = S^T = \begin{bmatrix} 1 & 6 & 5 \\ 6 & 2 & 4 \\ 5 & 4 & 3 \end{bmatrix}.$$

Mnożenie macierzy: mnożenie macierzy A przez macierz B jest operacją, której efektem jest macierz wynikowa $C := A \cdot B$. Wymaga się przy tym, aby macierz A miała tyle samo kolumn, ile wierszy ma macierz B. Wynikowa macierz C ma tyle wierszy, ile wierszy posiada macierz A i tyle kolumn, ile kolumn ma macierz B. Jeżeli założyć, że A i B są macierzami kwadratowymi o rozmiarze $n \times n$, to także macierz C jest kwadratowa i ma identyczny rozmiar. Rozważa się 3 macierze kwadratowe A, B oraz C, każdą o rozmiarze $n \times n$. Macierze A i B są dane, zaś macierz C jest nieznana:

$$A = \begin{bmatrix} a_{11} & * & * & * & a_{1n} \\ * & * & * & * & * \\ a_{i1} & a_{i2} & \ldots & \ldots & a_{in} \\ * & * & * & * & * \\ a_{n1} & * & * & * & a_{nn} \end{bmatrix} \quad B = \begin{bmatrix} * & * & b_{1j} & * & * \\ * & * & b_{2j} & * & * \\ \ldots & \ldots & \vdots & \ldots & \ldots \\ * & * & \vdots & * & * \\ * & * & b_{nj} & * & * \end{bmatrix} \quad C = \begin{bmatrix} * & * & \vdots & * & * \\ * & * & \vdots & * & * \\ \ldots & \ldots & c_{ij} & \ldots & \ldots \\ * & * & \vdots & * & * \\ * & * & \vdots & * & * \end{bmatrix}$$

Należy znaleźć macierz C jako iloczyn macierzy A i B:

$$C := A \cdot B .$$

Pojedynczy element znajdujący się w i-tym wierszu oraz w j-tej kolumnie macierzy C (oznaczony jako c_{ij}) liczy się, jako iloczyn skalarny i-tego wiersza macierzy A i j-tej kolumny macierzy B. Dla podkreślenia tego faktu, operację tę można zapisać w następujący sposób:

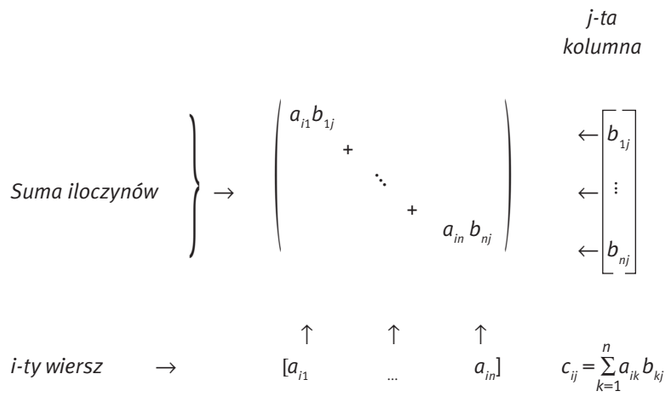
$$\begin{bmatrix} * & * & b_{1j} & * & * \\ * & * & b_{2j} & * & * \\ \ldots & \ldots & \vdots & \ldots & \ldots \\ * & * & \vdots & * & * \\ * & * & b_{nj} & * & * \end{bmatrix}$$

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ a_{i1} & a_{i2} & \ldots & \ldots & a_{in} \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & \vdots & * & * \\ * & * & \vdots & * & * \\ \ldots & \ldots & c_{ij} & \ldots & \ldots \\ * & * & \vdots & * & * \\ * & * & \vdots & * & * \end{bmatrix}$$

Dla znalezienia wszystkich elementów macierzy C należy znaleźć wszystkie iloczyny skalarne odpowiednich wierszy macierzy A oraz odpowiednich kolumn macierzy B. Funkcja służąca do znajdowania elementów macierzy C ma następującą postać:

```
void mnozenie_macierzy(int A[SIZE][SIZE], int B[SIZE][SIZE], int C[SIZE][SIZE], int n)
{  int i,j;
// C := A*B
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
      { C[i][j]=iloczyn_skalarny(i-ty wiersz macierzy A, j-ta kolumna macierzy B);
      }
}
```

Iloczyn skalarny *i*-tego wiersza macierzy *A* i *j*-tej kolumny macierzy *B* można obrazowo przedstawić w postaci następującego schematu:



Iloczyn ten jest sumą *n* iloczynów $a_{ik}b_{kj}$. Ostatecznie algorytm mnożenia macierzy ma następującą postać:

```
void mnozenie_macierzy(int A[SIZE][SIZE], int B[SIZE][SIZE], int C[SIZE][SIZE], int n)
{   int i,j;
//mnożenie macierzy: C := A*B
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
      { //iloczyn skalarny i-tego wiersza macierzy A
        //oraz j-tej kolumny macierzy B
        C[i][j]=0;
        for (k=0;k<n;k++) C[i][j] += A[i][k] * B[k][j];
      }
}
```

NOTA O WYDAWCY

Warszawska Wyższa Szkoła Informatyki jest wyższą uczelnią niepubliczną utworzoną na podstawie decyzji Ministra Nauki i Szkolnictwa Wyższego z dnia 19 lipca 2000 roku.

Uczelnia prowadzi studia wyższe na poziomie inżynierskim, magisterskim i podyplomowym wyłącznie na kierunku informatyka.

Decyzją Ministra Nauki i Szkolnictwa Wyższego z dnia 7 lutego 2008 roku Warszawska Wyższa Szkoła Informatyki otrzymała uprawnienia do prowadzenia studiów I stopnia na kierunku informatyka – na czas nieokreślony.

Decyzją Ministra Spraw Nauki i Szkolnictwa Wyższego z dnia 24 września 2008 roku Warszawska Wyższa Szkoła Informatyki uzyskała uprawnienia do prowadzenia studiów II stopnia – magisterskich.

W Warszawskiej Wyższej Szkole Informatyki aktualnie (2010/2011) kształci się blisko tysiąc trzystu polskich i zagranicznych studentów zdobywających kompetencje zawodowe z zakresu informatyki. Kompetencje te są potwierdzane dyplomami państwowymi – dyplomem inżyniera, magistra oraz dyplomem studiów podyplomowych na kierunku informatyka. Ponadto Uczelnia oferuje możliwość uzyskania cenionych na ryku pracy certyfikatów branżowych ICT (PRINCE2® Foundation, PMI®, CCNA, Audytora wewnętrznego systemu zarządzania bezpieczeństwem informacji wg ISO 27001, Projektanta zabezpieczeń sieci teleinformatycznych wg ISO 27001, Menedżera zarządzania bezpieczeństwem informacji, MCTS oraz MCITP).

Uczelnia prowadzi w szerokim zakresie otwarte studia informatyczne (kursy, szkolenia, warsztaty) dla uczniów szkół średnich – między innymi w ramach projektu Informatyka+, dla absolwentów szkół wyższych oraz dla wszystkich osób zainteresowanych pogłębieniem swojej wiedzy i kwalifikacji informatycznych, niezależnie od wieku oraz stopnia znajomości informatyki.

Warszawska Wyższa Szkoła Informatyki prowadzi działalność o charakterze dydaktycznym i naukowo-badawczym, ze szczególnym uwzględnieniem problematyki edukacyjnej i kształcenia w zakresie informatyki, jako dziedziny o potencjalnie największych perspektywach na rynku pracy. Podstawowym zadaniem prowadzonych w Uczelni studiów oraz innych form kształcenia jest wyposażenie studentów w wiedzę teoretyczną oraz w umiejętności praktyczne z dziedziny informatyki, które pozwolą na uzyskanie zatrudnienia zgodnego z posiadanymi kwalifikacjami, oczekiwaniami i predyspozycjami. W dłuższym horyzoncie czasowym zdobyta w Warszawskiej Wyższej Szkole Informatyki wiedza i umiejętności powinny również sprzyjać dalszemu doskonaleniu własnego warsztatu pracy poprzez kontynuację nauki w różnych formach kształcenia ustawicznego lub poprzez samokształcenie.

Więcej informacji o Uczelni na stronie www.wwsi.edu.pl