HOMO INFORMATICUS

czyli człowiek w zinformatyzowanym świecie

redakcja naukowa
Maciej M. Sysło

Warszawa 2012

Publikacja współfinansowana ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego

CZŁOWIEK – NAJLEPSZA INWESTYCJA

HOMO INFORMATICUS
czyli człowiek w zinformatyzowanym świecie

Redaktor naukowy: prof. dr hab. Maciej M. Sysło
Redakcja i korekta: 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

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

Wydanie pierwsze

ISBN 978-83-921270-6-2

Projekt graficzny: Marzena Kamasa

Druk: PPH ZAPOL

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

SPIS TREŚCI

Wprowadzenie 5

1. *Logika na co dzień*, Andrzej Szałas 9
2. *Języki – komunikacja z człowiekiem i maszynami*, Grzegorz Jakacki . . . 39
3. *Kubełkowe struktury danych*, Krzysztof Diks, Wojciech Śmietanka . . . 77
4. *Czy wszystko można obliczyć. Łagodne wprowadzenie do złożoności obliczeniowej*, Jarosław Grytczuk 95
5. *Homo informaticus colligens, czyli człowiek zbierający dane*, Krzysztof Stencel 117
6. *Elektroniczny biznes – mariaż ekonomii i informatyki*, Wojciech Cellary 147
7. *Informatyka medyczna*, Ryszard Tadeusiewicz 165
8. *Komputer – obiekt i narzędzie edukacji. Poznawcze walory informatyki i technologii informacyjno-komunikacyjnej*, Maciej M. Sysło 197
9. *Kariera w ICT – dobry wybór*, Andrzej Żyławski 231
10. *Historia rachowania – ludzie, idee, maszyny. Historia mechanicznych kalkulatorów*, Maciej M. Sysło 267

Skorowidz 309

HOMO INFORMATICUS
czyli człowiek w zinformatyzowanym świecie

I TY możesz stać się *Homo informaticus* – człowiekiem rozumiejącym informatykę, jej techniki i technologie, jak i mechanizmy rządzące jej zastosowaniami.

Biologicznie, człowiek to *Homo sapiens* – istota myśląca. Rozwój przemysłu w XIX wieku doprowadził do pojawienia się *Homo faber* – człowieka pracującego, kształtującego otoczenie i panującego nad nim za pomocą różnych narzędzi fizycznych, jako przeciwność Boga Twórcy – *deus faber*, i w opozycji do *Homo ludens* – człowieka bawiącego się, zainteresowanego rozrywką, spędzającego głównie czas wolny, któremu znacznie przysłużył się rozwój wynalazków drugiej połowy XX wieku: filmu, telewizji i Internetu.

Dzisiaj jest potrzebny *Homo informaticus* – człowiek rozumiejący prawa i mechanizmy związane z informatyką oraz znaczenie tej dziedziny dla ludzkości, świadomie i celowo korzystający przy tym z infrastruktury budowanej ze środków (sprzętu) i narzędzi (oprogramowania) informatycznych, biorący udział w rozwoju tej infrastruktury działaniami w obszarach swoich zawodowych i osobistych potrzeb i zainteresowań.

Niniejsza publikacja stwarza okazję i zachęca do pogłębionego spojrzenia na kilka wybranych obszarów informatyki przez pryzmat współczesnych jej zastosowań, pojawiających się w rękach *Homo faber* – człowieka pracującego niemal w każdej dziedzinie, i będących na jego usługach, a także gdy staje się *Homo ludens* i spędza czas wolny, bawiąc się w najprzeróżniejszy sposób. Każdy artykuł w sposób poglądowy, niemal „bez wzorów", wprowadza w często bardzo złożony świat informatyki, bez zbędnej teorii prowadząc do rozwiązań w wybranych obszarach, które mają obecnie duży wpływ na poziom życia, dobrobytu obywateli i całych społeczeństw. Zwraca się przy tym uwagę na historyczny rozwój pojęć i osiągnięć informatycznych, co pokazuje, że informatyka – jeszcze bez komputerów i swojej nazwy – ma długą historię, toczącą się równolegle w wielu dziedzinach wraz z rozwojem nauki i techniki oraz nieskrępowanej myśli ludzkiej.

Książka jest adresowana głównie do dwóch grup czytelników. Osoby wahające się z podjęciem decyzji o swojej przyszłej karierze zawodowej – ma zachęcić do obrania dalszej drogi kształcenia i kariery zawodowej związanej z informatyką i jej zastosowaniami. Natomiast ci, którzy już wybrali informatykę, jako specjalność swojego dalszego rozwoju, znajdą w tekstach przystępne wprowadzenie do aktualnej tematyki w wybranych jej działach.

Krótki rzut oka na zawartość książki.

Człowiek stara się wykorzystywać komputery do analizy sytuacji i wyciągania na tej podstawie wniosków – tak powstają systemy inteligentne, często samodzielnie wnioskujące i podejmujące decyzje. Podstawową nauką zajmującą się metodami wnioskowania jest logika i jej właśnie jest poświęcony **rozdział pierwszy**. Omówiono w nim metodę rezolucji oraz wnioskowanie na podstawie baz wiedzy. Metody te znajdują zastosowania w naukach ścisłych, ekonomicznych i humanistycznych, a w informatyce – są stosowane w robotyce, systemach eksperckich i decyzyjnych.

Rozdziału drugi jest poświęcony językom komunikacji, naturalnym i sztucznym. Ten drugi typ języków rozpowszechnił się w związku z komunikacją z maszynami i szczególne wśród nich miejsce zajmują języki programowania maszyn (komputerów). Pierwsze języki były szorstkie i niewygodne do stosowania. Z biegiem czasu powstawały języki coraz doskonalsze. Rozwinięto metody opisu języków pomagające zachować precyzję i poprawność wypowiedzi. Stworzono narzędzia umożliwiające tłumaczenie języków wygodniejszych dla człowieka na języki bardziej pasujące do maszyn. W tym rozdziale przedstawiono przykłady różnych języków sztucznych, zademonstrowano kilka prostych metod ich projektowania oraz opisano historię rozwoju komputerów z punktu widzenia komunikacji programisty z maszyną.

W **rozdziale trzecim** przedstawiono struktury danych, które są proste koncepcyjnie, stosunkowo łatwe w implementacji i nieźle zachowują się w praktyce. Pomimo swojej prostoty opis tych struktur trudno znaleźć w podręcznikach algorytmiki, dlatego opisano je tutaj i powinny zainteresować każdego, kto chciałby poszerzyć swoją wiedzę algorytmiczną. Zaprezentowane idee umożliwiają otrzymanie algorytmów działających o czynnik pierwiastkowy szybciej. Oprócz przedstawienia ciekawych struktur danych i ich zastosowań, nie mniej ważny jest proponowany sposób rozwiązywania problemów algorytmicznych, polegający na formułowaniu najpierw algorytmów w sposób abstrakcyjny, a następnie poszukiwaniu najlepszych metod implementacji wykorzystywanych w opisie algorytmów abstrakcyjnych konstrukcji.

W **rozdziale czwartym** snuta jest opowieść o jednym z siedmiu problemów milenijnych – zagadnieniu **Czy P=NP?**. Chociaż to pytanie jest centralnym problemem teorii obliczeń – leżącej na styku matematyki i informatyki – jego znaczenie wykracza daleko poza te dyscypliny, dotykając fundamentalnych, filozoficznych pytań o naturę świata i ludzkiego umysłu. Jest przy tym rzeczą zaskakującą, że w istocie problem Czy P=NP? można sprowadzić do prostych układanek, takich jak popularne sudoku – czy istnieje efektywny sposób rozwiązania sudoku dowolnego rozmiaru? Na te i podobne pytania wciąż nie znamy odpowiedzi. Dzięki osiągnięciom teorii obliczeń wiemy jednak, że znalezienie takiego sposobu przyniosłoby zarazem rozwiązanie wszystkich podobnych pro-

blemów na świecie. Wydaje się to zatem niemożliwe, lecz dowodu matematycznego owej niemożliwości jak dotąd nie znaleziono.

Dane zgromadzone przez przedstawicieli *Homo informaticus* mierzy się setkami eksabajtów. Aby można było je przetwarzać, ktoś musi tymi danymi zarządzać, przechowywać je oraz zabezpieczać je przed niepowołanym dostępem. Tym właśnie zajmuje się bliski ich ziomek – *Homo informaticus colligens*, o którym jest mowa w **rozdziale piątym**. Omówione zostały różne rodzaje baz danych, od historycznych po obecnie stosowane oraz dopiero wchodzące na rynek, jak bazy: sieciowe, hierarchiczne i obiektowe. Dokładnie omówiono obecnie najpowszechniej stosowane bazy relacyjne wraz z i ich podstawowym językiem SQL. Rozdział kończy opis baz nurtu NOSQL, takich jak: bazy klucz-wartość, dokumentowe i grafowe.

Rozdział szósty jest poświęcony związkom informatyki z ekonomią. Przedstawiono w nim, jak bardzo zmieniło się zarządzanie przedsiębiorstwami, kontakty przedsiębiorstw z klientami i współpraca przedsiębiorstw dzięki zastosowaniu metod oraz narzędzi informatyki i telekomunikacji. Omówiono specyficzne cechy elektronicznego biznesu, w szczególności zalety elektronicznego biznesu typu przedsiębiorstwo-klient oraz przedsiębiorstwo-przedsiębiorstwo. Skupiono uwagę na organizacjach wirtualnych powstających dzięki informatyce i telekomunikacji. Na końcu przedstawiono dwa nowe paradygmaty informatyczne, które mają bezpośredni wpływ na elektroniczny biznes: architekturę usługową SOA i przetwarzanie w chmurze. Konkludując, mariaż ekonomii i informatyki prowadzi do elektronicznego biznesu i elektronicznej gospodarki, w której powstają nowe jakościowo miejsca pracy o ogromnych możliwościach.

W **rozdziale siódmym** zarysowano obszary zastosowań informatyki w medycynie, gdzie komputer staje się sojusznikiem lekarza w jego walce o zdrowie i życie ludzkie. Wyposażony w odpowiednią aparaturę komputerową, współczesny lekarz może lepiej i szybciej rozpoznawać procesy chorobowe trapiące pacjenta, a także trafniej ustalać sposób leczenia wykrytej choroby. Dzisiejsza skomputeryzowana aparatura medyczna pozwala lekarzom działać skutecznie i pewnie, eliminując przez to cierpienia pacjentów i oddalając ryzyko powikłań. Szanse sukcesu w walce o zdrowie pacjentów znacząco zwiększa dzisiaj informatyka, dostarczając nowych i doskonalszych narzędzi komputerowych dla medycyny.

W **rozdziale ósmym**, przedmiotem rozważań jest komputer jako obiekt i narzędzie edukacji – zarówno uczy się o nim, jak i stosuje w nauczaniu innych dziedzin. Komputer jest wyjątkową technologią wśród technologii kształcenia, odgrywającą coraz bardziej rolę pomocy intelektualnej. Pierwsza część rozdziału jest opisem krótkiej historii komputerów w edukacji, w drugiej – przedstawiono obecny stan edukacji informatycznej i kształcenia informatycznego w szkołach, a na końcu nakreślono wizję niedalekiej przyszłości dla technologii w edukacji. Ten rozdział jest wyraźnie adresowany do uczniów, jednym z najważniejszych

priorytetów współczesnych systemów edukacji jest bowiem personalizacja, czyli dostosowanie kształcenia do indywidualnych zainteresowań, możliwości i potrzeb uczniów, a tego celu nie można osiągnąć bez udziału uczących się.

Rozdział dziewiąty zawiera omówienie uwarunkowań związanych z wyborem, realizacją i rozwojem kariery zawodowej w ICT. Ten wybór powinien być poprzedzony oceną własnych zainteresowań i predyspozycji. Warto też uświadomić sobie, jakie czynniki oraz motywy mają znaczący wpływ na wybór przyszłego zawodu. Ważne jest też zapoznanie się z wymogami kompetencyjnymi dla zawodu informatyka i ich konfrontacja z własnymi oczekiwaniami i możliwościami. Wybór zawodu zawsze wiąże się z koniecznością wyboru ścieżki edukacyjnej, która gwarantuje zdobycie niezbędnych do wykonywania zawodu umiejętności i kwalifikacji, potwierdzonych stosownymi dyplomami i certyfikatami. Warto również wstępnie ocenić możliwości kształcenia ustawicznego oraz awansu zawodowego w obszarze ICT. W podjęciu decyzji o wyborze profesji informatyka i dalszym rozwoju kariery zawodowej z pewnością przydatne jest zapoznanie się z podstawowymi informacjami na temat rynku pracy ICT oraz perspektywami jego rozwoju.

W ostatnim, **dziesiątym rozdziale** pochylamy się nad historią informatyki, chociaż dla wielu osób ta dziedzina nie ma jeszcze swojej historii. Współczesny komputer elektroniczny jest jednak ukoronowaniem wspólnych wysiłków cywilizacji i pokoleń, rozwijających w ciągu wieków wiele różnych dziedzin nauki i techniki, które kształtowały również sposoby rachowania i konstrukcje urządzeń wspomagających złożone i masowe obliczenia. Od zarania ludzkości bowiem człowiek starał się ułatwiać sobie prowadzenie rachunków i obliczeń, posługując się przy tym różnymi urządzeniami. Tak rodziły się abaki (liczydła), kalkulatory i wreszcie komputery. Jeśli nawet uznaje się, że informatyka ma swoją historię, to na ogół niewielką uwagę przywiązuje się do urządzeń mechanicznych. A przecież twórcami pierwszych takich maszyn były nieprzeciętne umysły XVII wieku: John Napier, Blaise Pascal i Gottfried Leibniz. Mechanizmy użyte przez Pascala i Lebniza były stosowane w kalkulatorach mechanicznych do ostatnich dni tych urządzeń. Co więcej, ich rozwój i produkcja doprowadziły do sytuacji w latach 50.-70. minionego wieku, w której każdy człowiek potrzebujący takiego urządzenia mógł sobie je nabyć, podobnie jak dzisiaj każdy może mieć komputer osobisty. Ten rozdział jest poświęcony najważniejszym osiągnięciom w dziedzinie kalkulatorów mechanicznych i kilku ważnym ideom z okresu ich świetności, które można dzisiaj odnaleźć, w nieco przetworzonej postaci, wśród najważniejszych mechanizmów napędzających rozwój współczesnej technologii i informatyki.

Wrocław, we wrześniu 2012 roku.

Maciej M. Sysło

Andrzej Szałas

Instytut Informatyki
Uniwersytet Warszawski

Logika na co dzień

Codzienna działalność człowieka wymaga ciągłej analizy sytuacji i wyciągania wniosków. Podobnie dzieje się w systemach informatycznych, w szczególności w systemach inteligentnych, samodzielnie wnioskujących i podejmujących decyzje. Podstawową nauką zajmującą się metodami wnioskowania jest logika i jej właśnie poświęcony jest niniejszy rozdział. Omówimy najpierw podstawowe mechanizmy wnioskowania, w tym rezolucję – najpopularniejszą metodę automatycznego wnioskowania. Następnie przejdziemy do regułowych języków programowania, dla których impulsem była metoda rezolucji. Od uzasadniania poprawności rozumowań przejdziemy do wnioskowania na podstawie baz wiedzy. Obok zastosowań w naukach ścisłych, ekonomicznych czy humanistycznych, metody te mają ogromne znaczenie w informatyce, w tym w robotyce, systemach eksperckich i decyzyjnych.

1. Wprowadzenie

Jeśli jestem głodny, staram się znaleźć coś do zjedzenia. Jeśli chcę przejść na drugą stronę ulicy, rozglądam się i – jeśli sytuacja jest bezpieczna – przechodzę. Jeśli chcę wyszukać dane dotyczące logiki w informatyce, wpisuję w okienko wyszukiwarki Google frazę *logika informatyka*. Jeśli chcę policzyć pierwiastek równania **ax+b=c**, wykonuję instrukcję **jeśli** a=/=0 **to** x=(c–b)/a, być może wpisując ją do arkusza kalkulacyjnego jako =JEŻELI(A1<>0;(C1–B1)/a) lub w języku programowania, takim jak C, C++ czy Java, np. jako **if** a!=0 {x=(c–b)/a}. Jeśli mam podwyższoną temperaturę i jestem przeziębiony – biorę odpowiedni lek. Jeśli nie umiem zdiagnozować choroby, albo jest ona zbyt poważna na samodzielną terapię – idę do lekarza. Jeśli prowadzę pojazd, dojeżdżam do skrzyżowania równorzędnego i z prawej strony nadjeżdża inny pojazd, ustępuję mu pierwszeństwa.

Co łączy te przykłady?

Zauważmy, że wszystkie powyższe zdania zaczynają się od słowa „jeśli". Nasza codzienna działalność bardzo często polega na używaniu zdań tego typu. W logice nazywamy je **implikacjami**, a – przy pewnych ograniczeniach – **regułami**. Do reguł przejdziemy nieco później. Teraz przyjrzyjmy się implikacji. Jest ona powszechnie stosowana – wyraża się przy jej użyciu reguły postępowania, przepisy prawne, sposoby dochodzenia do decyzji, diagnozy lekarskie, strategie gier itd. Wnioskowanie jest często oparte na implikacjach – na podstawie przesłanek wyciągamy wnioski, które stają się przesłankami w kolejnych fazach wnioskowania. Dla zilustrowania tej metody przyjrzyjmy się następującemu rozumowaniu:

(a) pojadę dziś do pracy autobusem lub tramwajem,
(b) jeśli pada deszcz, nie wybieram autobusu (przystanek autobusowy jest dalej niż tramwajowy i bardziej zmoknę),
(c) pada deszcz,
(d) wniosek: pojadę do pracy tramwajem.

Skoro pada deszcz, nie wybieram autobusu. Tego wniosku używam w dalszym rozumowaniu: ponieważ pojadę autobusem lub tramwajem, a nie wybieram autobusu, pozostaje jedynie tramwaj. Uzyskany wniosek uznamy więc za poprawny. Ale na jakiej podstawie? Czy to rozumowanie można uzasadnić metodami takimi, jakie przyjmuje się w naukach ścisłych?

Nauka zajmująca się modelowaniem wnioskowania i jego badaniem nazywa się **logiką** (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 ona istotną rolę w średniowieczu, burzliwy jej rozwój datuje się od końca XIX wieku. George Boole, Charles Peirce, John Venn, potem Bertrand Russell czy Gottlob Frege są wybitnymi logikami z tego okresu. Informatyczne 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, Gerhard Gentzen, Kurt Gödel czy Alfred Tarski. Pierwsze matematyczne modele maszyn matematycznych zawdzięczamy kontynuacji prac tych logików, prowadzonych przez Emila Posta, Alonzo Churcha, Stephena Kleenego (prekursorzy języków funkcyjnych), jak również Alana Turinga czy Claude'a Shannona (prekursorzy języków imperatywnych).

Jak wspomnieliśmy – implikacja jest podstawą wnioskowań, jest więc ważnym elementem logik. Rozumowania przeprowadzane w matematyce, informatyce, fizyce czy w życiu codziennym opierają się na zdaniach wyrażających interesujące nas prawdy. Zdania te dzielimy na **proste** (**atomowe**) i złożone. Zdania proste wyrażają pewne fakty, jak „jestem głodny", „chcę przejść na drugą stronę ulicy". Natomiast zdania złożone są tworzone z innych zdań właśnie przy pomocy spójników. Implikacja jest jednym z takich spójników: na przykład zdanie „jeśli jestem głodny, to staram się znaleźć coś do zjedzenia" powstaje z dwóch zdań prostych poprzez ich połączenie spójnikiem „jeśli... to...". Implikacja występuje także w arkuszach kalkulacyjnych i w językach programowania jako podstawa instrukcji warunkowych „**if** {}".

Innymi typowymi spójnikami są **negacja** (nie), **koniunkcja** (i), **alternatywa** (lub) oraz **równoważność** (wtedy i tylko wtedy, gdy). Również te spójniki są wszechobecne w języku naturalnym, a więc i w codziennym wnioskowaniu. Występują również w naukach ścisłych. Informatyka, leżąca na skrzyżowaniu nauk ścisłych i codziennych aktywności człowieka, także nie może się bez nich obyć.

2. Modelowanie

W nauce **modelem** nazywamy uproszczony opis rzeczywistości, pozwalający skutecznie wnioskować, a jednocześnie pomijający szczegóły mniej istotne z punktu widzenia prowadzonych rozumowań. Prawa dynamiki Isaaca Newtona opisują podstawowe zasady rządzące ruchem i siłami. Tworzą prosty model, skuteczny dopóki nie mamy do czynienia z prędkościami bliskimi prędkości

światła. Niels Bohr stworzył słynny model atomu, prosty pojęciowo i oddający podstawowe zasady zachowania atomu.

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 złożone procesy przepływu wody w rurach wodociągowych itd. Prostota modelu 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.

Wnioskowanie zawsze bazuje na jakimś mniej lub bardziej abstrakcyjnym modelu rzeczywistości.

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. Przedmioty o innych kolorach robot powinien przepuszczać dalej. 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:

(a) jeśli obserwowany obiekt jest zielony, to przenieś go na taśmę z lewej strony,
(b) 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. 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. Daje jednocześnie wiedzę o warunkach, jakie powinno spełniać otoczenie robota, by ten mógł działać efektywnie wykorzystując swoje możliwości.

Możemy więc dostrzec, 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...). 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.

W przykładzie z taśmami produkcyjnymi interesują nas przedmioty, których atrybutem jest kolor. Gdybyśmy nieco skomplikowali zadanie, zakładając, że na lewo przenosimy zielone owoce, na prawo – czerwone pomidory, zaś inne obiekty przepuszczamy dalej, zaczynają się pojawiać pojęcia takie jak „owoc", „pomidor", których atrybutem jest kolor.

3. Od modelu do bazy wiedzy

Zwykle model opisujemy początkowo w nieformalny sposób, najczęściej w języku naturalnym. Aby można go było wykorzystać we wnioskowaniu logicznym – musimy ten nieformalny opis przetworzyć na opis wykorzystujący notację logiczną. Powstaje w ten sposób pewna **baza wiedzy**. Następnie jesteśmy zainteresowani wyciąganiem wniosków wynikających z tej bazy wiedzy. Bazy wiedzy mogą opisywać stosunkowo prostą rzeczywistość, jak meble w danym pokoju, poprzez sytuacje dużo bardziej skomplikowane, jak opis chorób, czy zasad ruchu drogowego, po naprawdę trudne do obsługi i wnioskowania, jak obejmujące wybrane teorie matematyczne, fizyczne, chemiczne, biologiczne itd.

Co to znaczy, że wyciągamy wnioski z bazy wiedzy? Otóż interesuje nas, jakie wnioski wynikają z wiedzy zgromadzonej w danej bazie. Jeśli Δ jest bazą wiedzy, zaś W – interesującym nas wnioskiem, badamy, czy z Δ można wywnioskować W, czyli czy Δ implikuje W. Odnosząc się do prostej bazy wiedzy, dotyczącej wyboru pomiędzy autobusem i tramwajem, samą bazę stanowią zdania (a), (b), (c), zaś wnioskiem jest zdanie (d). Podkreślmy, że jeśli baza wiedzy zawiera wiele zdań, przyjmujemy, że zdania te są połączone spójnikiem „i". Baza składająca się ze zdań (a), (b), (c) jest więc rozumiana jako zdanie „(a) i (b) i (c)".

Z praktycznego punktu widzenia potrzebujemy więc języka, w którym będziemy opisywali wiedzę i wnioski, oraz mechanizmów wnioskowania, najlepiej dających się skutecznie implementować. Wprowadzimy więc logikę tradycyjnie nazywaną **klasycznym rachunkiem zdań**.

4. Klasyczny rachunek zdań

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

4.1. Język klasycznego rachunku zdań

Aby wprowadzić rachunek zdań, zaczyna się od **zmiennych zdaniowych** reprezentujących wartości logiczne **prawda**, **fałsz**, a zarazem zbiory obiektów mających cechy opisywane tymi zmiennymi (w tym ujęciu zmienne zdaniowe odpowiadają cechom, czyli atrybutom obiektów). Możemy na przykład użyć zmiennych „czerwony", „zielony". Mogą one przyjąć wartości prawda, fałsz w zależności od tego, czy dany obiekt jest czerwony (zielony). Możemy też patrzeć na te zmienne, jako na reprezentujące odpowiednio czerwone i zielone obiekty.

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.

4.2. Tablice logiczne

Znaczenie (czyli **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. Przyjmujemy też notację dla tych spójników: $\neg$ (negacja), $\wedge$ (koniunkcja), $\vee$ (alternatywa), $\Rightarrow$ (implikacja) oraz $\Leftrightarrow$ (równoważność).

Mamy następującą tablicę dla negacji:

p	$\neg p$
0	1
1	0

oraz kolejną dla pozostałych rozważanych spójników:

		koniunkcja	**alternatywa**	**implikacja**	**równoważność**
p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Te tablice dają podstawę bardzo skutecznemu mechanizmowi sprawdzania poprawności wnioskowania dla formuł ze stosunkowo niewielką liczbą zmiennych zdaniowych. Mianowicie konstruujemy tablice logiczne, w których:

- w pierwszych kolumnach umieszczamy zmienne zdaniowe,
- w kolejnych kolumnach umieszczamy wyrażenia występujące w badanej formule ułożone w ten sposób, by wartość danego wyrażenia można było policzyć na podstawie wartości wyrażeń występujących we wcześniejszych kolumnach.

Wiersze w tabeli wypełnia się zaczynając od kolumn odpowiadających zmiennym zdaniowym. Wypełniamy te kolumny wszystkimi możliwymi układami wartości logicznych, jakie można przypisać zmiennym zdaniowym. W następnych krokach wyliczamy 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. Jest nazywana **kontrtautologią**, jeśli zawsze przyjmuje wartość 0 (fałsz).

Dla przykładu sprawdźmy jakie wartości przyjmuje formuła $\neg(p \vee q) \Rightarrow \neg p$. Tworzymy tablicę logiczną:

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg(p \vee q) \Rightarrow \neg p$
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**.

Zasada dowodzenia przez doprowadzanie do sprzeczności, nazywana też dowodem nie wprost, została odkryta już przed niemal 2,5 tysiącami lat. Przypisuje się ją Zenonowi z Elei (495 r. p.n.e.) choć – może mniej jawnie – była używana przez jego poprzedników. Zenon z Elei znany był jak polemista, doskonalący sztukę prowadzenia sporów. Słynne są również jego paradoksy.

Zasada ta jest również bardzo ważna współcześnie. Stosuje się ją w matematyce, ale też odgrywa zasadniczą rolę we wnioskowaniu z baz wiedzy. Będziemy z niej korzystać przy omawianiu metody rezolucji, najpopularniejszej współczesnej metody automatycznego wnioskowania. Zasada sprowadzania do sprzeczności mówi, że w celu wykazania implikacji $p \Rightarrow q$, zaprzeczamy q i wykazujemy, że takie zaprzeczenie prowadzi do fałszu. Innymi słowy, chcemy wykazać, że formuła:

$$(p \Rightarrow q) \Leftrightarrow ((p \wedge \neg q) \Rightarrow 0)$$

jest tautologią. Konstruujemy odpowiednią tablicę logiczną:

p	q	$p \Rightarrow q$	$\neg q$	$p \wedge \neg q$	$(p \wedge \neg q) \Rightarrow 0$	$(p \Rightarrow q) \Leftrightarrow ((p \wedge \neg q) \Rightarrow 0)$
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

W kolumnie odpowiadającej badanej formule występuje wyłącznie wartość 1, badana formuła jest więc rzeczywiście tautologią.

4.3. 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. Wykonajmy teraz prosty rachunek. Załóżmy, że mamy formułę mającą 100 zmiennych. Tabela logiczna będzie więc miała 2^{100} wierszy, bo tyle jest różnych przypisań dwóch wartości logicznych stu zmiennym. Ile czasu spędziłby na obliczeniach bardzo szybki komputer, wykonujący, powiedzmy 2^{54} operacji na sekundę (to więcej niż 10^{16} operacji na sekundę)?

Aby wygenerować 2^{100} wierszy potrzebujemy:

- więcej niż $2^{100}/2^{54}$ $(=2^{46})$ sekund, czyli więcej niż $2^{46}/60$ minut,
- to więcej niż $2^{46}/2^{6}$ $(=2^{40})$ minut
- i więcej niż $2^{40}/2^{6}$ $(=2^{34})$ godzin,
- i więcej niż $2^{34}/2^{5}$ $(=2^{29})$ dób,
- to z kolei więcej niż 2^{20} lat (czyli więcej niż milion lat).

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. 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 oraz 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:

$$p \vee \neg q \vee \neg r \vee s$$

zaś nie jest: $p \vee \neg\neg q$ (bo $\neg\neg q$ nie jest negacją zmiennej, a negacją negacji zmiennej). Nie jest też klauzulą $p \wedge \neg r$ (ponieważ jest to koniunkcja, a nie alternatywa).

Przyjmuje się, że pusta klauzula (niemająca żadnych wyrażeń) jest równoważna fałszowi (czyli 0).

5.1. Dlaczego klauzule są ważne?

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

$$(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \Rightarrow (r_1 \vee r_2 \vee \ldots \vee r_m)$$

jest równoważna klauzuli:

$$\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_k \vee r_1 \vee r_2 \vee \ldots \vee r_m.$$

Implikacje wspomnianej postaci nazywamy **regułami**. Podobnymi regułami posługujemy się codziennie, np.:

- gorączka ∧ kaszel ⇒ przeziębienie ∨ grypa,
- deszcz ∧ bezwietrznie ⇒ parasol ∨ kurtka_z_kapturem,
- deszcz ∧ wiatr ⇒ samochód.

Klauzule są maszynową reprezentacją reguł, wygodną z punktu widzenia stosowania metody rezolucji.

5.2. Przekształcanie formuł do postaci klauzulowej

Okazuje się, że każdą formułę rachunku zdań można przekształcić do równoważnej jej formuły w postaci klauzulowej. Aby dokonać takiego przekształcenia, zastępujemy podformuły występujące w formule wejściowej zgodnie z podanymi poniżej zasadami aż do momentu uzyskania koniunkcji klauzul.

Łatwo się przekonać, stosując metodę tablic logicznych, że formuła zastępowana przyjmuje zawsze tę samą wartość logiczną co formuła ją zastępująca:

Formuła zastępowana	Formuła zastępująca
$A \Leftrightarrow B$	$(\neg A \vee B) \wedge (A \vee \neg B)$
$A \Rightarrow B$	$\neg A \vee B$
$\neg\neg A$	A
$\neg(A \wedge B)$	$\neg A \vee \neg B$
$\neg(A \vee B)$	$\neg A \wedge \neg B$
$A \vee (B \wedge C)$	$(A \vee B) \wedge (A \vee C)$
$(B \wedge C) \vee A$	$(A \vee B) \wedge (A \vee C)$

Zakładamy, że usuwane są:

- zbędne nawiasy (np. $((A))$ można zastąpić przez (A), zaś $(A \vee B) \vee C$ – przez $A \vee B \vee C$,

- powtórzenia wyrażeń występujących w alternatywach (np. $(A \vee A \vee B)$ można zastąpić przez równoważną formułę $(A \vee B)$),
- powtórzenia wyrażeń występujących w koniunkcjach (np. $(A \wedge A \wedge B)$ można zastąpić przez równoważną formułę $(A \wedge B)$).

Dla przykładu rozważmy formułę: $(\neg(p \wedge \neg q) \vee (p \Rightarrow r)) \vee (r \Leftrightarrow \neg s)$. Jej sprowadzenie do postaci klauzulowej może składać się z kroków:

- $((\neg p \vee \neg\neg q) \vee (p \Rightarrow r)) \vee (r \Leftrightarrow \neg s)$,
- $((\neg p \vee \neg\neg q) \vee (\neg p \vee r)) \vee (r \Leftrightarrow \neg s)$,
- $(\neg p \vee \neg\neg q \vee \neg p \vee r) \vee (r \Leftrightarrow \neg s)$,
- $(\neg p \vee q \vee \neg p \vee r) \vee (r \Leftrightarrow \neg s)$,
- $(\neg p \vee q \vee \neg p \vee r) \vee ((\neg r \vee \neg s) \wedge (r \vee \neg\neg s))$,
- $(\neg p \vee q \vee \neg p \vee r) \vee ((\neg r \vee \neg s) \wedge (r \vee s))$,
- $(\neg p \vee q \vee r) \vee ((\neg r \vee \neg s) \wedge (r \vee s))$,
- $(\neg p \vee q \vee r \vee \neg r \vee \neg s) \wedge (\neg p \vee q \vee r \vee r \vee s)$,
- $(\neg p \vee q \vee r \vee \neg r \vee \neg s) \wedge (\neg p \vee q \vee r \vee s)$.

Wynikową formułę można z łatwością uprościć zauważając, że pierwsza klauzula ma zawsze wartość prawda (ponieważ zawiera alternatywę $r \vee \neg r$):

- $1 \wedge (\neg p \vee q \vee r \vee s)$,

czyli:

- $(\neg p \vee q \vee r \vee s)$.

5.3. Metoda rezolucji

Metoda rezolucji wykorzystuje zasadę dowodzenia przez doprowadzanie do sprzeczności.

Metodę rezolucji wprowadził John Alan Robinson w 1965 roku. Była ona jednak stosowana już w XIX wieku, pod inną nazwą i w zakresie ograniczonym do formuł wybranej postaci. Wykorzystywał ją Charles Lutwidge Dodgson, znany także jako Lewis Carroll, autor m.in. *Alicji w krainie czarów*. W jego podręczniku *Symbolic Logic* (1896 r.) metoda ta nosi nazwę *method of underscoring*.

Metoda rezolucji jest w swojej istocie oparta na przechodniości implikacji, czyli na regule mówiącej że:

z przesłanek $(p \Rightarrow q)$ oraz $(q \Rightarrow r)$ mamy prawo wnioskować $(p \Rightarrow r)$.

Na przykład:

z przesłanek (deszcz ⇒ mokro) oraz (mokro ⇒ ślisko)
możemy wywnioskować, że (deszcz ⇒ ślisko).

Sformułowanie tej reguły w postaci klauzulowej jest następujące:

z przesłanek $(\neg p \vee q)$ oraz $(\neg q \vee r)$ wywnioskuj $(\neg p \vee r)$.

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

$$\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_k \vee r_1 \vee r_2 \vee \ldots \vee r_m$$
$$p_1 \vee \neg q_1 \vee \ldots \vee \neg q_n \vee s_1 \vee s_2 \vee \ldots \vee s_m$$

mamy prawo wywnioskować klauzulę:

$$\neg p_2 \vee \ldots \vee \neg p_k \vee r_1 \vee r_2 \vee \ldots \vee r_m \vee \neg q_1 \vee \ldots \vee \neg q_n \vee s_1 \vee s_2 \vee \ldots \vee s_m,$$

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.

Zauważmy, że po wywnioskowaniu nowej klauzuli pewne wyrażenia mogą się w niej powtarzać. Jak już wiemy $A \vee A$ jest równoważne A, więc powtórzenia wyrażeń po prostu usuwamy. Na przykład, mając klauzule $p \vee q \vee r$ oraz $p \vee \neg q$ możemy zastosować regułę rezolucji i uzyskać wniosek $q \vee q \vee r$. Ponieważ q się powtarza, możemy usunąć powtórzenie i uzyskujemy $q \vee r$.

Jak wspomnieliśmy, reguła 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 zastosowaniem metody rezolucji jest wykazywanie, że pewna formuła wynika z bazy danych wiedzy, w której wiedza jest reprezentowana jako zbiór klauzul. Chcemy sprawdzić czy $\Delta \Rightarrow q$, gdzie Δ jest bazą wiedzy, a q jest formułą wykazywaną przy założeniu, że wiedza zawarta w Δ odzwierciedla interesującą nas rzeczywistość. W metodzie rezolucji zaprzeczamy implikacji $(\Delta \Rightarrow q)$. Zaprzeczenie to jest równoważne formule $(\Delta \wedge \neg q)$. Czyli $\neg q$ dokładamy do bazy wiedzy Δ (przekształcając $\neg q$ do postaci klauzulowej) i staramy się wyprowadzić klauzulę pustą.

Zauważmy, że baza Δ 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 $(p \Rightarrow q) \wedge (\neg p \Rightarrow q)$ 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:

$$(\neg\text{deszcz} \wedge \text{parasol} \Rightarrow \neg\text{zmoknę}) \wedge (\text{deszcz} \wedge \text{parasol} \Rightarrow \neg\text{zmoknę}),$$

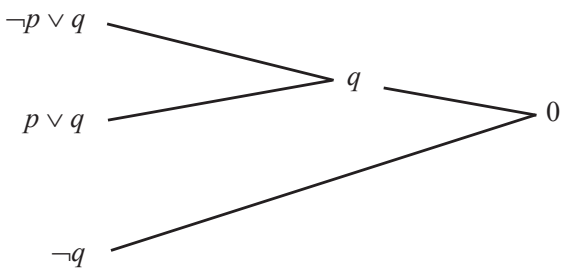
a więc wnioskuję, że skoro zawsze biorę parasol, nie zmoknę bez względu na to, czy będzie deszcz, czy nie. Może to niezbyt praktyczny wniosek, zwłaszcza w czasie upałów, ale rzeczywiście gwarantuje ochronę przed zmoknięciem.

Zasadę wnioskowania przez przypadki można zapisać w postaci klauzulowej jako:

$(\neg p \vee q)$ – pierwsze założenie w postaci klauzulowej
$(p \vee q)$ – drugie założenie w postaci klauzulowej
$\neg q$ – zaprzeczona konkluzja.

Stosując regułę rezolucji do dwóch pierwszych klauzul uzyskujemy klauzulę $(q \vee q)$, 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 1.

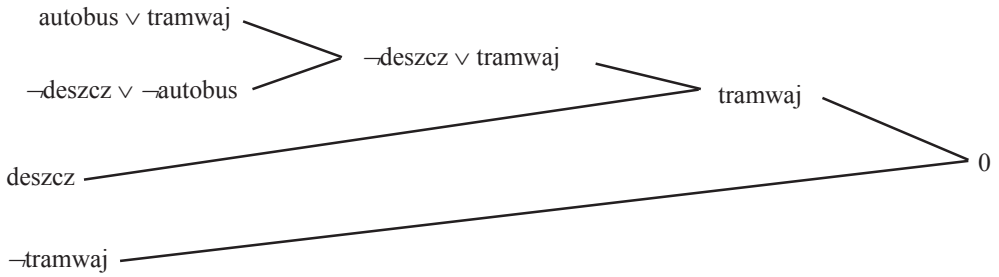


Rysunek 1. Graficzna reprezentacja wnioskowania rezolucyjnego

Zbadajmy jeszcze słuszność wcześniej rozważanego rozumowania dotyczącego wyboru pomiędzy autobusem lub tramwajem. Reprezentacja zdań (a)–(d) w logice może być następująca:

autobus ∨ tramwaj,
deszcz ⇒ ¬autobus (ta implikacja jest równoważna klauzuli ¬deszcz ∨ ¬autobus),
deszcz,
wniosek: tramwaj.

Aby zastosować metodę rezolucji, negujemy wniosek, dołączamy go do bazy wiedzy i staramy się uzyskać klauzulę pustą reprezentującą fałsz (0). Odpowiednie wnioskowanie rezolucyjne ilustruje rysunek 2.



Rysunek 2. Przykładowe wnioskowanie rezolucyjne

Rozważmy jeszcze inny przykład wnioskowania. Mamy do czynienia z sytuacją, w której robot ma wybrać kierunek ruchu: w lewo, na wprost lub w prawo. Jeśli nie pada deszcz, powinien iść w prawo. Jeśli pada deszcz, nie powinien iść w lewo, ani na wprost. Jaki kierunek robot może wybrać?

Stwórzmy najpierw odpowiednią bazę wiedzy:

- lewo ∨ wprost ∨ prawo,
- ¬deszcz ⇒ prawo,
- deszcz ⇒ ¬lewo ∧ ¬wprost.

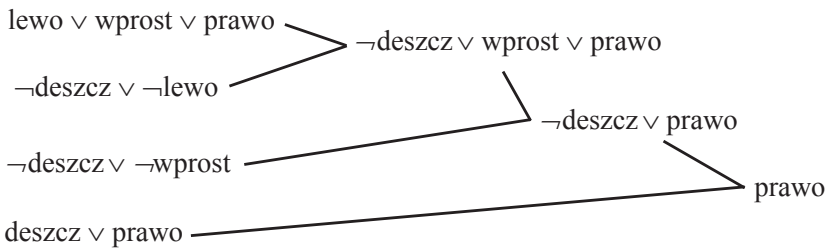
Po przekształceniu tej bazy do postaci klauzulowej uzyskamy:

- lewo ∨ wprost ∨ prawo,
- deszcz ∨ prawo,
- ¬deszcz ∨ ¬lewo,
- ¬deszcz ∨ ¬wprost.

Dwie ostatnie klauzule są uzyskane z implikacji deszcz ⇒ ¬lewo ∧ ¬wprost. Mianowicie, zgodnie z podanymi wcześniej zasadami, jest ona równoważna: ¬deszcz ∨ (¬lewo ∧ ¬wprost), czyli:

$$(\neg\text{deszcz} \vee \neg\text{lewo}) \wedge (\neg\text{deszcz} \vee \neg\text{wprost}).$$

Rysunek 3 przedstawia odpowiednie wnioskowanie rezolucyjne (dla czytelności kolejność klauzul została nieco zmieniona, co oczywiście nie wpływa na wynik).



Rysunek 3. Przykładowe wnioskowanie rezolucyjne

Właściwym wnioskiem jest więc wybór kierunku w prawo. Możemy zauważyć bowiem, że gdybyśmy rozważali wnioskowanie „baza wiedzy implikuje wniosek", to stosując metodę rezolucji dodajemy zaprzeczony wniosek do bazy wiedzy i staramy się uzyskać fałsz. Gdybyśmy dołączyli do bazy zaprzeczony wniosek, czyli „¬prawo", uzyskanie fałszu byłoby już natychmiastowe, ponieważ mamy wywnioskowany fakt „prawo".

6. Logika wyszukiwania

Możemy już stwierdzić, że logika jest we wnioskowaniu wszechobecna, gdyż fałsz, prawda, wnioskowanie, model, pojęcie, związek (relacja), reguła, czy spójniki (∧, ∨, ¬, ⇒) są podstawowymi konceptami rozważanymi w logice.

Przejdźmy teraz do wyszukiwarek internetowych. Internet jest ogromną bazą wiedzy (bez względu na to, jak oceniamy jakość tych czy innych obszarów tej wiedzy). Zasadniczą różnicą w porównaniu z tradycyjnymi bazami danych, zorganizowanymi w bardzo uporządkowany sposób, jest to, że w Internecie występuje ogromna różnorodność zasobów i brak ich jednolitej struktury.

Dla ustalenia uwagi załóżmy, że interesującymi nas obiektami z Internetu są strony WWW. Wpisując w okienko wyszukiwarki Google zestaw słów kluczowych, pytamy o te strony, na których występują wszystkie wypisane słowa kluczowe. Jest to tzw. **wyszukiwanie AND**, odpowiadające koniunkcji (w języku polskim „and" znaczy „i").

Jak już wiemy, aby koniunkcja *p* AND *q* była prawdziwa, prawdziwe muszą być oba zdania składowe: zdanie *p* i zdanie *q*. Jeśli 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 +infor-*

matyka (operator + umieszczony przed danym słowem oznacza, że musi ono wystąpić na wyszukanej stronie)[1].

Co jeszcze pojawia się w Google? Twórcy tej wyszukiwarki oferują też **wyszukiwanie OR**. W języku polskim „or" to „lub", czyli logiczny spójnik alternatywy. Przypomnijmy sobie, że 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. Ponieważ nie używamy nawiasów, potrzebna jest zasada wyjaśniająca, jak rozumieć wpisywane wyrażenia. Przy dodawaniu i mnożeniu wyrażenie x+y*z rozumiemy jako x+(y*z). Znak mnożenia ma większą siłę niż znak dodawania. Podobnie przyjmujemy w wyrażeniach zawierających spójnik OR oraz AND, przy czym rolę mnożenia odgrywa OR, zaś dodawania – AND[2]. Oznacza to, że wyrażenie

lekcja informatyka OR *logika*

wyszukiwarka Google rozumie jako

$$lekcja \wedge (informatyka \vee logika),$$

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

I mamy jeszcze operator '-', który umieszczony przed słowem oznacza, że *nie* może ono wystąpić na wyszukanej stronie. Spójnik ten odpowiada więc negacji. Wpisanie do wyszukiwarki wyrażenia *-logika* spowoduje więc wyszukanie tych stron WWW, na których nie występuje słowo *logika*.

Google posługuje się logiką, interpretując wyrażenia logiczne i wyszukując zgodnie z nimi interesujące nas zasoby.

7. Programowanie w logice

Tradycyjne języki programowania, jak C, C++ czy Java, odzwierciedlają metodologię wyrażoną tytułem klasycznej już książki Niklausa Wirtha:

Algorytmy + struktury danych = programy.

[1] Tak naprawdę Google czasem łamie tę zasadę, bowiem algorytmy porządkownia stron w połączeniu z aukcjami związanymi ze sprzedażą reklam nie zawsze dobrze odpowiadają potrzebom wyrażonym przez wyszukującego.

[2] Ta konwencja jest charakterystyczna dla Google. Tradycyjnie koniunkcja jest „silniejsza" niż alternatywa.

Główny nacisk jest tu położony na łatwość zapisywania algorytmów i mniejszą (np. w C) lub większą (w programowaniu obiektowym, jak C++, czy Java) łatwość opisu struktur danych. Programowanie polega tu na opracowaniu właściwych struktur danych oraz na wprowadzeniu algorytmu rozumianego jako dokładny opis procesu przetwarzania danych, precyzujący kolejne kroki do wykonania w danej sytuacji. Opis ten jest następnie kompilowany lub interpretowany i wykonywany przez komputer. W wielu obszarach zastosowań okazuje się, że to tradycyjne podejście może być zastąpione takim, w którym podaje się fakty i reguły, a same obliczenia pozostawia komputerowi. To podejście, zaproponowane przez Roberta Kowalskiego w roku 1972, zakłada zmianę powyższej zasady na zasadę:

algorytm = logika + sterowanie,

w której logika mówi o tym, jaki cel należy osiągnąć, zaś sterowanie o tym, jak osiągnąć oczekiwany cel. Następuje tu oddzielenie logiki od sterowania: logikę określa użytkownik, zaś sterowanie jest wyznaczane przez komputer. Pojawia się naturalne pytanie: czy taki cel można osiągnąć? Czyli – czy możliwe jest programowanie w logice?

Aby wyjaśnić zasadę wyznaczania przez komputer sterowania przyjrzyjmy się typowemu sposobowi rozumowania, w którym z posiadanej wiedzy staramy się uzyskać interesujące nas wnioski. Mamy więc pewną, być może złożoną, bazę wiedzy i reguły wnioskowania. Na przykład:

- z faktu, że Jan jest ojcem Marii i Jacka wnioskujemy, że Maria i Jacek są – być może przyrodnim – rodzeństwem,
- z faktu, że mamy zielone światło wnioskujemy, że możemy przejść przez ulicę (nadal zachowując pewną ostrożność).

Dane są więc fakty: „Jan jest ojcem Marii", „światło jest zielone" oraz reguły, które na podstawie tych faktów pozwalają wyciągnąć wnioski „Maria i Jacek są rodzeństwem", „możemy przejść przez ulicę".

Pojawia się naturalne pytanie, w jakich dziedzinach zastosowań takie podejście ma sens, a w jakich zastosowanie podejść tradycyjnych jest się lepsze. Okazuje się, że podejście tradycyjne, oparte na algorytmach i strukturach danych, jest skuteczniejsze w zastosowaniach nastawionych na obliczenia numeryczne, w których podstawowe są operacje na liczbach, lub nastawionych na złożone struktury danych. W obliczeniach związanych z wnioskowaniem i przetwarzaniem danych symbolicznych (tj. nieliczbowych) wygodniejsze okazuje się podejście regułowe, wchodzące w skład **deklaratywnej** metody programowania, w której wyznacza się cele do osiągnięcia, pozostawiając komputerowi znalezienie metody osiągnięcia tych celów. Do języków deklaratywnych zaliczamy języki funkcyjne np. Lisp, Schema, ML, języki programowania w logice, jak

Prolog, oraz wybrane języki zapytań, jak choćby SQL czy Datalog. Spośród wymienionych języków podejście regułowe realizuje Prolog i Datalog. Języki regułowe są też stosowane w dużych systemach komercyjnych, jak Oracle Business Rules, IBM ILOG czy Business Rules Framework wchodzącym w skład Microsoft BizTalk Server.

Przykład

Rozpocznijmy od prostego przykładu – chcemy wnioskować o relacjach rodzinnych. Zacznijmy najpierw od pytania, jakie informacje wystarczają do określenia związków rodzinnych. Można wybrać różne informacje bazowe, na przykład możemy zauważyć, że wystarczy informacja o tym, kto jest czyim rodzicem i o płci. Załóżymy więc, że mamy dane relacje bycia rodzicem, mężczyzną i kobietą. W języku naturalnym relacja bycia rodzicem dotyczy dwóch osób – rodzica i potomka. Relacja płci dotyczy pojedynczych osób. Możemy na przykład powiedzieć, że Jan jest rodzicem Marii, Jan jest mężczyzną i Maria jest kobietą. W logice używamy zwykle nieco innej notacji: nazwę relacji umieszczamy na początku, a następnie piszemy **argumenty** tej relacji, czyli wyrażenia określające konkretne obiekty lub zmienne. Podobnie zapisuje się relacje w językach programowania.

Relacje występujące w naszym przykładzie zapiszemy w notacji logicznej jako:

- rodzic(X, Y) – oznaczające, że osoba X jest rodzicem osoby Y,
- kobieta(X) – oznaczające, że X jest kobietą,
- mężczyzna(X) – oznaczające, że X jest mężczyzną.

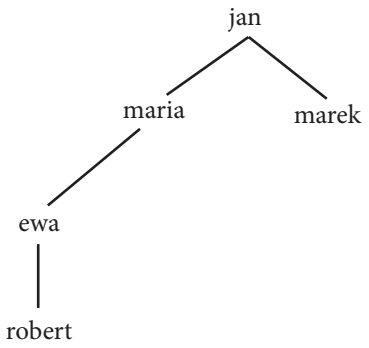
Możemy teraz opisywać fakty:

- rodzic(jan, maria), rodzic(jan, marek), rodzic(maria, ewa), rodzic(ewa, robert),
- kobieta(maria), kobieta(ewa),
- mężczyzna(jan), mężczyzna(marek), mężczyzna(robert).

Zauważmy, że imiona napisaliśmy małymi literami. To celowy zabieg – w omawianych językach regułowych stałe zapisuje się rozpoczynając nazwę małą literą. Gdy nazwę rozpoczyna wielka litera – mamy do czynienia ze **zmienną**. Jeśli napiszemy rodzic(jan, X), X oznacza zmienną, mogącą w trakcie obliczeń przyjąć konkretne wartości. Dotychczas mieliśmy do czynienia ze zmiennymi zdaniowymi, przyjmującymi wartości prawda, fałsz. Zmienne występujące powyżej mają inną rolę – reprezentują obiekty z danej dziedziny.

Pisząc fakt, taki jak rodzic(jan, maria) stwierdzamy, że ten fakt **zachodzi**, czyli jest prawdziwy w opisywanej sytuacji.

Podane wcześniej fakty definiują następujące drzewo genealogiczne:



Rysunek 4. Przykładowe drzewo genealogiczne

Możemy teraz zapisać reguły dotyczące związków rodzinnych. Zacznijmy od reguły definiującej relację bycia matką:

osoba X jest matką osoby Y jeśli X jest rodzicem Y i X jest kobietą.

Regułę tę możemy zapisać jako:

matka(X, Y) jeśli rodzic(X, Y) i kobieta(X).

Fakty wypisane po słowie „jeśli" nazywamy **przesłankami** reguły, zaś fakt występujący przed „jeśli" – jej **wnioskiem**[3]**.**

Zauważmy, że implikację piszemy teraz odwrotnie: zaczynamy od wniosku, a kończymy przesłankami. Ten styl notacji wynika z konwencji przyjętej w językach regułowych. Zauważmy też, że fakty są prostymi regułami, bowiem każdy fakt *R* można zapisać jako „*R* jeśli prawda", bowiem z prawdy możemy wywnioskować tylko fakty prawdziwe.

Używając powyższych faktów możemy wywnioskować np., że prawdziwy jest fakt:

matka(maria, ewa),

ponieważ korzystając z naszej reguły mamy:

matka(maria, ewa) jeśli rodzic(maria, ewa) i kobieta(maria),

co w świetle prawdziwości faktów rodzic(maria, ewa) oraz kobieta(maria), pozwala nam wyciągnąć omawiany wniosek.

Bycie ojcem można zapisać analogicznie:

ojciec(X, Y) jeśli rodzic(X, Y) i mężczyzna(X).

Jeśli siostrę danej osoby zdefiniujemy jako kobietę mającą wspólnego z nią rodzica, to odpowiednią regułę możemy zapisać:

[3] W literaturze wniosek nazywany jest często **nagłówkiem**, zaś zbiór przesłanek – **ciałem** reguły.

siostra(X, Y) jeśli kobieta(X) i rodzic(Z, X) i rodzic(Z, Y) i X =/= Y.

W powyższej regule Z oznacza tego samego rodzica dla X i dla Y. Oczywiście musieliśmy założyć, że X jest osobą różną od Y, ponieważ nikt nie jest swoją siostrą.

Jak zdefiniować dziadka? Możemy użyć poniższej reguły:

dziadek(X, Y) jeśli mężczyzna(X) i rodzic(X, Z) i rodzic(Z, Y).

Spróbujmy teraz zdefiniować przodka w dowolnym pokoleniu. Aby to osiągnąć, możemy użyć definicji:

(i) przodek(X, Y) jeśli rodzic(X, Y),
(ii) przodek(X, Y) jeśli rodzic(X, Z) i przodek(Z, Y).

Pierwsza z tych reguł stwierdza oczywisty fakt, iż rodzic jest przodkiem. Druga z nich stwierdza, że rodzic przodka jest także przodkiem. Użyliśmy więc **rekursji**, czyli w definicji reguły przodek, wśród przesłanek pojawia się odwołanie do przodka.

Popatrzmy jak wygląda przykładowe wnioskowanie z użyciem tych reguł. Chcemy wywnioskować, że Jan jest przodkiem Ewy. Na podstawie reguły (i) wnioskujemy, że Maria jest przodkiem Ewy, gdyż jest jej rodzicem. Stwierdzamy więc, że prawdziwy jest fakt przodek(maria, ewa). Wiemy, że rodzic(jan, maria). Zapisujemy regułę (ii) przyjmując, że X = jan, Z = maria oraz Y = ewa:

przodek(jan, ewa) jeśli rodzic(jan, maria) i przodek(maria, ewa).

Ponieważ przesłanki są prawdziwe, prawdziwy jest także wniosek. Podobnie możemy wywnioskować, że Maria jest przodkiem Roberta i znów stosując regułę (ii) mamy:

przodek(jan, robert) jeśli rodzic(jan, maria) i przodek(maria, robert),

czyli wnioskujemy również, że Jan jest przodkiem Roberta.

8. Podstawy języka Prolog

Prolog jest najbardziej znanym językiem programowania realizującym podejście regułowe. Istnieje wiele jego implementacji, w tym bardzo dobra jest darmowa implementacja SWI Prolog, dostępna pod adresem http://www.swi-prolog.org/. Inną wartą polecenia darmową implementacją jest Eclipse: http://eclipseclp.org/.

Programy zapisane w języku Prolog składają się z reguł i faktów. Fakty zapisuje się jako $R(t_1,..., t_n)$, gdzie R jest relacją n-argumentową, zaś $t_1,..., t_n$ są wyrażeniami bez zmiennych. Przykłady faktów już poznaliśmy wcześniej. Reguły mają postać:

$$R_1(t_1,..., t_{n1}) :- R_2(s_1,..., s_{n2}),..., R_k(u_1,..., u_{nk}).$$

Symbol ':-' czytamy jako „jeśli", zaś przecinki oddzielające przesłanki oznaczają spójnik „i".

Zakłada się, że argumenty relacji R_1,..., R_k są dowolnymi wyrażeniami zawierającymi stałe, zmienne i symbole funkcyjne, przy czym zmienne występujące we wniosku muszą też występować w przesłankach. Na przykład poprawna jest reguła:

$$R(\mathrm{X}, f(\mathrm{Y})) :- S(\mathrm{X}, a),\ T(\mathrm{Y}).$$

Natomiast nie jest poprawna:

$$R(\mathrm{X}, f(\mathrm{Y})) :- S(\mathrm{X}, a),\ T(\mathrm{X}).$$

ponieważ zmienna Y występuje we wniosku, a nie występuje w przesłankach.

Jeśli zbiór przesłanek jest pusty, przyjmujemy, że występuje tam prawda. Jeśli jakaś zmienna występuje w przesłankach, a nie występuje we wniosku oznacza ona istnienie pewnego obiektu. Na przykład zmienna Z w rozważanej wcześniej regule:

dziadek(X, Y):- mężczyzna(X), rodzic(X, Z), rodzic(Z, Y).

nie występuje we wniosku. Regułę tę odczytujemy więc jako:

X jest dziadkiem Y jeśli X jest mężczyzną
i istnieje osoba Z taka, że X jest rodzicem Z i Z jest rodzicem Y.

Gdy zdefiniowaliśmy już wszystkie reguły i opisaliśmy fakty, możemy zadawać pytania wyrażone przez ciąg przesłanek:

$$R(t_1,..., t_n),..., S(u_1,..., u_m).$$

Odpowiedź na tak zadane pytanie jest zdefiniowana następująco:

- jeśli w zapytaniu nie występują zmienne, to:
  - jeśli fakty występujące w zapytaniu są prawdziwe, to odpowiedzią jest „yes" (tak), czyli potwierdzenie prawdziwości zapytania, tzn. potwierdzenie, że da się ono wywnioskować z podanego zestawu faktów za pomocą zdefiniowanych reguł,
  - jeśli przynajmniej jeden fakt jest fałszywy, to odpowiedzią jest „no" (nie), czyli stwierdzenie, że zapytanie nie da się wywnioskować,
- jeśli w zapytaniu występują zmienne, to wynikiem są wszystkie krotki (ciągi) wartości, które podstawione w miejsce zmiennych powodują, że zapytanie da się wywnioskować.

Na przykład:

- zapytanie dziadek(jan, ewa), siostra(maria, marek) da odpowiedź „yes",
- zapytanie dziadek(jan, ewa), siostra(maria, ewa) da odpowiedź „no",
- zapytanie rodzic(X, Y) da w wyniku:

X = jan Y = maria
X = jan Y = marek

X = maria Y = ewa
X = ewa Y = robert

- zapytanie przodek(X, Y), kobieta(X) da w wyniku:

X = maria Y = ewa
X = ewa Y = robert
X = maria Y = robert.

Jeśli nie jesteśmy zainteresowani wartością pewnej zmiennej, np. interesują nas kobiety będące przodkami, ale nie jesteśmy zainteresowani danymi tych przodków, możemy użyć **zmiennej anonimowej** oznaczanej znakiem podkreślenia:

przodek(X, _), kobieta(X).

Wartości zmiennych anonimowych są obliczane, ale nie są przekazywane na zewnątrz.

W implementacjach języka Prolog podstawowym mechanizmem obliczeniowym jest przedstawiona wcześniej zasada rezolucji, dostosowana do radzenia sobie z regułami wychodzącymi poza rachunek zdań. Szczegóły tego dostosowania tu pominiemy. Ważne jest, że oparty o tę zasadę mechanizm obliczeniowy jest poprawny (wyniki są rzeczywiście wnioskami z faktów, uzyskanymi przez stosowanie reguł) i pełny (tzn. jeśli zapytanie jest wnioskiem, to będzie ono odpowiednio obliczone).

Język Prolog zawiera arytmetykę i listowe struktury danych. Istnieją bogate biblioteki implementujące wiele innych struktur. Przyjrzyjmy się jeszcze operacjom na listach. **Lista** to ciąg pewnych elementów. Listy zapisujemy jako $[a_1,..., a_n]$. Na przykład listą jest [1, 2, 3, *a, z*, 4]. Listy też często zapisujemy w postaci [G | Og], gdzie G jest wyróżnionym elementem nazywanym **głową** listy, zaś Og – listą pozostałych elementów, zwaną **ogonem** listy. Listę pustą zapisujemy jako [].

Rozważmy kilka przykładów przetwarzania list. Na początek sprawdźmy, czy element X występuje w danej liście:

```
jest(X, [X |]).
jest(X, [_, Y]:- jest(X, Y).
```

Pierwsza reguła stwierdza, że X występuje na liście zaczynającej się od X. Druga z nich mówi, że jeśli X jest na liście Y, to jest również na liście, której ogonem jest Y.

Aby zilustrować użycie arytmetyki, obliczmy ile jest elementów na zadanej liście:

```
ile(0, []).
ile(X, [_ | Z]:- ile(Y, Z), X = Y + 1.
```

Pierwsza z reguł mówi, że jest 0 elementów w pustej liście. Druga z nich, że w liście złożonej z jakiegokolwiek elementu początkowego '_' i ogona Z jest o jeden element więcej niż w Z.

Prolog jest stosowany w przetwarzaniu języka naturalnego, w systemach inteligentnych i wszędzie tam, gdzie wnioskowanie jest naturalnym sposobem dochodzenia do wymaganych wyników. Okazuje się, że w takich zastosowaniach kod w języku Prolog jest wielokrotnie (a czasem nawet kilkusetkrotnie) krótszy niż kody programów rozwiązujących takie same zagadnienia, napisanych w językach C, C++ czy Java.

9. Języki zapytań

Na pewno czytelnik zauważył, że opisując fakty mamy do czynienia z bazami danych. Mianowicie relacja odpowiada tabeli w relacyjnej bazie danych, zaś zbiór faktów definiuje kolejne wiersze w tej tabeli. Przykładowe fakty dotyczące bycia rodzicem można interpretować jako tabelę z dwoma kolumnami:

rodzic	**dziecko**
jan	maria
jan	marek
maria	ewa
ewa	robert

Również każdą tabelę można opisać zestawem faktów, po jednym fakcie dla opisu każdego z wierszy. Dlatego też jest naturalne, by faktami opisywać dane o elementach, zaś regułami – tabele, nieistniejące w rzeczywistości, ale obliczane, gdy zachodzi taka potrzeba. Poza zyskiem pamięciowym, można w ten sposób wyrazić zapytania niedające się zapisać w tradycyjnym standardzie SQL.

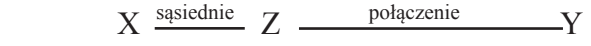
Podstawowym językiem regułowym rozważanym w kontekście baz danych jest **Datalog**. Jest on pewnym ograniczeniem języka Prolog. Mianowicie nie dopuszcza się w nim symboli funkcyjnych. Jedynymi argumentami relacji mogą więc być stałe i zmienne. Baza danych składa się z dwóch części: bazy faktów i bazy reguł. Datalog ma wiele implementacji, w tym darmowe. Bardzo dobrą edukacyjną implementacją Datalogu jest DES, dostępny pod adresem: http://www.fdi.ucm.es/profesor/fernan/des/. Zawiera on także implementację języka SQL, jest więc doskonałym narzędziem do nauczania baz danych.

Aby zilustrować tworzenie baz danych w ujęciu regułowym, rozważmy przykład, w którym interesuje nas znajdowanie połączeń kolejowych, być może z przesiadkami. Projektując taką bazę danych musimy zastanowić się, jakie informacje powinny być przechowywane w bazie danych. Naturalnym pierwszym pomysłem jest trzymanie wszystkich połączeń. Gdybyśmy jednak mieli tylko jedną linię kolejową przechodzącą kolejno przez 100 stacji, to wszystkich

połączeń byłoby 100*99 (każda stacja połączona z 99 innymi stacjami), czyli 9 900. Tymczasem wystarczy przechowywać informacje jedynie o połączeniach pomiędzy sąsiednimi stacjami, a więc informacje o nie więcej niż 99 połączeniach. Zdefiniujmy więc bazę danych zawierającą relację sąsiednie(X, Y), mówiącą o tym, że stacje X i Y są sąsiednie i połączone ze sobą. Teraz połączenia możemy zdefiniować regułami:

(iii) połączenie(X, Y):- sąsiednie(X, Y).
(iv) połączenie(X, Y):- sąsiednie(X, Z), połączenie(Z, Y).

Pierwsza z tych reguł jest oczywista. Druga z nich stwierdza, że X i Y są połączone, jeśli istnieje stacja Z sąsiednia do X (a więc połączona z X), która to stacja Z jest połączona z Y. Sytuację tę ilustruje poniższy diagram:



Zauważmy, że jeśli stacje X i Y są połączone, tzn. połączenie(X, Y) jest prawdą, przejazd między X oraz Y może wymagać zmiany pociągu.

Załóżmy, że mamy bazę faktów:

sąsiednie(warszawa, działdowo), sąsiednie(działdowo, iława),
sąsiednie(iława, tczew), sąsiednie(tczew, gdańsk).

Jak możemy teraz obliczyć relację „połączenie”?

Datalog ma dwie podstawowe metody obliczeniowe – poprzez wspomnianą przy okazji języka Prolog rezolucję oraz tzw. **metodę wstępującą**, w której zaczynając od faktów generujemy przy pomocy reguł wszystkie możliwe wnioski. Okazuje się, że metoda wstępująca zawsze się zatrzymuje i działa w czasie dopuszczalnym z praktycznego punktu widzenia (ograniczonym przez wielomian, gdy rozmiarem danych jest liczba obiektów/stałych występujących we wszystkich relacjach). Zanim sformalizujemy tę metodę przyjrzyjmy się jej działaniu na przykładzie połączeń kolejowych:

- reguła (iii) powoduje wygenerowanie faktów:
  połączenie(warszawa, działdowo), połączenie(działdowo, iława),
  połączenie(iława, tczew), połączenie(tczew, gdańsk),
- reguła (iv) powoduje dodatkowo wygenerowanie najpierw:
  połączenie(warszawa, iława), połączenie(działdowo, tczew),
  połączenie(iława, gdańsk),
  a więc połączeń stacji wymagających jednej stacji pośredniej,
- następnie reguła (iv) generuje dodatkowo:
  połączenie(warszawa, tczew), połączenie(działdowo, gdańsk),
  a więc połączeń stacji wymagających dwóch stacji pośrednich,
- w ostatniej iteracji regułą (iv) generuje fakt:

połączenie(warszawa, gdańsk),
a więc połączenie wymagające trzech stacji pośrednich.

Ogólnie algorytm obliczania relacji zdefiniowanych regułami jest następujący (ten algorytm jest poglądowy, rzeczywiste implementacje wykorzystują szereg zaawansowanych technik):

- niech A będzie zbiorem faktów
- dopóki A się zmienia:
  dodawaj do A nowe fakty, które powstają ze stosowania reguł w ten sposób, że ilekroć wszystkie przesłanki instancji danej reguły (z ustalonymi wartościami wszystkich zmiennych) są w zbiorze A, dołączamy do A również wniosek tej reguły.

Gdybyśmy chcieli rozszerzyć rozważaną bazę danych o połączenia autobusowe i możliwość przesiadek między autobusami i pociągami, wystarczy dodać relację sąsiednieA(X, Y) stwierdzającą, że X oraz Y są połączone autobusem bez przystanków pośrednich. Podane uprzednio reguły (iii) oraz (iv) wystarczy teraz uzupełnić o reguły:

połączenie(X, Y):- sąsiednieA(X, Y).
połączenie(X, Y):- sąsiednieA(X, Z), połączenie(Z, Y).

10. Uwagi o metodologii postępowania

Języki regułowe są doskonałym narzędziem modelowania złożonej rzeczywistości, a jednocześnie dostarczają mechanizmów wnioskowania. Z jednej strony mamy więc model, a z drugiej – natychmiastową możliwość jego testowania poprzez mechanizm zapytań występujący zarówno w języku Prolog, jak i regułowych językach bazodanowych.

Uporządkujmy teraz nieco metodologię stosowaną przy modelowaniu. Jak już podkreślaliśmy – skuteczność wnioskowań zależy od przyjętego modelu rzeczywistości.

Przyjrzyjmy się tej metodologii na jeszcze jednym przykładzie. Załóżmy, że chcemy utworzyć bazę danych w języku Datalog, pozwalającą opisywać ulice w danym mieście oraz interesujące miejsca, do których można tymi ulicami dotrzeć. Interesują nas skrzyżowania ulic oraz informacje, czy dana ulica jest zamknięta dla ruchu, czy przejezdna.

Zacznijmy od identyfikacji obiektów, o jakich będziemy gromadzić informacje. Z pewnością będą to ulice oraz interesujące miejsca. Ich atrybutami będą:

- w przypadku ulic - nazwa ulicy oraz czy informacja o przejezdności ulicy, opisywane jako ulica(U, P),
- w przypadku interesujących miejsc - typ miejsca (pomnik, muzeum, teatr...) oraz nazwa miejsca, opisywane jako miejsce(N, T).

Pozostały do określenia relacje. Z punktu widzenia naszego przykładu – mamy dwie relacje:

- skrzyżowanie(U1, U2), gdzie U1, U2 są nazwami ulic, stwierdzające, że istniejące ulice U1 i U2 tworzą skrzyżowanie,
- blisko(N, U), gdzie N jest nazwą miejsca, zaś U jest nazwą ulicy, stwierdzające, że miejsce N jest w pobliżu ulicy U.

Możemy teraz opisywać fakty takie jak:

ulica(kwiatowa, przejezdna), ulica(maków, zamknięta), ulica(bratków, przejezdna),
skrzyżowanie(kwiatowa, maków), skrzyżowanie(maków, bratków),
miejsce(lalka, teatr), miejsce(sienkiewicz, pomnik),
blisko(lalka, bratków), blisko(sienkiewicz, maków).

Zauważmy, że stwierdziliśmy istnienie skrzyżowania ulicy Kwiatowej z ulicą Maków, ale nie mamy informacji, że istnieje skrzyżowanie ulicy Maków z ulicą Kwiatową. Aby nie wypisywać wszystkich takich informacji możemy dodać regułę[4]:

skrzyżowanie(U1, U2):- skrzyżowanie(U2, U1).

Załóżmy, że jesteśmy zainteresowani uzyskiwaniem informacji, czy z danej ulicy możemy dotrzeć do zadanego interesującego miejsca. Aby to było możliwe, musimy mieć dodatkową relację można(U, N), stwierdzającą, że z ulicy U można dotrzeć w pobliże miejsca N. Aby zdefiniować tę relację, potrzebujemy jeszcze wiedzy, czy z danej ulicy można dojechać na podaną ulicę:

dojazd(U1, U2):- U1 = U2.
dojazd(U1, U2):- skrzyżowanie(U1, U3), ulica(U3, przejezdna),
dojazd(U3, U2).

Pierwsza reguła mówi, że z danej ulicy można dojechać na nią samą. Druga stwierdza, że można dojechać z ulicy U1 na ulicę U2, jeśli istnieje przejezdna ulica U3 mająca skrzyżowanie z U1, z której to ulicy U3 można dojechać na ulicę U2.

Relację „można" opisujemy teraz regułą (dlaczego jest poprawna?):

można(U, N):- dojazd(U, U1), blisko(N, U1).

Teraz jesteśmy gotowi do zadawania wiele interesujących pytań, np.:

- miejsce(N, teatr), blisko(N, U) – wyszukaj pary N = nazwa teatru U = nazwa ulicy, takie że N jest blisko U,
- miejsce(N, teatr), można(U, N) – wyszukaj pary N = nazwa teatru U = nazwa ulicy, takie że z ulicy U można dojechać w pobliże N.

Mamy więc mini system doradczy dla przewodnika po mieście.

[4] Ze względu na przyjęte mechanizmy obliczeniowe taka reguła zawsze działa w języku Datalog, ale nie we wszystkich implementacjach języka Prolog, którego obliczenia mogą się tu zapętlić.

Na zakończenie podajmy jeszcze dwie wskazówki, jak tworzyć reguły zawierające alternatywę wśród przesłanek oraz koniunkcję we wniosku. Otóż często chciałoby się mieć reguły postaci:

W:- *A* lub *B*.
R i *P*:- *Q*.

Ani Prolog, ani Datalog nie pozwalają na wyrażanie takich reguł bezpośrednio. Możemy jednak skorzystać z praw logiki i uzyskać reguły równoważne powyższym. Po pierwsze, następująca formuła jest zawsze prawdziwa:

[(*A* lub *B*) implikuje *W*] jest równoważne [(*A* implikuje *W*) i (*B* implikuje *W*)].

Zapisując powyższą równoważność w języku rachunku zdań uzyskamy:

$$[(A \vee B) \Rightarrow W] \Leftrightarrow [(A \Rightarrow W) \wedge (B \Rightarrow W)].$$

Możemy teraz sprawdzić prawdziwość tej równoważności używając metody tablic logicznych lub rezolucji.

Regułę „*W*:- *A* lub *B*" możemy więc zastąpić dwoma regułami zgodnymi ze składnią języka Prolog/Datalog:

W:- *A*.
W:- *B*.

Dla uzyskania drugiego rodzaju reguł skorzystamy z innego prawa logicznego:

[*Q* implikuje (*R* i *P*)] jest równoważne [(*Q* implikuje *R*) i (*Q* implikuje *P*)],

co znów można sprawdzić w rachunku zdań, ponieważ powyższą formułę można w sposób naturalny zapisać jako:

$$[Q \Rightarrow (R \wedge P)] \Leftrightarrow [(Q \Rightarrow R) \wedge (Q \Rightarrow P)].$$

Reguła „*R* i *P*:- *Q*" jest więc równoważna następującym dwóm regułom:

R:- *Q*.
P:- *Q*.

Na przykład:

rodzic(X, Y):- matka(X, Y) lub ojciec(X, Y)

zapisujemy jako reguły:

rodzic(X, Y):- matka(X, Y).
rodzic(X, Y):- ojciec(X, Y).

Natomiast:

silny(X) i zdrowy(X):- wysportowany(X)

zapisujemy jako:

silny(X):- wysportowany(X).
zdrowy(X):- wysportowany(X).

11. Podsumowanie

Na stosunkowo niewielu stronach tego rozdziału przeszliśmy w rzeczywistości bardzo długą drogę: od modelowania i wnioskowania w klasycznym rachunku zdań (o źródłach jeszcze w starożytności) po języki regułowe, dla których impulsem była metoda rezolucji opublikowana przed niecałymi pięćdziesięcioma laty. Od uzasadniania poprawności rozumowań przeszliśmy do wnioskowania na podstawie baz wiedzy na gruncie rachunku zdań po to, by pokazać mechanizmy programowania w logice i powrócić do baz wiedzy w języku bardziej zaawansowanym niż rachunek zdań.

Rachunek zdań, choć wydaje się prosty, swój obecny kształt uzyskał przed prawie stu pięćdziesięciu laty. 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 dolarów (http://www.claymath.org/millennium/P_vs_NP/), jest równoważny wykazaniu istnienia lub nieistnienia efektywnego algorytmu badania, czy dana formuła klasycznego rachunku zdań jest spełnialna[5].

Mimo swojej siły rachunek zdań 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, znajdując zastosowania w systemach autonomicznych, jak bezzałogowe helikoptery czy złożone systemy robotyki.

Jednym z ważnych rozszerzeń rachunku zdań są języki regułowe. Reguły odpowiadają implikacji, jednak mamy w nich dużo bogatszy repertuar środków wyrazu dzięki relacjom i wyrażeniom funkcyjnym. Języki regułowe są jednym z najważniejszych podejść realizujących paradygmat programowania deklaratywnego, a więc takiego, w którym opisujemy cel do obliczenia, zamiast podawania algorytmu precyzującego krok po kroku obliczenia prowadzące do tego celu. Są one stosowane w systemach sztucznej inteligencji, w tym w systemach doradczych, bazach wiedzy czy w analizie języka naturalnego. Jako narzędzie edukacyjne umożliwiają one wprowadzenie podstawowych zasad logiki wnioskowania. Są też skutecznym narzędziem nauki myślenia rekurencyjnego.

[5] Więcej o problemach milenijnych można przeczytać w rozdziale *Czy wszystko można obliczyć. Łagodne wprowadzenie do złożoności obliczeniowej.*

Świat logik jest bardzo bogaty. Logika występuje praktycznie w każdej dziedzinie życia i nauki i choć jednym z najważniejszych obszarów jej zastosowań jest informatyka, nie można zapomnieć o jej użyciu w codziennych życiowych aktywnościach, naukach ścisłych, ale i w humanistycznych. Trudno sobie bowiem wyobrazić jakiekolwiek prace w filozofii, historii czy prawie bez rygorystycznych zasad stojących za przeprowadzanymi tu rozumowaniami.

Literatura

1. Kowalski R., *Logika w rozwiązywaniu zadań*, WNT, Warszawa 1989
2. Niederliński A., *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy ECLiPSe*, Wydawnictwo PKJS, Gliwice 2012 (istnieje również darmowa wersja pierwszego wydania, udostępniona przez Autora: http://www.pwlzo.pl/download/PWLZO_zab.pdf)
3. Wirth N., *Algorytmy + struktury danych = programy*, WNT, Warszawa 2004

Prof. dr hab. Andrzej Szałas

zajmuje się logicznymi podstawami informatyki i sztucznej inteligencji, specjalizując się w nieklasycznych metodach wnioskowania, w tym w językach regułowych. Jego wcześniejsze badania dotyczyły też systemów operacyjnych, języków obiektowych oraz semantyki współbieżności. W roku 1980 ukończył Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Następnie odbył studia doktoranckie w Instytucie Matematycznym Polskiej Akademii Nauk. Stopnie doktora (1984) oraz doktora habilitowanego (1991) uzyskał na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. W roku 1999 otrzymał tytuł naukowy profesora nauk matematycznych w zakresie informatyki. Obecnie jest zatrudniony na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego oraz w Department of Computer and Information Science na Uniwersytecie w Linköpingu (Szwecja). Pracował też jako profesor lub profesor wizytujący na polskich i zagranicznych uczelniach. Opublikował 6 książek oraz ponad 100 artykułów naukowych.

andrzej.szalas@mimuw.edu.pl

Grzegorz Jakacki

Codility Ltd.

Języki – komunikacja z człowiekiem i maszynami

Często nie zdajemy sobie sprawy, że otaczają nas nie tylko języki naturalne (polski, angielski), ale także wiele języków sztucznych, którymi bezwiednie posługujemy się na co dzień w komunikacji z ludźmi i maszynami. Przyciski pilota od telewizora, liczby rzymskie, oznaczenia na mapach, a nawet zielone światło na skrzyżowaniu to elementy języków sztucznych. Szczególnie ciekawa grupa to języki programowania, czyli języki służące do precyzyjnego instruowania maszyn.

Od lat 50. XX wieku trwają prace nad językami do komunikacji z maszyną. Pierwsze języki były szorstkie i niewygodne do stosowania. Komunikacja była trudna, trzeba było wkładać dużo pracy w to, żeby się dobrze wysłowić. Łatwo było się pomylić, zostać zrozumianym opacznie. Z biegiem czasu powstawały języki coraz doskonalsze. Rozwinięto metody opisu języków, pomagające zachować precyzję i poprawność wypowiedzi. Stworzono narzędzia pomagające tłumaczyć języki wygodniejsze dla człowieka na języki bardziej pasujące do maszyn. Zaprojektowano mechanizmy językowe, dzięki którym maszyna potrafi zrozumieć i wyłapać niektóre pomyłki rozmówcy. Zbadano style komunikacji z maszyną i stworzono języki, ułatwiające spójne formułowanie wypowiedzi, a utrudniające bałaganiarstwo.

W tym rozdziale pokazujemy przykłady różnych języków sztucznych, demonstrujemy kilka prostych metod projektowania takich języków oraz opowiadamy historię rozwoju komputerów z punktu widzenia komunikacji programisty z maszyną.

1. Prolog

Wiosną 1965 roku na biurku Martina Minsky'ego, kierownika projektu Sztucznej Inteligencji na Politechnice Massachusetts (Massachusetts Institute of Technology), odezwał się bakelitowy telefon. Dzwonił Stanley Kubrick, reżyser i producent filmowy, który od ponad roku pracował nad scenariuszem do nowego filmu science-fiction dla wytwórni Metro-Goldwyn-Mayer. Kubricka interesowały perspektywy rozwoju sztucznej inteligencji, a to akurat była domena Minskiego. Nie tylko badał sztuczną inteligencję, ale też filozofował na pokrewne tematy. W filmie Kubricka komputer HAL 9000, potężna maszyna obdarzona zdolnością odczuwania emocji i komunikacji za pomocą mowy, zainstalowany jako komputer pokładowy na statku kosmicznym, popada w wewnętrzny konflikt moralny spowodowany sprzecznymi rozkazami i, by go uniknąć, po kolei morduje członków załogi. Ostatni pozostały przy życiu astronauta z trudem dostaje się do wnętrza maszyny i, nie mając innych możliwości zatrzymania komputera, rozpoczyna sukcesywny demontaż modułów pamięci. Komputer próbuje negocjować z astronautą, a czując nadciągający koniec, zaczyna się bać. Astronauta demontuje kolejne moduły pracującej maszyny, wciąż prowadząc z nią rozmowę. HAL stopniowo popada w otępienie, a na koniec mentalnej agonii głosem przechodzącym w powolny bełkot, śpiewa piosenkę *Daisy Bell*, której nauczono go na wczesnym etapie programowania. Film *Odyseja Kosmiczna 2001* miał premierę 2 kwietnia 1968 roku, a mówiący komputer HAL 9000 zawładnął masową wyobraźnią.

Komunikacja z maszyną za pomocą mowy to marzenie, które pojawia się od zarania komputerów. Póki co komputery mają problem z reagowaniem na ludzką mowę. Przez ostatnie 50 lat to raczej ludzie uczyli się mówić tak, żeby komputer rozumiał, o co im chodzi.

2. Moje pierwsze sztuczne języki

2.1. Jak babcia pokazała mi język

Pierwszego sztucznego języka do komunikacji z maszyną nauczyłem się w wieku trzech lat, gdy babcia pokazała mi sygnalizator na przejściu dla pieszych. Maszyna komunikowała się ze mną bardzo prostym językiem: czerwony ludzik, zielony ludzik, który czasami migał. Ten język ma proste zasady:

(1) czerwony i zielony ludzik nigdy nie świeciły się jednocześnie,
(2) jeśli zielony ludzik migał, to znaczyło, że zaraz pokaże się czerwony.

Ten język służy do przekazywania informacji, czy można bezpiecznie przejść przez jezdnię. Ja też komunikowałem coś sygnalizatorowi. Akurat koło domu mojej babci na słupku sygnalizatora zainstalowano małe blaszane pudełko z gumowym przyciskiem. Mogłem nacisnąć gumowy przycisk, co w języku sygnalizacji oznaczało „chcę przejść, daj mi zielone światło".

Wiele lat później odkryłem, że sygnalizatory w innych krajach (np. w USA) porozumiewają się z przechodniami podobnie, chociaż język jest nieco inny. Był tam biały idący ludzik oraz pomarańczowy napis STOP i, by zasygnalizować, że faza bezpiecznego przejścia dobiega końca, sygnalizator mrugał napisem STOP, odmiennie od polskich sygnalizatorów, posługujących się wówczas migającym zielonym sygnałem.

Moja znajomość języka sygnalizacji świetlnej pogłębiła się, kiedy zdawałem egzamin na prawo jazdy. Poznałem wtedy język sygnalizatorów ulicznych przeznaczonych dla samochodów. Ten język jest dużo bogatszy od języka komunikacji z pieszymi. Ma więcej znaków, np. światło żółte, małą zieloną strzałkę pod sygnalizatorem, sygnały kierunkowe. Znaki można łączyć, np. światło czerwone z małą zieloną strzałką oznacza „nie wolno jechać prosto przez skrzyżowanie, ale można skręcić, ustępując pierwszeństwa". Pewne kombinacje znaków mają inne znaczenie od tych samych znaków występujących osobno, np. światło czerwone z żółtym oznacza co innego niż światło czerwone i co innego niż światło żółte. Pewne kombinacje sygnałów są niepoprawne, np. jednoczesne światło czerwone i zielone.

2.2. Język liczb arabskich

Drugim sztucznym językiem, którego się nauczyłem jeszcze w przedszkolu, był dziesiętny pozycyjny system zapisu liczb, który znają wszyscy, więc nie jest czymś nadzwyczajnym[1]. Ten język ma dziesięć **symboli-cyfr** (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), które można łączyć zapisując je poziomo, jeden obok drugiego w jeden napis zwany **liczbą**. Narzuca on niewiele ograniczeń. W zasadzie tylko takie, że cyfry należy pisać jedna obok drugiej poziomo, a nie np. po skosie albo pionowo. O ile w języku sygnalizatorów ulicznych pewne kombinacje symboli są zakazane, o tyle w przypadku zapisu pozycyjnego każdy ciąg cyfr jest liczbą. Może jedynie nie należy pisać zer z lewej strony liczby (np. 0005), ale nawet jeżeli ktoś to zrobi, to wciąż można zrozumieć, o co chodzi (0005 to po prostu 5).

2.3. Język liczb rzymskich

W moim domu rodzinnym wisiał zegar z kukułką. Majstrowałem przy nim czasami, zmuszając kukułkę do nadmiarowego kukania, skutkiem czego cykl

[1] O pochodzeniu systemów liczenia jest mowa w rozdziale *Historia rachowania – ludzie, idee, maszyny*.

kukułki rozjeżdżał się z tarczą i kukułka kukała np. 12 razy o godzinie szóstej. Zegar miał tarczę, a na niej wyrażenia innego sztucznego języka. Podstaw tego języka nauczyła mnie mama, a szczegóły poznałem w pierwszej klasie podstawówki. Ten język ma siedem symboli I, V, X, L, C, D i M, które można zapisywać w ciągach, oznaczających liczby. Oznaczają one:

I = 1 V = 5 X = 10 L = 50 C = 100 D = 500 M = 1000

Kolejne godziny na naszym zegarze z kukułką oznaczone były słowami I, II, III, IIII, V, VI, VII, VIII, IX, X, XI, XII. Zegar z kukułką zawisł w moim rodzinnym domu w roku 1982, czyli MCMLXXXII.

Język liczb rzymskich ma bardziej złożone zasady tworzenia poprawnych liczb niż dziesiętny system pozycyjny. Na przykład, w liczbie MCMLXXXII mamy M = 1000, CM = 900, LXXX = 80 i II = 2, co razem daje 1982. „Cyfry" w tym systemie (np. L = 50) można modyfikować, dopisując z prawej lub lewej strony symbole oznaczające mniejsze wartości. Dopisanie z prawej powiększa wartość, a dopisanie z lewej – zmniejsza[2]. Na przykład LX to „pięćdziesiąt zwiększone o dziesięć", czyli 60, LXXX to „pięćdziesiąt trzy razy zwiększone o 10", czyli 80, CM to „tysiąc zmniejszone o sto", czyli 900. Zawsze mniejsza liczba modyfikuje większą, a więc CM to „tysiąc zmniejszone o sto", a nie „sto zwiększone o tysiąc".

Reguły języka liczb rzymskich powodują powstawanie niejednoznaczności. Po pierwsze tę samą liczbę można wyrazić na wiele sposobów, np. 4 to IIII, ale również IV. Po drugie trudno powiedzieć, czy MXXL to M XXL, MX XL, czy MXX L (czyli odpowiednio 1030, 1050 czy 1070). Takie niejednoznaczności istnieją w wielu językach. Niejednoznaczność typu IIII/IV pojawia się nawet w dziesiętnym systemie pozycyjnym, np. 0005 i 5. Od takich niejednoznaczności roi się też w językach naturalnych, np. w języku polskim „auto" i „samochód" oznaczają to samo. Niejednoznaczność w interpretacji słowa MXXL wydaje się jednak poważniejszym problemem.

Niejednoznaczność w języku polskim ilustruje historia o informatyku wysłanym na zakupy:
– Kup serdelki, a jeśli będą jajka, to kup dwa.
– OK.
Wraca po chwili.
– Kupiłem dwa serdelki.
– A jajka?
– Były.

[2] Taki system liczbowy nazywa się **addytwnym**.

Język polski jest wieloznaczny i dopuszcza dwie różne interpretacje: „Kup serdelki. Jeśli będą jajka, to kup dwa serdelki" oraz „Kup serdelki. Jeśli będą jajka, to kup dwa jajka".

W języku liczb rzymskich istnieją dodatkowe reguły, które zapobiegają niejednoznacznościom w rodzaju MXXL, na przykład:

(1) liczba powinna zostać tak zapisana, aby osobno były zapisane tysiące, osobno setki, osobno dziesiątki i osobno jedności,
(2) każdy z symboli I, X, C i M może wystąpić co najwyżej trzy razy pod rząd (co przesądza na rzecz IV, a przeciwko IIII),
(3) każdy z symboli D, L i V może wystąpić co najwyżej raz pod rząd,
(4) I można odjąć wyłącznie od L lub X,
(5) X można odjąć wyłącznie od L lub C,
(6) C można odjąć wyłącznie od D lub M,
(7) nie wolno odejmować V, L ani D,
(8) można odjąć co najwyżej jeden symbol.

Interpretacja liczby MXX L jako 1020 + 50 = 1070 narusza regułę (1) – liczbę 1070 należy zapisać jako MLXX. Interpretacja liczby MX XL jako 1010 + 40 = 1050 narusza regułę (1) – liczbę 1050 należy zapisać jako ML. Interpretacja liczby M XXL jako 1000 + 30 = 1030 narusza regułę (8) – liczbę 1030 należy zapisać jako MXXX. Możliwe są jeszcze inne interpretacje zapisu MXXL, np. M X XL, czyli 1000 + 10 + 40, ale jest on też zabroniony przez powyższe reguły.

Język liczb rzymskich ma proste reguły dotyczące tworzenia nowych wyrażeń (dopisywanie z lewej i z prawej), ale ma złożone reguły dotyczące tego, które z wyrażeń są poprawne. W tym języku pojawia się również zjawisko kontekstowości, np. znak X może mieć różne znaczenie, w zależności od tego, gdzie występuje i co jest dookoła niego. Może oznaczać 10, ale też może oznaczać „dodaj 10 od budowanej liczby" albo „odejmij 10 od budowanej liczby". Choć liczby rzymskie znałem od pierwszej klasy, w praktyce po raz pierwszy zetknąłem się z kontekstowością nieco później.

2.4. Konwersacje z telewizorem

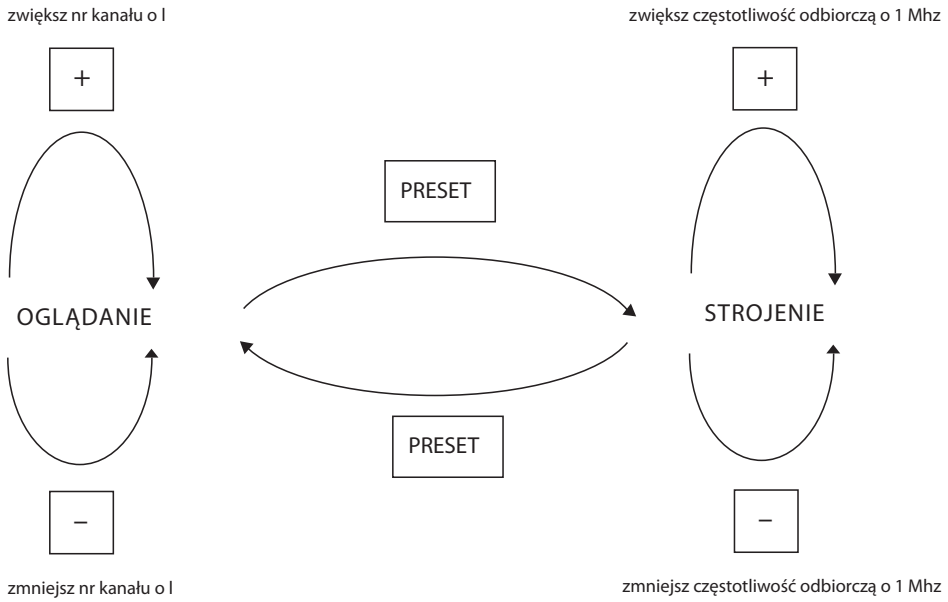
Kiedy miałem 12 lat w naszym domu pojawił się pierwszy japoński telewizor marki Otake. Z pilotem! Poczuliśmy powiew wielkiego świata, ale skokiem technologicznym okazało się dopiero strojenia kanałów. Poprzednik Otake, radziecki Elektron, miał sześć przycisków służących do wyboru kanałów. Naciśnięcie przycisku oznaczało wybór kanału. To i tak było więcej niż potrzeba, bo Telewizja Polska nadawała wówczas tylko dwa programy. Obok każdego przycisku było kółeczko do strojenia częstotliwości odbiorczej dla danego kanału.

Obrót w lewo oznaczał mniejszą częstotliwość, a obrót w prawo – większą. To było proste i intuicyjne.

Telewizor Otake komunikował się jednak za pomocą innego języka. Ukryty pod klapką panel zawierał kilka przycisków: dwa przyciski oznaczone + i –, przycisk PRESET oraz przycisk SHIFT. Wszystkie służyły do strojenia, ale ich znaczenie zmieniało się w zależności od tego, co naciśnięto wcześniej. To była ta kontekstowość. Telewizor miał wyświetlacz pokazujący numer kanału z zakresu 00-99. Naciśnięcie i przytrzymanie przycisku PRESET przez trzy sekundy powodowało wejście telewizora w tryb strojenia. Numer kanału zaczynał migać, natomiast przyciski + i – zmieniały swoje znaczenie. Zamiast przełączać kanał, powodowały zmniejszenie lub zwiększenie częstotliwości odbiorczej danego kanału. Naciśnięcie przycisku PRESET powodowało powrót do normalnego trybu. Dodatkowo przytrzymanie przycisku SHIFT powodowało modyfikację funkcji przycisków + i –. W normalnym trybie pracy zaczynały przełączać numer kanału o dziesięć, natomiast w trybie strojenia zmieniały częstotliwość odbiorczą o większy skok. Z kolei przycisk SHIFT nie miał żadnej samodzielnej funkcji, służył tylko do modyfikacji znaczenia innych przycisków. Taki modyfikator występuje nie tylko w języku telewizora Otake. Mała zielona strzałka w prawo na sygnalizatorze ulicznym jest takim modyfikatorem. W języku zapisu liczb całkowitych takim modyfikatorem jest np. znak minus stawiany przed liczbą. Sam nie ma znaczenia, natomiast modyfikuje znaczenie stojących za nim symboli. W języku polskim podobną funkcję mają np. przedrostki takie jak niby-, anty-, vice-.

Nastroiłem dwa kanały na program pierwszy i drugi TVP. Telewizor wciąż mnie intrygował, więc nastawiłem te same programy na kolejnych kanałach – program pierwszy na parzystych, program drugi na nieparzystych. To nie zaspokoiło mojej ciekawości, więc przeczytałem instrukcję. Zauważyłem, że instrukcja nie precyzuje, co się zdarzy, jeśli jednocześnie nacisnę + i –. Postanowiłem to sprawdzić empirycznie, ale o moich eksperymentach dowiedzieli się rodzice i dostałem szlaban nie tylko na zabawę telewizorem, ale i na telewizję. Na szczęście wkrótce nasz sąsiad nabył taki sam telewizor i wezwał mnie jako specjalistę od komunikacji z maszyną. Mogłem więc kontynuować eksperymenty w języku telewizora Otake. Jednoczesne naciśnięcie + i – powodowało pojawianie się na wyświetlaczu dziwnych znaków, wcale nieprzypominających cyfr. Doszedłem do wniosku, że tego typu kombinacje powinny być w języku telewizora Otake zabronione. Akurat tak się złożyło, że kolejni znajomi kupowali japońskie telewizory, miałem więc jeszcze kilka okazji komunikowania się z odbiornikami marek Otake, Hitachi, Fujitsu i JVC. Na języki tych telewizorów składały się naciśnięcia przycisków. Te naciśnięcia można było łączyć na dwa sposoby: albo naciskać przyciski jeden po drugim, albo jednocześnie (np. SHIFT równocześnie z +). Pewne kombinacje były dopuszczalne, pewne nie,

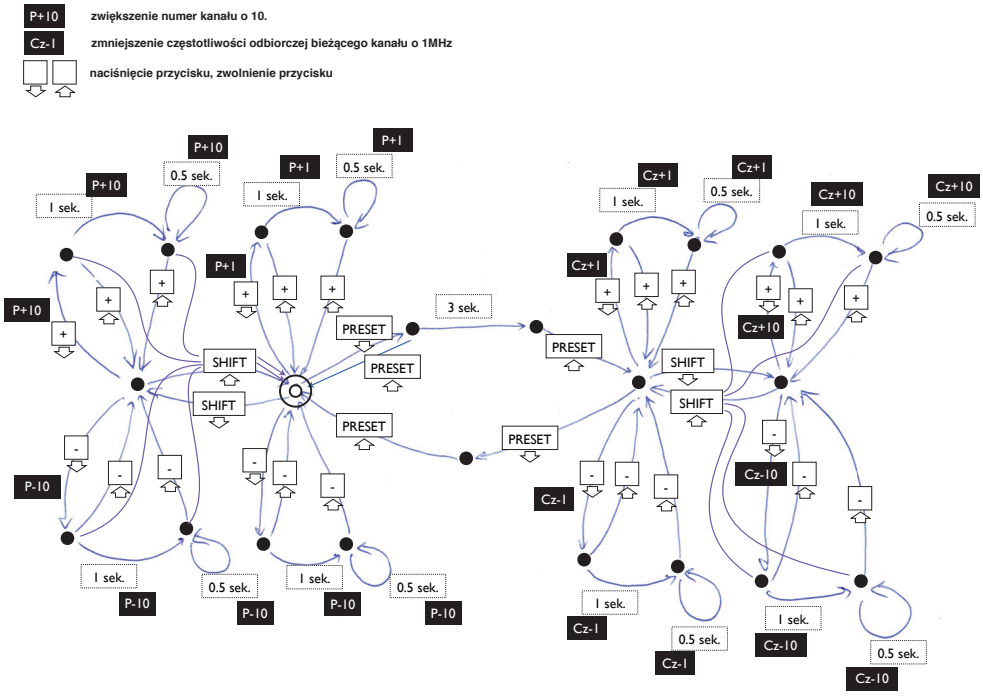
np. jednoczesne naciskanie + oraz – wyraźnie nie wychodziło na zdrowie sterownikowi telewizora Otake. Z nudów zacząłem rysować schematy opisujące języki telewizorów, patrz. rys. 1.



Rysunek 1. Opis przejść pomiędzy stanami telewizora Otake, wersja pierwsza

Nie wiedziałem wtedy jeszcze, że badam języki formalne, a te rysunki to schematy tzw. automatów skończonych opisujących języki. Wiedziałem natomiast, że mój rysunek jest niewystarczający. Nie potrafiłem znaleźć sposobu, by umieścić na nim SHIFT, albo żeby przekazać informację, że PRESET trzeba przytrzymać przez 3 sekundy, a + wystarczy nacisnąć bardzo krótko. Telewizory miały też taką dodatkową funkcję, że gdy przycisk + lub – przytrzymało się przez dłuższą chwilę, to jego działanie zaczynało się powtarzać, np. telewizor przełączał kolejne programy lub przeglądał coraz większe częstotliwości odbioru.

Wpadłem wtedy na pomysł, żeby spojrzeć na język telewizora Otake z innej perspektywy. Początkowo przyjąłem, że podstawowym jego elementem jest wciśnięcie przycisku. Do opisania pewnych cech języka to nie wystarczało. Naciśnięcie przycisku SHIFT było zazwyczaj rozciągnięte w czasie, od naciśnięcia przycisku do jego zwolnienia mogło upłynąć wiele sekund. Postanowiłem spojrzeć na ten język tak, jakby naciśnięcia i zwolnienia przycisków były odrębnymi zdarzeniami. Naciśnięcie oznaczyłem strzałką w dół, zwolnienie strzałką w górę, patrz rys. 2.



Rysunek 2. Opis przejść pomiędzy stanami telewizora Otake, wersja druga

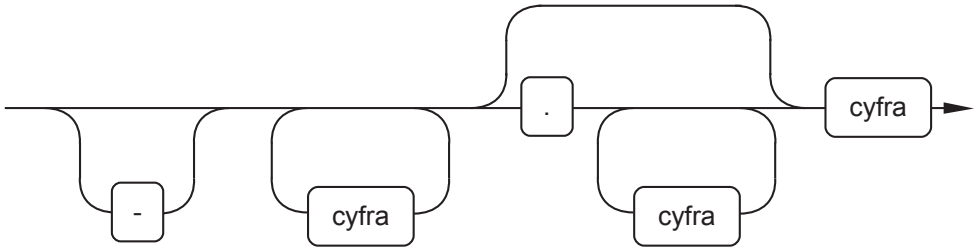
W trakcie rysowania stwierdziłem, że oprócz naciśnięcia i zwolnienia przycisku jest jeszcze jeden podstawowy element języka: upływ czasu. Jeżeli nacisnąłem PRESET i przytrzymałem go, to po upływie trzech sekund coś się działo (telewizor przełączał się w tryb strojenia), nawet jeśli nie zwolniłem ani nie nacisnąłem żadnego przycisku.

Z diagramami opisującymi języki spotkałem się ponownie parę lat później. W szkole średniej sięgnąłem po książkę *Algorytmy + Struktury Danych = Programy* Niklausa Wirtha [4]. Niewiele z niej rozumiałem, ale bardzo podobał mi się Dodatek B, zawierający całą masę diagramów opisujących różne fragmenty **języka Pascal**, m.in. diagram podobny do pokazanego na rysunku 3[3]. Jest to diagram opisujący język zapisu liczb. Dopuszczalnymi wyrażeniami tego języka są np. **5, –64, .01, –3.14**.

Dopiero po latach skonstatowałem, że rysunki w Dodatku B wspomnianej książki, a także diagramy opisujące sterownik telewizora Otake, to opisy języków.

[3] W istocie połączyłem na tym jednym diagramie dwa diagramy występujące we wspomnianej książce, ale nie ma to tutaj większego znaczenia.

Te opisy są sformułowane w podobny sposób. Składają się z prostych symboli: kropek i etykietowanych strzałek. Te elementy można łączyć według ustalonych reguł (np. strzałka musi zaczynać i kończyć się na kropce). Tak narysowany diagram niesie pewną treść – opis zachowania sterownika. Zaciekawiło mnie, że te kropki i etykietowane strzałki to też język. Za pomocą jednego sztucznego języka opisywałem inne.



Rysunek 3. Diagram opisujący popularną notację liczb wymiernych

3. Opisywanie języków

Człowiek zajmuje się badaniami języków naturalnych i sztucznych od tysiącleci. Można to robić na wiele różnych sposobów, w tym rozdziale skupiamy się na podejściu, w którym wyróżnia się:

- podstawowe symbole języka, czyli **leksykę**,
- zasady łączenia tych symboli w wyrażenia, czyli **gramatykę**,
- znaczenie wyrażeń, czyli **semantykę**.

Tabela 1 zawiera opisy leksyki, gramatyki i semantyki przykładowych języków.

Tabela 1.
Opisy przykładowych języków pojawiających się w tym rozdziale

Język	Leksyka	Gramatyka	Semantyka
Sygnalizator uliczny	czerwony ludzik, zielony ludzik	czerwony i zielony nie występują razem	zielony – droga wolna, zielony migający – zaraz będzie czerwony, czerwony – stój
Sterownik telewizora	kropki i etykietowane strzałki	strzałka musi zaczynać się i kończyć w kropce	etykiety na strzałkach oznaczają przyciski telewizora, kropki oznaczają stany telewizora; naciśnięcie przycisku odpowiada przejściu po strzałce

Tabela 1 (cd.)

Język	Leksyka	Gramatyka	Semantyka
Liczby arabskie	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	ciągi symboli (w linii)	wartość liczby uzyskujemy mnożąc wartość cyfry przez 10 do potęgi *k*, gdzie *k* oznacza numer pozycji licząc od prawej i zaczynając od zera
Liczby rzymskie	I, V, X, L, C, D, M	ciągi symboli (w linii), każdy z symboli co najwyżej 3 razy pod rząd, od V i X można odejmować wyłącznie I...	Przyjmij wartość zero i powtarzaj następującą operację: odetnij najdłuższy możliwy z poniższych przedrostków i do wartości dodaj odpowiadającą mu liczbę: I → 1 IL → 49 IX → 9 X → 10 XL → 40 XC → 90 C → 100 CD → 400 CM → 900 M → 1000 *Uwaga: ten sposób interpretacji wartości zakłada, że liczba jest poprawnie zapisana.*
Proste wyrażenia algebraiczne	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, *, (,)	gramatyka jest opisana na rysunku 11	Zinterpretuj wyrażenie według gramatyki opisanej na rysunku 11 i wykonaj działania arytmetyczne w kolejności wyznaczonej przez rozkład gramatyczny.
Język maszynowy	Liczby naturalne	ciągi liczb naturalnych, zapisanych według specyfikacji procesora	powodują modyfikacje pamięci przez kopiowanie wartości, operacje arytmetyczne i skoki *Semantykę szczegółowo opisuje dokumentacja techniczna procesora.*
Język mnemoniczny	Rozkazy: MOV, SUB, ADD, JMP, CMP...	ciągi rozkazów z jednym lub dwoma argumentami	rozkazy należy przetłumaczyć na kod maszynowy według opisu języka mnemonicznego i interpretować jako kod procesora

Tabela 1 (cd.)

Język	Leksyka	Gramatyka	Semantyka
Język C	liczby, słowa, znaki przestankowe, operatory +, – ..., nawiasy	gramatyka opisana w postaci klauzul podobnych do BNF w dokumencie standaryzacyjnym ISO/IEC JTC1/SC22/WG14	semantyka opisana za pomocą prozy technicznej w dokumencie standaryzacyjnym ISO/IEC JTC1/SC22/WG14

Pierwsze poważne i udane podejście do precyzyjnego opisu gramatyki języka wykonał hinduski lingwista Pānini (पाणिनि) w VI wieku p.n.e. Opracował on precyzyjny system opisu gramatyki i sformułował w nim 3959 reguł opisujących gramatykę języka Sanskryt. Na następne równie udane podejście trzeba było czekać dosyć długo. Powszechnie uważa się, że dopiero prace Noama Chomskiego, Johna Backusa oraz Petera Naura na przełomie lat 50. i 60. XX wieku posunęły tę dziedzinę do przodu. W roku 1959 John Backus, projektując dla firmy IBM język IAL (znany obecnie pod nazwą Algol 58), zaproponował ścisły sposób opisu gramatyki pewnego sztucznego języka. Ale o tym za chwilę.

Komputer HAL 9000 z *Odysei Kosmicznej 2001* porozumiewał się z astronautami po angielsku. Była to pociągająca wizja, daleka jednak od rzeczywistości lat 60. Język komunikacji człowieka z maszyną był w tamtych czasach wyjątkowo surowy i zdecydowanie był to język wygodny dla maszyny, ale niekoniecznie dla człowieka.

Może trudno w to uwierzyć, ale od końca lat 40. komputery w zasadzie buduje się podobnie. Komputer składa się z pamięci oraz procesora. Pamięć to przechowalnia liczb. Komórki pamięci, ponumerowane kolejnymi liczbami całkowitymi, mogą przechowywać liczby całkowite z niewielkiego zakresu (współcześnie zazwyczaj 0-255). Procesor pobiera wartości z kolejnych komórek pamięci i interpretuje je jako rozkazy powodujące zmianę wartości pewnych komórek pamięci lub wysyłanie sygnałów elektrycznych na wyprowadzenia procesora. To wszystko. Część komórek pamięci zawiera wartości przeznaczone do sterowania zachowaniem procesora (tzw. **program**), część zawiera wartości, na których procesor wykonuje operacje (tzw. dane). Na przykład, współczesny procesor firmy Intel zinterpretuje ciąg liczb 49, 237, 103, 138, 69, 0, 103, 2, 69, 1, 103, 136, 69, 2 jako program powodujący dodanie wartości z komórki pamięci o adresie 0 do komórki pamięci o adresie 1 i zapisanie wyniku w komórce pamięci o adresie 2.

Liczby w pamięci składające się na program dla procesora można traktować jak wyrażenia języka. Jest to bardzo surowy język komunikacji z maszyną. Stąd też nazwa – **język maszynowy**. Symbolami języka maszynowego są liczby natu-

ralne zapisane w pamięci. Znaczenie tych liczb opisuje dokumentacja techniczna procesora.

W czasach, gdy powstawał film o mówiącym komputerze HAL 9000, komunikacja z maszynami odbywała się wciąż na tym najniższym poziomie. Programiści wprowadzali do komputerów kod maszynowy za pomocą specjalnych kart dziurkowanych, a w starszych modelach po prostu za pomocą przełączników i kabli elektrycznych. Był to proces żmudny i podatny na błędy. Ludzki mózg, przyzwyczajony do języków, w których występują czasowniki i rzeczowniki, niekoniecznie płynnie porusza się w języku maszynowym, w którym występują tylko liczby. O ile wygodniej człowiekowi posługiwać się frazą np. „wyzeruj rejestr EBP" zamiast „49, 237". Wszystkie kody języka maszynowego można ponazywać takimi mnemonicznymi nazwami, tworząc język podobny do języka maszynowego, jednak o wiele bardziej czytelny. Na przykład, programy dodawania dwóch liczb w obu językach i w języku wyrażeń algebraicznych mają postać:

Kod maszynowy:	Kod mnemoniczny (język procesora i686)	Język wyrażeń algebraicznych
49, 237	XOR %EBP, %EBP	
103, 138, 69, 0	MOV 0(%EBP), %AL	x + y
103, 2, 69, 1	ADD 1(%EBP), %AL	
103, 136, 69, 2	MOV %AL, 2(%EBP)	

Rysunek 4. Programy, które służą do obliczania wartości x + y

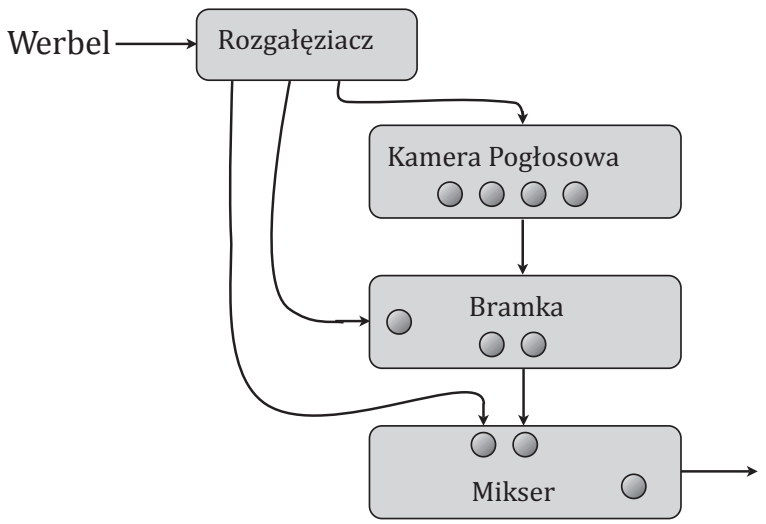
Dźwięk perkusji w utworze *In The Air Tonight* Philla Collinsa jest tak niesamowity, że kopiowali go wszyscy muzycy rockowi przez całe lata 80. XX wieku. Dźwięk werbla jest zaskakująco bogaty, jednocześnie krótki i spójny. Ten niezwykły efekt dźwiękowy zaprojektował Hugh Padgham, pracujący z Collinsem nad płytą Face Value. Konfiguracja przetworników sygnału wykorzystanych do stworzenia tego efektu jest pokazana na rysunku 5. Sygnał z mikrofonu nagrywającego perkusję rozdziela się na trzy linie. Jedna z nich jest podłączona do wejścia kamery pogłosowej. Kamera dodaje do sygnału pogłos, sprawiający wrażenie, że perkusista gra w wielkiej sali, w której dźwięk odbija się od ścian. Pogłos bardzo wzbogaca dźwięk, sprawia jednak, że krótkie, zdecydowane uderzenia (np. uderzenia werbla) stają się rozlazłe i rozwleczone w czasie – jeszcze długo po wybrzmieniu oryginalnego dźwięku słychać cichnące pogłosy i echa. Bramka przepuszcza sygnał tylko wtedy, kiedy poziom głośności sygnału na wejściu sterującym jest odpowiednio wysoki. Podanie oryginalnego sygnału na wejście steru-

jące powoduje, że dźwięk wzbogacony o pogłosy jest ucinany zaraz po wybrzmieniu oryginalnego, krótkiego uderzenia werbla. Najlepsze efekty uzyskuje się miksując tak opracowany sygnał z odrobiną pierwotnego sygnału z mikrofonu. Całość należy dostroić „na ucho", np. dobierając odpowiednie opóźnienia w kamerze pogłosowej, czas otwarcia bramki po wybrzmieniu sygnału sterującego, czy też poziom sygnału sterującego, poniżej którego bramka się zamyka.

W podobny sposób – przez strojenie i łączenie modułów kablami – programowało się pierwszy komputer elektroniczny – ENIAC. Program zapisywano najpierw na papierze, po czym zespół programistek mozolnie przenosił program na maszynę, wpinając kable i przełączając przełączniki. Dane do przetwarzania wprowadzano za pomocą czytnika kart perforowanych, służącego do wczytywania ciągów liczb wydziurkowanych na tekturowych kartach. Wyniki były dziurkowane przez maszynę na takich samych, czystych kartach.

Programowanie maszyny ENIAC było uciążliwe i często trwało tygodniami. Był to jeden z problemów, który próbowano rozwiązać w powstającym projekcie nowego komputera EDVAC. Udział w tych pracach brał John von Neumann, matematyk z Uniwersytetu Princeton, zaangażowany do Projektu Manhattan, dotyczącego broni masowego rażenia. Von Neumann korzystał z maszyny ENIAC do wykonywania obliczeń dotyczących bomby wodorowej. W notatkach z czerwca 1945 roku proponował konstrukcję, polegającą na wprowadzaniu do maszyny programu w tej samej postaci, co dane, tj. za pomocą ciągu liczb zapisanego na kartach perforowanych. Liczby miały oznaczać kolejne rozkazy dla maszyny, a maszyna miała te rozkazy po kolei wykonywać. Było to podejście zupełnie odmienne, ograniczało m.in. możliwość równoległego wykonywania operacji (w każdym momencie maszyna miała wykonywać tylko jedną operację z listy, podczas gdy w komputerze ENIAC możliwe było np. podłączenie równoległe dwóch modułów sumujących liczby jednocześnie), jednakże rozwiązywało o problem kłopotliwej i czasochłonnej rekonfiguracji kabli i przełączników przy każdorazowej zmianie programu. Pomysł został zrealizowany w roku 1948 po modyfikacji komputera ENIAC, jeszcze przed ukończeniem komputera EDVAC. Początkowy ciąg liczb wczytanych z kart perforowanych był interpretowany jako lista rozkazów, czyli program, a reszta stanowiła dane dla programu.

Przy takim podejściu przetwarzanie danych trwało sześciokrotnie dłużej, natomiast maszynę dawało się przeprogramować w ciągu jednego dnia, zamiast kilku tygodni – wystarczyło wczytać nowy zestaw kart perforowanych. Dodatkowo okazało się, że w przypadku większości obliczeń o wiele więcej czasu schodzi na wczytanie danych z kart perforowanych, niż na wykonanie samych obliczeń, więc spowolnienie nie miało znaczenia. Olbrzymia większość współczesnych komputerów, w tym komputery osobiste IBM PC, jest zbudowana według architektury von Neumanna – programy, tak jak dane, są przechowywane i przetwarzane w postaci ciągów liczb (lub symboli). Nie powinno więc nas dziwić, że programy są przechowywane na dysku, w tym samym miejscu, gdzie trzymamy zdjęcia, piosenki lub film.



Rysunek 5. Połączenia realizujące efekt dźwiękowy Gated Reverb

4. Nie możesz zrozumieć, czego nie możesz nazwać

Język mnemoniczny jest łatwiejszy dla człowieka, ponieważ przypomina trochę język naturalny. Każdy wiersz programu (patrz rys. 4) to fraza rozkazująca (polecająca). MOV to czasownik, po którym następują dwa dopełnienia. Oznacza tyle, co „skopiuj z X do Y". %EBP to rzeczownik, oznacza pomocniczą komórkę pamięci umieszczoną bezpośrednio w procesorze. Podobnie %AL. 1(X) to wyrażenie oznaczające „jedna komórka dalej niż". 1 odpowiada tu przydawce w języku polskim. Wyrażenie 1(%EBP) oznacza „jedna komórka pamięci za komórką o numerze przechowywanym w %EBP".

Język mnemoniczny łatwo przetłumaczyć na kod maszynowy, w zasadzie wystarczy słownik kodów mnemonicznych (np. XOR → 49) i parę prostych zasad dotyczących interpretacji wyrażeń takich jak X(Y), np. 1(%EBP). Tę pracę programiści wykonywali kiedyś ręcznie. Program napisany na papierze w języku mnemonicznym był ręcznie tłumaczony na kod maszynowy i wprowadzany do komputera. Zadanie to przerzucono na komputery w latach 50. XX wieku. Był to duży postęp, ale otchłań pomiędzy językiem mnemonicznym a płynną angielszczyzną komputera HAL 9000 wciąż była olbrzymia. W szczególności programiści musieli bardzo pilnować, co program przechowuje w których komórkach pamięci. Przypadkowe zapisanie danej do komórki przechowującej program mogło spowodować wytworzenie błędnego kodu, skutkującym nieprzewidzianym wykonaniem reszty programu.

Programiści piszący programy w języku maszynowym musieli nieźle się napocić, żeby zaprogramować nawet proste dodawanie. Nic dziwnego, że tęsknili za dobrze nam znanym językiem wyrażeń algebraicznych. Uczymy się go w szkole, składają się na niego liczby, litery, znaki działań (+, –, *, kreska ułamkowa), nawiasy. Zapis wyrażeń w takim języku ma wadę, gdyż na przykład dzielenia nie można zapisać w jednym wierszu. Wybrnięto z tego wprowadzając specjalny znak dzielenia /, a także umożliwiając zapisywanie potęgowania w jednym wierszu za pomocą znaku ^, zatem wyrażenie 6^10 oznacza tyle, co 6^{10}. W takiej konwencji na język wyrażeń składają się liczby, litery, znaki działań (+, –, *, /, ^), nawiasy, a zasady łączenia symboli są następujące:

(1) dwa wyrażenia można zestawić umieszczając obok siebie i łącząc je znakiem działania,
(2) wyrażenie można otoczyć parą nawiasów i otrzymać nowe wyrażenie.

Wystarczy spojrzeć na rysunek 4, żeby uświadomić sobie, o ile wygodniejszy jest język wyrażeń algebraicznych od języka mnemonicznego. Pierwszą osobą, którą zrobiła z tym problemem coś konkretnego był niemiecki inżynier Konrad Zuse. Zaprojektował on i zbudował podczas II wojny jeden z pierwszych komputerów na świecie. Pod koniec wojny salwował się ucieczką z bombardowanego Berlina i oderwany od pracy przy budowie komputera zaczął zastanawiać się nad tym, co trapiło go od pewnego czasu: kod maszynowy nie jest najwygodniejszym językiem programowania. Przemyślenia zaowocowały projektem języka Plankalkül, który Zuse opublikował parę lat po wojnie. Plankalkül wyprzedził swoją epokę i mimo wielu przełomowych rozwiązań, w szczególności możliwości zapisywania programów za pomocą składni przypominającej wyrażenia algebraiczne, nie doczekał się popularyzacji aż do lat 70. XX wieku.

Konrad Zuse, niemiecki inżynier budownictwa, przed wojną trafił do firmy Henschel, producenta lokomotyw i samolotów. Monotonia wykonywanych codziennie ręcznych obliczeń matematycznych skłoniła go do skonstruowania programowalnego mechanicznego kalkulatora V1, który służył do wykonywania obliczeń na liczbach wymiernych; instrukcje były wczytywane z dziurkowanej taśmy filmowej. Gdy we wrześniu 1939 roku samoloty Henschel Hs 123 bombardują Warszawę, Zuse ma 29 lat, a na koncie dwa patenty dotyczące konstrukcji maszyn liczących. Powołany do wojska przekonuje przełożonych, że konstruowane przez niego maszyny przysłużą się machinie wojennej. Tak powstaje kalkulator programowalny V2, ulepszona wersja V1, w której Zuse zastąpił elementy mechaniczne elektrycznymi przekaźnikami telefonicznymi. Gdy w czerwcu 1941 roku Niemcy atakują Związek Radziecki, a Hitler instaluje sztab w Wilczym Szańcu pod Kętrzynem, Zuse kończy prace nad maszyną V3, pierwszym

działającym komputerem elektromechanicznym. Komputer V3 pracuje dla Niemieckiego Instytutu Lotnictwa, gdzie jest używany do obliczeń związanych z aerodynamiką bomb szybujących. Zuse stara się o finansowanie maszyny V4, ulepszonej wersji V3, ale niemiecka administracja mu odmawia. Wiara w rychłe zwycięstwo Niemiec jest tak wielka, że urzędnicy nie widzą potrzeby budowania kolejnej maszyny. Pod koniec wojny Zuse ucieka z Berlina, a bomby aliantów niszczą jego komputery. Na przymusowych wakacjach w południowych Niemczech wymyśla Plankalkül, system zapisu rozkazów dla maszyn liczących, uważany za pierwszy zaawansowany język programowania. Na skutek izolacji spowodowanej przez wojnę do swoich odkryć dochodzi praktycznie sam, nie będąc świadomym prac Amerykanów i Brytyjczyków. Zuse po wojnie powrócił do konstruowania maszyn liczących. By uniknąć skojarzeń z niemieckimi pociskami rakietowymi V1 i V2, przemianował swoje konstrukcje na Z1, Z2, Z3. Komputer Z4 został zamówiony przez politechnikę w Zurychu i dostarczony przez Zusego w czerwcu 1950 roku. Język Plankalkül wyprzedził swoją epokę. Jego opis opublikowany pod koniec lat 40. XX wieku przeszedł bez echa. Plankalkül został odkryty ponownie w latach 70. i okazał się niezmiernie ciekawym projektem, zawierającym wiele użytecznych mechanizmów obecnych we współczesnych językach programowania.

Powstawały jednak inne projekty. Użytkownicy wczesnych komputerów szybko dostrzegli to, co Zuse przewidział już w latach 40. XX wieku: programowanie komputerów za pomocą ciągów liczb reprezentujących rozkazy jest bardzo uciążliwe. Konstrukcja komputera zaproponowana przez von Neumanna polegała na tym, że komputer pobierał z pamięci kolejne liczby i interpretował je jako rozkazy. Rozkazy mogły zmieniać zawartość pamięci, wprowadzać dane wejściowe, wyprowadzać dane wyjściowe lub wpływać na to, jaki rozkaz będzie wykonany w następnej kolejności. Nie od razu było to jasne, ale żeby dało się zaprogramować pewne obliczenia, potrzebny był **rozkaz rozgałęzienia**. Taki rozkaz polega na tym, że maszyna wykonuje różne akcje w zależności od prawdziwości pewnego warunku. Przykładem problemu, który jest trudno zrealizować bez użycia rozkazu rozgałęzienia, jest znajdowanie elementu maksymalnego w ciągu liczb dodatnich: wczytaj ciąg liczb i wyprowadź tę z nich, która ma największą wartość. Jeśli ograniczymy się do operacji arytmetycznych (+, –, *, /), to tego zadania w ogóle nie da się rozwiązać. Można to zrobić, jeżeli dysponujemy dodatkową operacją obliczania modułu liczby, czyli wartości bezwzględnej. Łatwo sprawdzić prawdziwość następującej równości:

$$\max(A, B) = (A + B + |A - B|) / 2$$

Kod maszynowy, realizujący obliczenie według powyższego wzoru ma następującą postać:

1	IN	1	do komórki nr 1 wprowadź liczbę A z wejścia (np. z karty perforowanej)
2	IN	2	do komórki nr 2 wprowadź liczbę B z wejścia (np. z karty perforowanej)
3	MOV	1,3	zapisz wartość z komórki nr 1 (A) do komórki nr 3
4	SUB	2,3	odejmij wartość z komórki nr 2 (B) od wartości w komórce nr 3 (A) i zapisz w komórce nr 3 (A – B)
5	ABS	3,3	w komórce nr 3 weź wartość bezwzględną jej zawartości
6	ADD	2,1	do wartości w komórce nr 1 (A) dodaj wartość z komórki nr 2 (B)
7	ADD	3,1	do wartości w komórce nr 1 (A+B) dodaj wartość z komórki nr 3 (|A – B|)
8	DIV	#2, 1	podziel wartość w komórce nr 1 (A+B+|A – B|) przez 2
9	OUT	1	wyprowadź wartość z komórki nr 1 ((A+B+|A – B|) / 2) na urządzenie wyjściowe

A jak obliczyć maksimum z trzech liczb bez użycia rozkazu rozgałęzienia? Zamiast od początku pisać nowy program, skorzystajmy z obserwacji, że:

$\max(A, B, C) = \max(\max(A, B), C)$,

a wtedy obliczenia mogą mieć następujący przebieg:

$\max(A, B) = (A + B + |A - B|) / 2$
$\max(X, C) = (X + C + |X - C|) / 2$
podstawiamy max(A, B) za X:
$\max(\max(A, B), C) = ((A + B + |A - B|) / 2 + C + |(A + B + |A - B|) / 2 - C|) / 2$

Program, który służy do obliczania wartości tego wyrażenia ma podobną postać jak wyżej, przy czym początkowy fragment kodu (obliczanie max (A, B)) jest identyczny, a zmiany zaczynają się od rozkazu 9 (w prawej kolumnie komentujemy tylko, jakie wartości są obliczane w poszczególnych komórkach przez kolejne rozkazy):

1	IN	1	1: A
2	IN	2	2: B
3	MOV	1,3	3: A
4	SUB	2,3	3: A – B
5	ABS	3,3	3: |A – B|
6	ADD	2,1	1: A + B
7	ADD	3,1	1: A + B + |A – B|
8	DIV	#2, 1	1: (A + B + |A – B|) / 2
9	IN	2	2: C
10	MOV	1,3	3: (A + B + |A – B|) / 2
11	SUB	2,3	3: (A + B + |A – B|) / 2 – C
12	ABS	3,3	3: |(A + B + |A – B|) / 2 – C|

13	ADD	2,1	1: (A + B + |A – B|) / 2 + C
14	ADD	3,1	1: (A + B + |A – B|) / 2 + C + |(A + B + |A – B|) / 2 – C|
15	DIV	#2,1	1: ((A + B + |A – B|) / 2 + C + |(A + B + |A – B|) / 2 – C|) / 2
16	OUT	1	wyprowadź wartość komórki nr 1

Rozkazy 2-8 i 9-15 są identyczne, ponieważ realizują to samo działanie – obliczanie maksimum dwóch liczb: rozkazy 2-8 obliczają maksimum z liczb A i B, natomiast rozkazy 9-15 obliczają maksimum z liczb max(A, B) i C. Ten schemat można w prosty sposób rozszerzyć na cztery liczby A, B, C, D. Wystarczy w tym celu jeszcze raz skopiować rozkazy 2-8 i wstawić przed rozkaz 16. Uzasadnienie i wykonanie tego rozszerzenia programu pozostawiamy Czytelnikowi.

Jak widać, bez użycia rozkazu rozgałęzienia można podać program służący do obliczania maksimum z ciągu liczb o ustalonej długości, nie można jednak podać jednego programu, który będzie działał dla ciągu o dowolnej długości, np. dla ciągu, którego koniec reprezentuje liczba – 1 (tzw. wartownik końca ciągu). Z rozkazem rozgałęzienia staje się to natomiast możliwe:

1	IN	1	1: pierwsza liczba z wejścia
2	IN	2	2: K z wejścia
3	CMP	# –1,2	pomiń następny rozkaz, jeśli wartość w komórce nr 2 jest równa –1
4	JMP	+2	przejdź do wykonania rozkazu 6 (o dwa rozkazy dalej)
5	JMP	+8	przejdź do wykonania rozkazu 13 (o osiem rozkazów dalej))
6	MOV	1,3	3: zapisz dotychczasowe maksimum z komórki nr 1 do komórki nr 3
7	SUB	2,3	3: odejmij liczbę K od wartości w komórce nr 3
8	ABS	3,3	3: weź wartość bezwzględną z wartości w tej komórce
9	ADD	2,1	1: dodaj wartość z komórki nr 2 do wartości w komórce 1
10	ADD	3,1	1: dodaj wartość z komórki nr 3 do wartości w komórce nr 1
11	DIV	#2, 1	1: podziel wartość w komórce nr 1 przez 2
12	JMP	–10	przejdź do wykonania rozkazu 2 (o 10 rozkazów wcześniej)
13	OUT	1	wyprowadź wartość komórki nr 1

Rozkaz rozgałęzienia (np. CMP w powyższym programie) daje komputerom ich prawdziwą moc. Niektóre z pierwszych maszyn liczących nie miały takich możliwości, np. kalkulator programowalny Z1 Konrada Zuse nie potrafił wykonywać rozgałęzień.

Wczesne kalkulatory programowalne nie zawierały rozkazu rozgałęzienia. Każdy program dla takiego kalkulatora można zapisać w postaci wzoru matematycznego. Na przykład, program, który sortuje dwie liczby M[1] i M[2], generuje posortowany ciąg O[1], O[2] według wzorów:

$$O[1] = M[1] / 2 + M[2] / 2 - |M[1] / 2 - M[2] / 2|$$
$$O[2] = M[1] / 2 + M[2] / 2 + |M[1] / 2 - M[2] / 2|$$

Proponujemy w podobny sposób opisać program, służący do sortowania trzech liczb M[1], M[2], M[3], a jeśli to jest zbyt proste, to proponujemy zapisanie sortowania czterech liczb M[1], M[2], M[3], M[4].

5. Język pomaga unikać błędów

Programy komputerowe rosły, a wraz z nimi rosły problemy. Wyobraźmy sobie, że pomiędzy rozkazy 11 i 12 wstawimy nowy rozkaz, powodujący wyprowadzenie dodatkowej informacji na urządzenie wyjściowe:

1	IN	1	1: pierwsza liczba z wejścia
2	IN	2	2: K z wejścia
3	CMP	# –1,2	pomiń następny rozkaz, jeśli wartość w komórce nr 2 jest równa –1
4	JMP	+2	przejdź do wykonania rozkazu 6 (o dwa rozkazy dalej)
5	JMP	+8	przejdź do wykonania rozkazu 12 (o siedem rozkazów dalej))
6	MOV	1,3	3: zapisz dotychczasowe maksimum z komórki nr 1
7	SUB	2,3	3: odejmij liczbę K od wartości w komórce nr 3
8	ABS	3,3	3: weź wartość bezwzględną z wartości w tej komórce
9	ADD	2,1	1: dodaj wartość z komórki nr 2 do wartości w komórce 1
10	ADD	3,1	1: dodaj wartość z komórki nr 3 do wartości w komórce nr 1
11	DIV	#2, 1	1: podziel wartość w komórce nr 1 przez 2
12	OUT	1	wyprowadź dotychczasowe minimum na urządzenie wyjściowe *<-nowy*
13	**JMP**	**–10**	przejdź do wykonania rozkazu 2 (o 10 rozkazów wcześniej)
14	OUT	1	wyprowadź wartość komórki nr 1

Nowy rozkaz ma teraz numer 12, a rozkazy, które były 12. i 13. stały się 13. i 14. Taki program ma jednak poważny błąd: na skutek przesunięcia się dawnych rozkazów 12 i 13, obecny rozkaz 13 (czyli JMP –10) spowoduje kontynuowanie wykonania programu od rozkazu 3 bez wczytania nowej danej, co powinno nastąpić w rozkazie 2. Rozkazy 3-13 będą wykonywane w nieskończonej pętli i program nigdy nie zakończy działania. Błąd polega na tym, że przy wstawianiu rozkazu trzeba uważać na odległości podane w rozkazach **skoków** JMP

(ang. *jumps*) i odpowiednio je poprawiać. To jest bardzo uciążliwe, programiści wymyślali więc mechanizmy umożliwiające unikanie tego kłopotu. Można na przykład nazwać niektóre rozkazy w programie:

```
                IN      1
WCZYTANIE:      IN      2
                CMP     # –1,2
                JMP     +2
                JMP     ZAKOŃCZENIE
                MOV     1,3
                SUB     2,3
                ABS     3,3
                ADD     2,1
                ADD     3,1
                DIV     #2, 1
                JMP     WCZYTANIE
ZAKOŃCZENIE:    OUT     1
```

Człowiek ma potrzebę nazywania: krajów, zjawisk, przedmiotów, innych ludzi. Coś, co nazwane, łatwiej zrozumieć i łatwiej się tym posługiwać. Nazwanie wybranych rozkazów programu umożliwia szybsze zrozumienie struktury programu, a także zapobiega błędom. Jeśli powtórzymy przykład z wstawieniem dodatkowego rozkazu OUT po rozkazie DIV, to dzięki temu, że posługujemy się nazwami, a nie numerami rozkazów, rozkaz JMP nie będą źródłem błędów.

Dalsze uproszczenie struktury programu uzyskuje się przez nazwanie komórek pamięci. Ostatecznie otrzymujemy następujący program:

```
A               DATA                    obliczone dotychczas maksimum
K               DATA                    nowa wczytana liczba
POM             DATA                    wartość pomocnicza do obliczania
                                        |A – K| / 2
                IN      A               A ← wejście
WCZYTANIE:      IN      K               K ← wejście
                CMP     #–1,K           jeżeli K = –1, pomiń następny
                                        rozkaz
                JMP     +2              pomiń następny rozkaz
                JMP     ZAKOŃCZENIE     kontynuuj od rozkazu
                                        ZAKOŃCZENIE
                MOV     A,POM           POM ← A
                SUB     K,POM           POM ← POM – K, czyli w POM
                                        mamy A – K
                ABS     POM,POM         POM ← |POM|, czyli w POM
                                        mamy |A – K|
                ADD     K,A             A ← A + K
                ADD     POM,A           A ← A+POM
                DIV     #2, A           A ← A / 2
                JMP     WCZYTANIE
ZAKOŃCZENIE:    OUT     A
```

Tak ewoluowały wczesne języki programowania. Posługiwanie się nazwami rozkazów, komórek pamięci, bloków rozkazów, często używanych fragmenty programów itd. umożliwiło swobodniejszą komunikacje programistów z komputerami, ale wciąż było daleko do wydawania głosem rozkazów dla komputera HAL 9000. W latach 50. XX wieku pojawił się język **Fortran**, do dzisiaj używany w obliczeniach numerycznych. Program do obliczania maksimum z ciągu liczb może mieć w Fortranie następującą postać (1957):

```
1       READ INPUT TAPE A          wprowadź do A pierwszą liczbę z wejścia
2       READ INPUT TAPE K          wprowadź do K kolejną liczbę z wejścia
3       IF (K) 6,4,4               jeśli K < 0, przejdź do instrukcji 6
                                   jeśli K = 0, przejdź do instrukcji 4
                                   jeśli K > 0, przejdź do instrukcji 4
4       A = (A+K+ABS(A–K))/2       przypisz do A wartość (A + K + |A – K|) / 2
5       GOTO 2                     przejdź do instrukcji 2
6       WRITE OUTPUT TAPE A        wyprowadź wartość z A
7       STOP                       zakończ wykonanie
```

To był postęp, szczególnie jeśli chodzi o wyrażenia algebraiczne. Instrukcja[4] 4 to olbrzymie uproszczenie w porównaniu z ciągiem rozkazów MOV/SUB/ABS/ADD/ADD/DIV z programu w języku maszynowym. Komunikacja z maszyną przeszła na wyższy poziom. Programiści przestali się martwić o całą masę szczegółów (jak odległość w rozkazach JMP, używanie dodatkowych komórek pamięci do mozolnego obliczania wartości wyrażeń arytmetycznych), dzięki czemu mogli bardziej skupić się na szukaniu rozwiązań optymalnych, a nie łatwych do zapisania. Kiedy proste jest proste, trudne staje się możliwe.

Nowe języki umożliwiały wyrażanie złożonych programów, ale nie ma róży bez kolców: wzrósł również stopień złożoności samych języków. Język maszynowy jest prosty. Składa się z kilkudziesięciu (czasem kilkuset) rozkazów. Każdy rozkaz realizuje prostą i łatwą do opisania operację. Rozkazy wykonywane są po kolei, wszystkie odstępstwa od tej zasady (np. rozkazy JMP lub CMP) są jasno widoczne w programie. Nie było łatwo używać tych języków, ale łatwo było je zrozumieć. Programy były duże i złożone, ale język prosty. Nowe języki sprawiły, że programy stały się krótsze i bardziej treściwe. Okazało się jednak, że opisanie, jak funkcjonuje język, nie jest już takie proste. Pułapki kryją się nawet w tak prostych konstrukcjach języka, jak doskonale

[4] Przyjęliśmy tutaj powszechnie stosowaną konwencję, że **rozkaz** oznacza polecenie dla komputera zapisane w kodzie maszynowym lub w języku mnemonicznym, a **instrukcja** jest poleceniem zapisywanym w językach wyższego poziomu, takich jak Fortran, Algol, Ada, Pascal, C, C++.

nam znane z matematyki wyrażenia algebraiczne. Wyrażenie (A + B – ABS (A – B)) / 2 jest[5] równe mniejszej spośród liczb A i B, zatem dla dowolnych liczb A i B zachodzi równość:

min(A, B) = (A + B – ABS(A – B)) / 2

W komputerach zazwyczaj nie można wykonywać obliczeń na dowolnych liczbach całkowitych. Jedna komórka współczesnej pamięci komputerowej może przechowywać tylko skończoną liczbę różnych wartości – w jednym bajcie można zapisać tylko 256 różnych wartości. Ponieważ w obliczeniach zazwyczaj nie występują dowolnie wielkie liczby, przyjmuje się, że wszystkie pojawiające się wartości liczbowe należą do pewnego ograniczonego zakresu, np. [–128..127] lub [–2147483648..2147483647]. Upraszcza to realizację obliczeń, ale wymaga dyscypliny, należy bowiem pilnować, aby wartości pojawiające się podczas obliczeń nie przekroczyły przyjętego zakresu.

Wyrażenie X + Y – Z to w zasadzie to samo, co X + Y + (– Z). Dzięki łączności dodawania, nie ma znaczenia, które dodawanie wykonamy najpierw, wynik powinien być taki sam:

X + Y – Z = (X + Y) – Z = X + (Y – Z)

Jeżeli jednak działania wykonujemy na liczbach z ograniczonego zakresu, to może nas spotkać przykra niespodzianka, np. dla X = 1, Y = 127, Z = 126 otrzymamy:

(X + Y) – Z = 128 – 126 → źle, nastąpiło przekroczenie zakresu [–128..127]
X + (Y – Z) = 1 + (127 – 126) = 1 + 1 = 2 ⟶ poprawnie

W zależności od tego, jak zinterpretujemy wyrażenie (A + B – ABS(A – B)) / 2,

((A + B) – ABS(A – B)) / 2 czy (A + (B – ABS(A – B))) / 2

dla A = 1 i B = 127 otrzymamy w pierwszym przypadku błąd, a w drugim poprawny wynik.

Tego rodzaju problemy stymulowały poszukiwania nowych metod precyzyjnego opisu języków programowania. Zaczęto zwracać uwagę na języki służące do opisu języków. Jednym z takich wynalazków jest metoda wykorzystana do opisu gramatyki języka **Algol 58**, zaproponowana przez Johna Backusa, znana

[5] W wielu językach programowania, wartość bezwzględna (moduł) |x| jest zapisywana ABS(x).

w informatyce pod nazwą **BNF** (Backhus-Naur Form, czyli Postać Backusa-Naura)[6]. W tej notacji wyrażenie można zdefiniować w następujący sposób:

wyrażenie::= wyrażenie znak wyrażenie | liczba | litera | ‘(’wyrażenie‘)’
znak::= ‘+’ | ‘–’ | ‘*’ | ‘/’ | ‘^’
liczba::= cyfra | liczba cyfra
cyfra::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’
litera::= ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’
| ‘J’ | ‘K’ | ‘L’ | ‘M’ | ‘N’ | ‘O’ | ‘P’ | ‘Q’ | ‘R’
| ‘S’ | ‘T’ | ‘U’ | ‘V’ | ‘W’ | ‘X’ | ‘Y’ | ‘Z’

Powyższy opis warto porównać z naiwnym, tekstowym opisem składni wyrażeń algebraicznych zamieszczonym kilka akapitów wcześniej. Słowa wzięte w cudzysłowy to **symbole** opisywanego języka. Słowa bez cudzysłowów to tzw. **metasymbole**, elementy używane do definicji języka. W notacji BNF metasymbole odpowiadają kategoriom elementów opisywanego języka. W tym przypadku kategoriami są: wyrażenia, działania, liczby, cyfry, litery itp. W tej notacji definicje metasymboli nazywa się **klauzulami**.

Klauzulę: wyrażenie::= wyrażenie znak wyrażenie | liczba | litera |
‘(’wyrażenie‘)’

należy czytać (interpretować) następująco:

Wyrażenie może być:
parą wyrażeń połączonych znakiem lub
liczbą lub
literą lub
wyrażeniem wziętym w parę nawiasów.

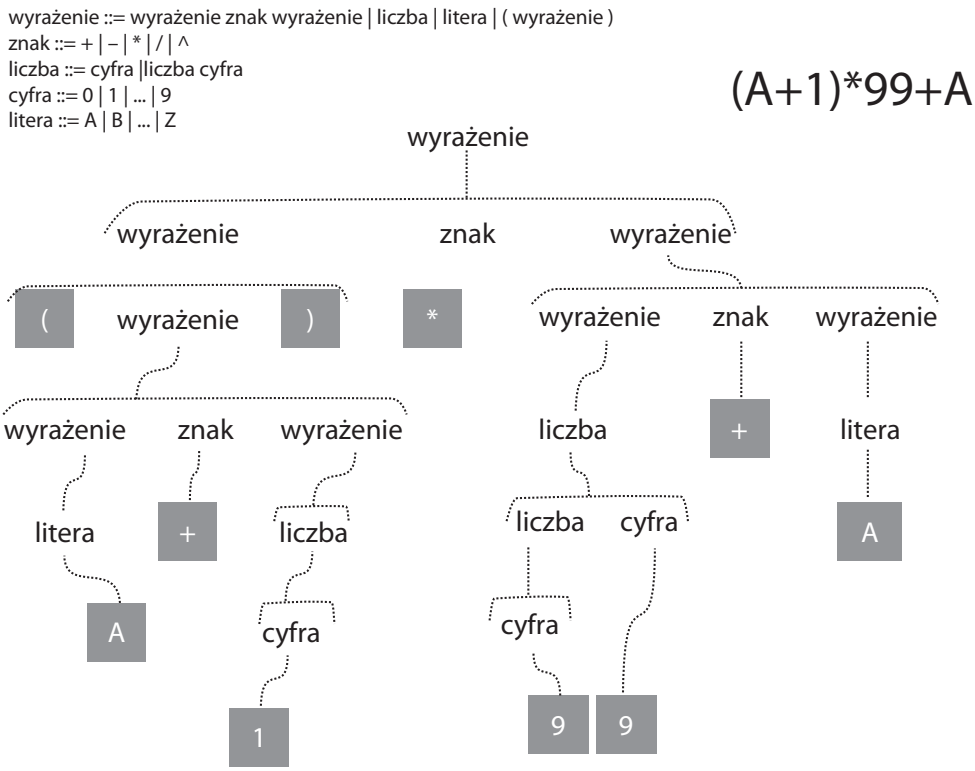
Z kolei klauzulę: liczba::= cyfra | liczba cyfra
należy czytać:

Liczba może być:
pojedynczą cyfrą
liczbą, po której następuje pojedyncza cyfra.

[6] Nie od razu zauważono, że Postać Backhusa-Naura jest bardzo podobna do metody zastosowanej przez Pāniniego do opisu gramatyki Sanskrytu w VI w p.n.e.

Notacja BNF rozwiewa wątpliwości związane z regułami łączenia symboli w wyrażenia języka. Proces tworzenia wyrażenia algebraicznego na podstawie powyższej gramatyki może przebiegać następująco:

(1) weź metasymbol ‘wyrażenie’
(2) zastąp dowolny metasymbol dowolną z klauzul stojących po jego prawej stronie
(3) jeżeli są jeszcze jakieś metasymbole, wróć od punktu (2).

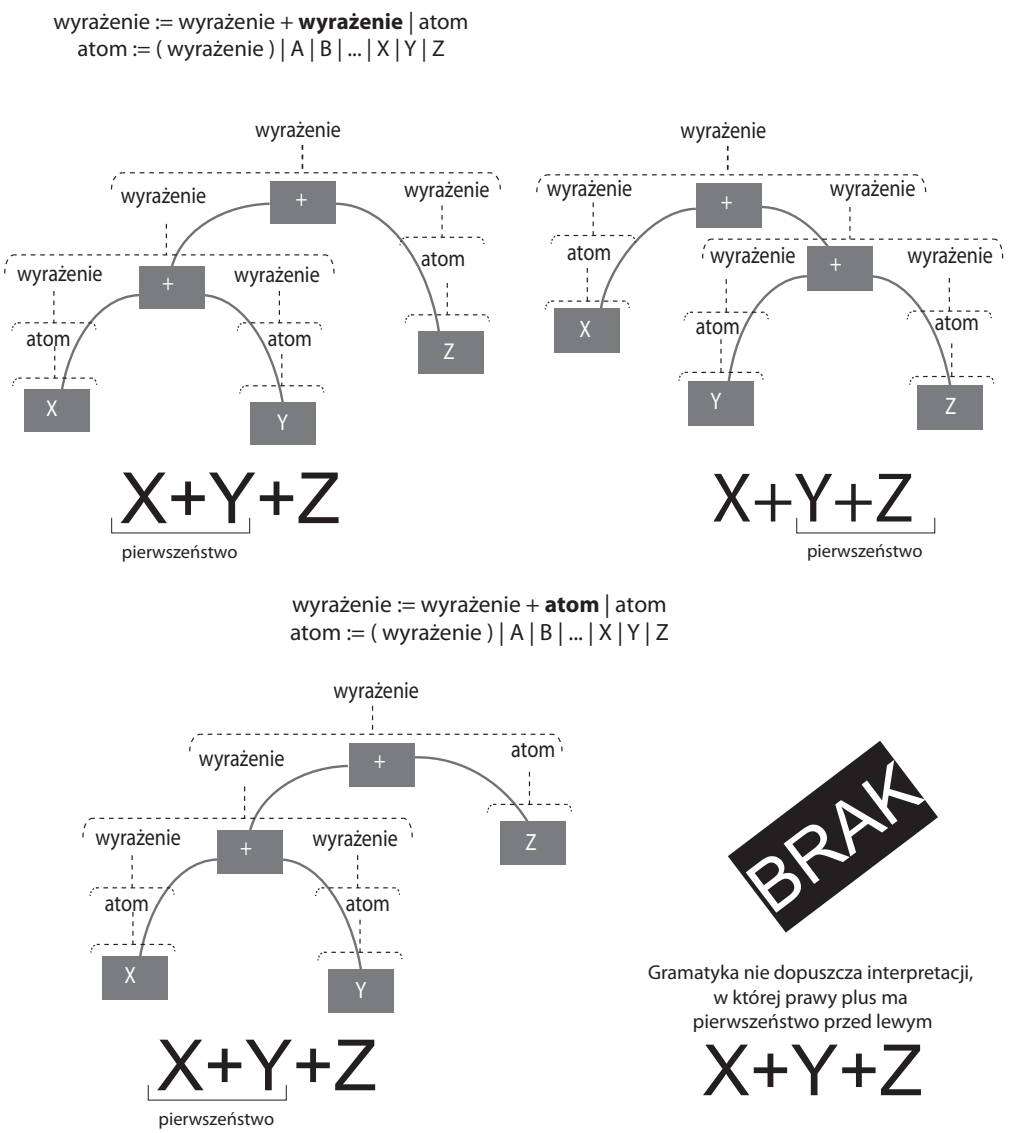


Rysunek 6. Tworzenie wyrażenia (A+1)*99+A na podstawie gramatyki. Ta gramatyka nie opisuje poprawne wyrażenia, ale nie sugeruje właściwej kolejności wykonania działań (np. pierwszeństwo mnożenia przed dodawaniem)

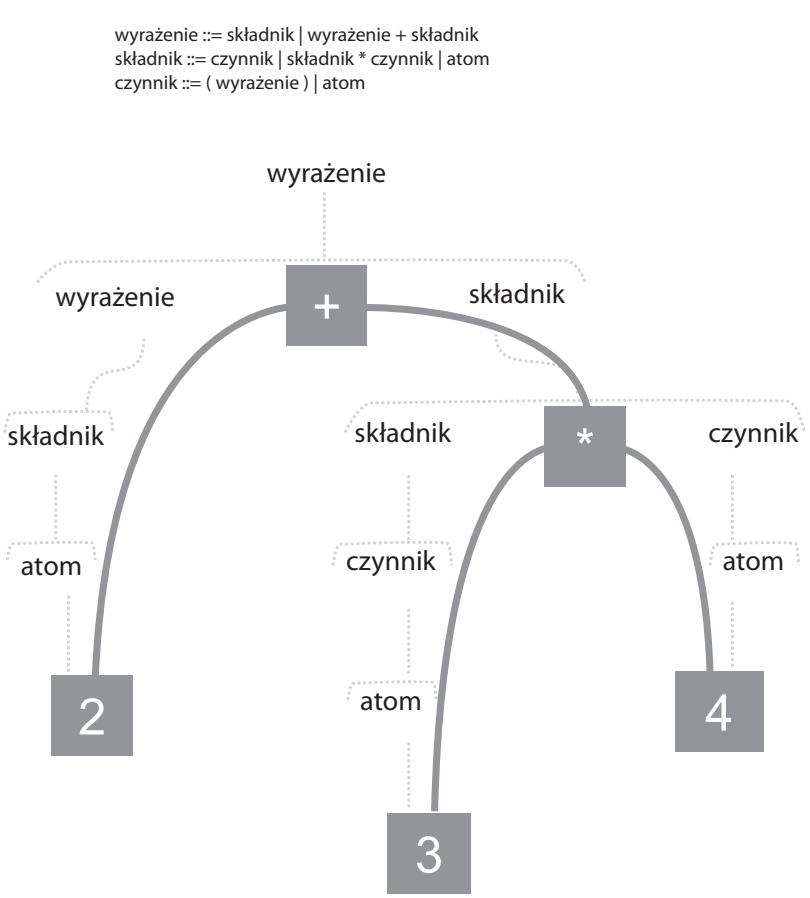
Na zapis w języku BNF można patrzeć z dwóch stron. Po pierwsze jest to opis zasad łączenia symboli pewnego (innego) języka. W powyższym przykładzie opisujemy zasady budowania prostych wyrażeń algebraicznych. Po drugie gramatyka w języku BNF może służyć do interpretacji wyrażeń języka. Na rysunku 7 pokazano dwie gramatyki BNF, opisujące proste wyrażenia.

Gramatyki różnią się jednym szczegółem (w pierwszej dopuszczamy dowolne wyrażenia po obu stronach znaku plus, a w drugiej prawe wyrażenie musi być

albo zmienną, albo wyrażeniem w nawiasach). Pierwsza gramatyka nie precyzuje więc kolejności wykonywania działań i wyrażenie X + Y + Z można w niej zinterpretować równie dobrze jako „najpierw X + Y, potem wynik + Z", jak i „najpierw Y + Z, potem X + wynik". Z kolei druga gramatyka dopuszcza wyłącznie tę pierwszą interpretację. Obie gramatyki opisują ten sam zbiór dopuszczalnych wyrażeń, ale druga gramatyka wymusza interpretację kolejności wykonywania działań. Taki mechanizm umożliwia bardzo precyzyjnie opisywać zachowanie programów, np. kolejność wykonywania obliczeń.



Rysunek 7. Wymuszanie kolejności obliczeń za pomocą gramatyki (w gramatyce na dole lewy plus wiąże mocniej)



Rysunek 8. Wymuszanie kolejności obliczeń za pomocą gramatyki (pierwszeństwo mnożenia przed dodawaniem)

Język Algol odniósł sukces, a notacja BNF stała się popularna. Możliwość zapisu programów za pomocą wyrażeń algebraicznych stała się standardem. Nowe, coraz lepsze języki mnożyły się jak grzyby po deszczu, a wraz z nimi specjalne programy (kompilatory), służące do tłumaczenia tych języków na kod maszynowy. W roku 1960 pojawił się **COBOL**, język programowania, który rozpalał nadzieje, że mit komputera rozumiejącego ludzki język się ziści. Postać programów w języku COBOL bardzo przypominała zdania w naturalnym języku angielskim. Twórcy języka wierzyli, że analitycy, managerowie i inni nie-programiści będą w stanie czytać i rozumieć programy. Niestety, okazało się, że rozdźwięk pomiędzy strukturą języka angielskiego a kodem maszynowym, do którego w końcu trzeba było tłumaczyć wyrażenia w języku COBOL, jest zbyt wielki. Pozorne podobieństwo języka COBOL do języka angielskiego sprowadzało nie-programistów na manowce, a programistom wcale nie ułatwiało pracy. W latach 70. XX wieku okazało się, że struktura programów w tym języku nie ułatwia ich analizy. Edsger Dijkstra, propagator metod programowania opartych

na hierarchicznej strukturze programów (której brakuje programom w języku COBOL), w roku 1975 zgromił COBOL, posuwając się do stwierdzenia, że „używanie języka COBOL kaleczy mózg, nauczanie tego języka powinno więc być uznane za przestępstwo". Pierwsza nadzieja na komunikację z maszyną w języku naturalnym nieco zgasła.

6. Ewolucja języków programowania

Stawało się powoli jasne, że różne osoby będą komunikować się z komputerem w różny sposób. Oprócz specjalistów rozumiejących mechanizmy działania maszyn i szkolonych w metodach ich programowania, pojawili się zwykli użytkownicy – osoby nieprzeszkolone w programowaniu, które dzięki coraz większej dostępności komputerów uzyskały do nich bezpośredni dostęp, by realizować zadania naukowe lub komercyjne. Ewolucja języków komunikacji z komputerem poszła w dwóch kierunkach.

6.1. Język powłoki – konwersacja z maszyną

W roku 1969 pojawił się system operacyjny **Unix**. Był dziełem niewielkiego zespołu zdolnych programistów pracujących dla firmy Bell Labs. **System operacyjny** to główny program sterujący pracą komputera, zarządzający jego zasobami (pamięcią wewnętrzną, pamięcią dyskową, czasem procesora) oraz umożliwiający uruchamianie innych programów, tzw. aplikacji. Twórcy systemu Unix zaprojektowali go tak, by główna część systemu była możliwie mała, a jak najwięcej zadań było realizowanych przez niewielkie, odizolowane programy narzędziowe. Funkcje systemu uruchamiało się z tzw. **programu powłoki** (ang. *shell*). Program powłoki współpracował z użytkownikiem na zasadzie konwersacji: użytkownik wpisywał polecenie, a program powłoki odpowiadał, wypisując informację za pomocą drukarki lub terminala ekranowego. Program powłoki udostępniał prosty język poleceń. Jeśli jednak użytkownik wpisał polecenie nierozpoznawane przez program powłoki, system próbował znaleźć na dysku program o tej nazwie, co nieistniejące polecenie i uruchomić go. Dzięki takiemu rozwiązaniu rozszerzenie programu powłoki o nowe polecenia było bardzo proste – wymagało tylko zainstalowania na dysku programów o odpowiednich nazwach. Ponieważ środowisko systemu Unix było przyjazne dla programistów, język powłoki systemu Unix (wraz z nazwami najpopularniejszych programów narzędziowych) stał się niezwykle popularny wśród zawodowych programistów. Język powłoki ewoluuje do dziś. Występuje w dwóch głównych odmianach (Bourne-shell i C-shell). Daleko mu do swobody porozumiewania się z komputerem HAL 9000, ale opiera się na podobnej zasadzie – operator prowadzi konwersację z systemem.

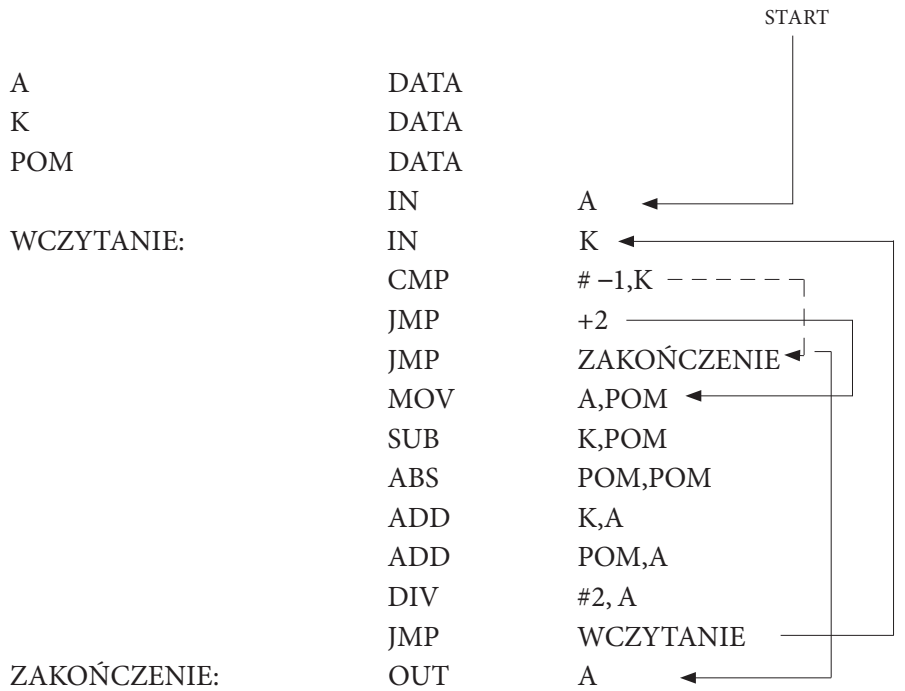
Sam program powłoki byłby jednak bezużyteczny, gdyby nie masa programów i programików, w które obrósł system Unix. Dziesiątki, setki, a z czasem tysiące małych i dużych programów wypełniających funkcje od najprostszych (zliczanie rozmiarów plików, wyszukiwanie frazy tekstowej, sortowanie) do bardzo złożonych (tłumaczenie programów na inne języki, skład tekstów, rozwiązywanie dużych układów równań) powstały i obsiadły system Unix jak rój pszczół.

Współczesne popularne systemy GNU/Linux mają budowę odziedziczoną po systemie Unix. Na system składa się duża liczba luźno powiązanych programów realizujących przeróżne funkcje, zarówno bardzo proste, jak i bardzo złożone. Na przykład system Ubuntu Linux zawiera ponad 38 tysięcy programów służących do wykonywania wszelkich wyobrażalnych zadań, które jest dziś w stanie wykonać komputer. Wiele z tych programów jest zbudowanych w ten sposób, że można nimi sterować z poziomu programu powłoki, często twórcy programów zachowują przy tym podobny styl komunikacji, różne programy obsługuje się zatem podobnie. Ta menażeria składa się na język komunikacji z maszyną. By rozpocząć konwersację w tym języku, wystarczy włączyć dowolny system GNU/Linux i uruchomić program Terminal.

6.2. Wojna z instrukcją Goto

Coraz większe komputery i coraz wygodniejsze języki programowania dawały coraz większe możliwości, a apetyty użytkowników rosły w miarę jedzenia. Pod koniec lat 60. XX wieku pojawiła się pierwsza mysz komputerowa, w latach 70. rozwinęły się interfejsy graficzne, stworzono gry zręcznościowe oraz arkusz kalkulacyjny. Stare nawyki umierają jednak powoli. Nowe języki programowania (C, Pascal) umożliwiały zwięzłe i eleganckie zapisywanie programów, jednak wielu programistów nie potrafiło się przestawić i skorzystać z tych udogodnień. Jedną z trudności w programowaniu za pomocą języka maszynowego są rozkazy powodujące zmianę miejsca wykonania kodu, tzw. **rozkazy skoku**. Na rysunku 9 pokazano program maszynowy opisywany wcześniej, z zaznaczonymi rozkazami skoków. Skoki są w swej naturze asymetryczne. Łatwo wypatrzeć początek skoku, ale trzeba się uważnie rozejrzeć by zauważyć, że np. rozkaz „MOV A,POM" jest zakończeniem jakiegoś skoku. Po uważnej analizie okolicznych rozkazów stwierdzimy, że faktycznie dwa wiersze wyżej znajduje się rozkaz „JMP +2", który powoduje skok do rozkazu „MOV A,POM". Stwierdzenie, czy w programie istnieją jakiekolwiek inne rozkazy skoku prowadzące do „MOV A,POM" wymaga przejrzenia... całego programu! To nie jest pro-

blemem, gdy program jest krótki, ale im dłuższy, tym trudniej mieć pewność (a rozmiar współczesnych programów sięga milionów rozkazów maszynowych). Nie koniec jednak na tym.



Rysunek 9. Program maszynowy z oznaczonymi instrukcjami skoków

Rozkazy skoków na rysunku 9 oznaczono strzałkami. Strzałka prowadząca od JMP WCZYTANIE do WCZYTANIE biegnie pod prąd normalnego kierunku wykonywania rozkazów, przez co pewna grupa rozkazów jest wykonywana wielokrotnie. Taką grupę rozkazów nazywamy **pętlą**. Dzięki pętli program może działać dla ciągów danych o różnej długości, powtarzając tę samą sekwencję rozkazów dla kolejnych elementów. Pętla daje komputerom moc. Bez pętli programowanie jest proste, ale słabe. Wielu programów nie da się napisać bez pętli. Pętle są możliwe dzięki rozkazom rozgałęzienia (CMP), inaczej wykonanie rozkazów pętli trwałoby w nieskończoność – rozgałęzienie jest potrzebne, aby zakończyć wykonanie pętli. Stwierdzenie, czy w programie wykonanie pętli kończy się poprawnie, jest podstawowym i czasami bardzo trudnym problemem przy pisaniu programów. Pod koniec lat 60. minionego stulecia teoretycy programowania, Robert Floyd oraz Antony Hoare, opracowali precyzyjne metody wnio-

skowania, pomagające stwierdzić, czy pętle w programie kończą się poprawnie. I tu wracamy do strzałek: strzałka od JMP ZAKOŃCZENIE i strzałka od JMP WCZYTANIE przecinają się. Jeżeli strzałki skoków mają przecięcia, to metody wnioskowania o pętlach bardzo się komplikują. Ale sprawa nie jest beznadziejna. Już w połowie lat 60. inni teoretycy (Corrado Böhm i Giuseppe Jacopini) wykazali, że każdy program można tak napisać, żeby strzałki się nie przecinały! Wystarczy chcieć i mieć taki nawyk. Najwyraźniej nie był to nawyk powszechny w roku 1968, kiedy holenderski informatyk i luminarz Edsger Dijkstra wylał swoją frustrację w artykule *Go To Statement Considered Harmful* (pl. Instrukcja Go To jest niebezpieczna). Wiele ówczesnych języków programowania (np. Fortran, Algol) zawierało instrukcję **Go To**, która była odpowiednikiem rozkazu skoku w języku maszynowym. Dijkstra argumentował, że jej nieodpowiedzialne użycie powoduje powstawanie programów złożonych i trudnych do analizy.

Artykuł Dijkstry był elementem szerszego ruchu znanego pod nazwą **programowanie strukturalne**, nawołującego do ograniczenia środków wyrazu przy pisaniu programów w celu polepszenia ich czytelności i jakości. Takie zmiany wymagają odejścia od przyzwyczajeń i nie dokonują się szybko. Realny wpływ nawoływań zwolenników programowania strukturalnego na kształt nowych języków programowania dało się zauważyć dopiero w latach 90.

To może wydawać się dziwne, ale wszystkie dominujące języki programowania począwszy od lat 60. po dzień dzisiejszy, są do siebie bardzo podobne. Wynika to z faktu, że do końca XX wieku w zasadzie wszystkie wytwarzane przemysłowo komputery miały konstrukcję zasadniczo nieodbiegającą od modelu von Neumanna.

Tak jak w większości języków naturalnych można wyróżnić rzeczowniki, czasowniki, przymiotniki i przysłówki, tak w większości współczesnych języków programowania można wyróżnić cztery podstawowe kategorie gramatyczne:

- **zmienne** (zazwyczaj oznaczane literami np. A, B... lub słowami), które reprezentują fragmenty pamięci komputera i służą do przechowywania danych;
- **wyrażenia** (zapisywane, jak wyrażenia algebraiczne w matematyce), które opisują niewielkie fragmenty obliczeń, np. (A + B + ABS(A – B)) / 2;
- **instrukcje przypisania** (zapisywane za pomocą znaków =,:= lub ←), które powodują przypisanie obliczonych wartości wyrażeń zmiennym, którym odpowiadają komórki pamięci;
- **instrukcje sterujące** (**if**, **goto**, **while**, **for**, **case**, **switch**, **call**...), które mają funkcje podobne do skoków i rozgałęzień w języku maszynowym, tj. przesądzają o tym, które fragmenty kodu będą wykonywane.

Zwolennicy programowania strukturalnego postulowali porzucenie nawyku tworzenia pętli za pomocą instrukcji **goto**. Współczesne języki programowania

na ogół zawierają pętlę **while,** którą można wykorzystać zamiast **goto** – gramatyka tej pętli ma postać:

instrukcja::= **while** warunek (wyrażenie logiczne) **do** blok |...
blok::= instrukcja | { ciąg-instrukcji }
ciąg-instrukcji::= instrukcja | ciąg-instrukcji **;** instrukcja

Wykonanie tej pętli polega na sprawdzeniu prawdziwości warunku i, jeśli jest spełniony, wykonaniu instrukcji oraz powrotu do wykonania pętli od nowa. Poniżej jest pokazana relacja między instrukcją **while** a **goto**.

```
while (WARUNEK) {
        INSTRUKCJE
}
```

```
pętla:
if (not WARUNEK) goto koniec;
… INSTRUKCJE …
        goto pętla
koniec:
```

zagnieżdżone pętle	zazębione pętle
… A: … … B: … … JMP B … JMP A …	… A: … … B: … … JMP A … JMP B …

while (X < N) {
 …
 while (Y < M) {
 …
 }
 }
	gramatyka instrukcji **while** uniemożliwia zapisanie zazębionych pętli

Rysunek 10. Zazębiające się pętle

Na rysunku 10 zilustrowano niekorzystne zjawisko zachodzenia na siebie dwóch pętli w programie maszynowym. Takie zazębianie się pętli utrudnia analizę (strzałki skoków przecinają się). Pętla **while** uniemożliwia zazębienie się dwóch pętli.

Instrukcję **goto** wyłączono z języka Java, nie ma jej także w języku Python. Wiele innych współczesnych języków wciąż zawiera instrukcję **goto**, np. ogłoszony w roku 2000 język C#.

6.3. Podprogramy – rozszerzanie języków

```
void main() {
   int A, K;
   scanf(„%d", &A);
wczytanie:
   scanf(„%d", &K);
   if (K == – 1) goto koniec;
   A = (A + K + abs(A – K)) / 2;
   goto wczytanie;
zakończenie:
   printf(„%d\n", A);
}
```

```
int wprowadź_liczbę() {
   int liczba;
   scanf(„%d", &liczba);
   return liczba;
}

int wyprowadź_liczbę(int liczba) {
   printf(„%d\n", liczba);
}

int minimum(int x, int y) {
   return (x + y + abs(x – y)) / 2;
}

void main() {
   int A, K;
   A = wprowadź_liczbę();
   while ((K = wprowadź_liczbę()) != –1)
{
      A = minimum(A, K);
   }
   wyprowadź_liczbę(A);
}
```

Rysunek 11. Niestrukturalna i strukturalna wersja programu, służącego do obliczania maksimum w języku C; użycie podprogramów zwiększa rozmiar kodu źródłowego, ale bardzo podnosi czytelność

Drugim postulatem programowania strukturalnego było posługiwanie się **podprogramami**. Wiele języków programowania ma możliwość nazwania i wyodrębnienia fragmentu kodu, który później może zostać użyty przez wymienienie nazwy. Jest to ta sama filozofia, która leży u podstaw systemu Unix – system (w tym przypadku język) składa się z małej liczby podstawowych elementów, do których użytkownik może dodać nowe elementy, opisane za pomocą tych już istniejących. Na podprogramy można patrzeć jak na mechanizm umożliwiający rozszerzenie języka o nowe wyrażenia.

Na rysunku 11 ilustrujemy postulaty programowania strukturalnego w praktyce. Program po lewej stronie jest napisany bez poszanowania zasad programowania strukturalnego, a ten z prawej korzysta ze strukturalnej pętli **while** oraz z podprogramów. Wszystkie, nawet bardzo proste operacje, zostały zamknięte w podprogramy, którym nadano czytelne nazwy. Nawet osoba nierozumiejąca do końca użytych w programie wyrażeń (np. scanf albo int) jest w stanie zrozumieć program po prawej, właśnie dzięki obecności zrozumiałych nazw podprogramów (wprowadź_liczbe, wyprowadź_liczbę, minimum).

Mechanizm podprogramów zapewnia nie tylko czytelność, ale także umożliwia często używane kawałki kodu napisać raz i odłożyć do ponownego użytku. W latach 80. XX wieku popularne stały się tzw. **biblioteki podprogramów** – zbiory podprogramów, przeznaczonych do konkretnych zadań (np. obliczeń na macierzach, obliczeń statystycznych, generowania grafiki trójwymiarowej itp.). Wielkim przebojem okazała się biblioteka STL dla języka C++, a niektóre języki zdobyły olbrzymią popularność dzięki łatwo dostępnym rozległym bibliotekom (np. Perl, Java).

Szacowana liczba języków programowania to dziś kilka tysięcy. W powszechnym użyciu jest jednak tylko kilkanaście z nich, reszta to języki wymarłe niszowe albo nowatorskie.

7. I co dalej?

Firma Apple 28 kwietnia 2010 roku kupiła firmę Siri, producenta oprogramowania o tej samej nazwie. Oprogramowanie Siri to automatyczny asystent, system adaptacyjnie rozpoznający ludzką mowę, zdolny do wykonywania prostych zadań związanych z wyszukiwaniem informacji i rezerwacją terminów w kalendarzu. Jesteśmy chyba wciąż bardzo daleko od komunikowania się z komputerami z taką swobodą, jak astronauci w *Odysei Kosmicznej 2001* Stanleya Kubrika. Pomiędzy językami naturalnymi, w którym komunikujemy się ze współplemieńcami, a językami programowania, w których instruujemy komputery, zieje wciąż olbrzymia otchłań. Programowanie cały czas wymaga dużej ostrożności i solidnego treningu. Z drugiej strony – postęp nie ustaje. Oto kilka pomysłów, które pojawiły się na przestrzeni ostatnich pięćdziesięciu lat:

Języki obiektowe

Wymyślone na początku lat 70. ubiegłego wieku i rozpropagowane w latach 80. były udaną próbą rozwiązania problemu nadmiernych zależności pomiędzy komponentami dużych systemów. Do tej rodziny należą języki Simula, Smalltalk, C++, Java, C# i działający w każdej przeglądarce internetowej JavaScript.

Języki funkcyjne

Powstałe w połowie lat 70. zrywają z tradycyjnym modelem *zmienne/wyrażenia/instrukcje*. W niektórych językach funkcyjnych w ogóle nie ma instrukcji. To pomaga uzasadniać poprawność napisanych w nich programów. Te języki są ważne również z tego powodu, że coraz więcej komputerów produkowanych w XXI wieku nie jest już zgodnych z tradycyjnym modelem von Neumanna, w którym instrukcje są wykonywane po kolei. Współczesne komputery osobiste wykonują wiele wątków obliczeń równolegle, choć wciąż te możliwości są słabo wykorzystane, m.in. ze względu na brak odpowiednich języków programowania. Wydaje się, że języki funkcyjne mogą lepiej nadawać się do programowania takich komputerów. Do tej rodziny należą: Lisp, SML, Haskell, Ocaml.

Języki skryptowe

Przedkładają wygodę pisania i uruchamia kodu oraz łatwość użycia nad wydajność. Zyskały popularność w latach 90., kiedy wydajność komputerów osobistych wzrosła na tyle, by wolniejsze wykonanie programów w językach skryptowych nie było uciążliwe dla użytkownika. Jeśli program napisany np. w języku C działa sekundę, to jego wersja napisana w języku Python może wymagać nawet 10 sekund. Szybkie komputery sprawiły, że języki skryptowe przestały się w dostrzegalny sposób ślamazarzyć. Do tej rodziny należą Python, Perl, Lua, Ruby.

Języki powłoki

To języki zintegrowane ze środowiskiem systemu operacyjnego. Można w nich pisać programy, ale np. oprogramowywanie obliczeń nie jest specjalnie wygodne. Spisują się natomiast bardzo dobrze jako języki dialogu operatora z systemem operacyjnym oraz spoiwo do sklejania ze sobą mniejszych programów, przekazywaniem danych między nimi i automatyzacją nudnych zadań. Wśród nich: Bash, TCSH, ZSH.

Języki przetwarzania danych

Służą do operacji na zbiorach danych[7]. To szeroka i różnorodna rodzina, jej członkiem jest zarówno leciwy, acz bardzo popularny język SQL do tworzenia

[7] Więcej na temat baz danych i ich języków można przeczytać w rozdziale *Homo informaticus colligens, czyli człowiek zbierający dane.*

i wyszukiwania informacji w bazach danych, ale także język Xpath do wyszukiwania informacji w ustrukturyzowanych plikach tekstowych. Do tej rodziny można również zaliczyć język akceptowany przez wyszukiwarkę Google, pozwalający modyfikować lub precyzować wyniki wyszukiwania.

Języki pomocnicze

Powstało wiele małych wyspecjalizowanych języków programowania, służących do jednego konkretnego celu. Na przykład język Sed służy do prostych operacji tekstowych. Pewne jego operacje rozumie nawet komunikator Skype! Jeżeli pomylisz się w czacie Skype (np. napiszesz „gura" zamiast „góra"), to już po wysłaniu komunikatu możesz wpisać w czat instrukcję języka Sed s/gura/góra/, a błąd w poprzednim komunikacie zostanie poprawiony! Inne języki z tej grupy to Awk i wyrażenia regularne (*regular expressions*).

Istnieje też pokaźna grupa sztucznych języków związanych z informatyką, które nie służą do pisania programów. Wśród nich jest rodzina języków do opisu wyglądu dokumentów i stron WWW (HTML, CSS, XLST), rodzina języków do opisu dokumentów przeznaczonych do druku (TeX, PostScript, Troff), języki do opisu trójwymiarowych scen dla programów generujących grafikę komputerową (np. PovRay), języki do opisu zachowania i topologii układów scalonych (Verilog, VHDL) itp.

Istniejących języków programowania jest kilka tysięcy. Ich ewolucja trwa od pół wieku. Z upływem czasu pojawia się coraz więcej wątpliwości, czy któryś z nich kiedykolwiek zbliży się do języka ludzkiego, umożliwiając człowiekowi rozmowę z maszyną tak, jak w *Odysei Kosmicznej 2001*. Być może tak się kiedyś stanie. Rok 2001 był jednak terminem zbyt ambitnym.

Epilog – Których języków programowania się uczyć?

Współczesny programista nie musi znać wielu języków programowania. Zazwyczaj dobrze zna kilka, a kilkanaście zna pobieżnie. Nie chodzi jednak o ilość. Wyobraź sobie, że uczysz się języków naturalnych: polskiego, rosyjskiego, słowackiego. Czy znając te języki warto uczyć się ukraińskiego? Zapewne nie, bo jest on podobny do trzech języków, które już znasz. Jeśli znasz C++, Javę i Pascala, zainwestuj w naukę języka, który jest istotnie odmienny. Oto subiektywna propozycja portfolio dla informatyka:

- przynajmniej jeden wysokowydajny popularny obiektowy język kompilowany (np. Java, C++, C#, Ada),
- język maszynowy jakiegoś procesora, bo warto wiedzieć, jak programy działają „na samym spodzie",
- język C, bo jest to wciąż najpopularniejszy język do pisania systemów operacyjnych i sterowników,

- język funkcyjny (Haskel, Standard ML, Ocaml),
- popularny wygodny język skryptowy (np. Python, Ruby, Lua) do pisania programów, w których poprawność jest dużo ważniejsza niż wydajność obliczeń (np. serwery WWW),
- język używanej powłoki (np. język powłoki Windows, Bash, ZSH, TCSH),
- języki opisu dokumentów WWW (obecnie HTML i CSS, ew. XML),
- dominujący język do implementacji interfejsów użytkownika w aplikacjach WWW (obecnie JavaScript),
- języki do operacji na tekstach, przydatne przy automatyzacji małych codziennych zadań (Sed, Awk, Perl),
- język o silnym systemie modułów (np. Modula-2, Standard ML, Ada),
- język skryptowy o dużej bibliotece (np. Python, Perl), do szybkiej implementacji zadań prostych, ale żmudnych (np. ściągnij stronę WWW przez HTTPS, zanalizuj jej treść w języku HTML i podaj rozmiar w znakach pisarskich),
- dominujący język zapytań baz danych (obecnie SQL),
- dominujący język programowania na wybranej dużej platformie mobilnej (np. iOS → Objective-C, Android → Java).

Oprócz tego, tak jak humaniście przystoi znać języki klasyczne łacinę i grekę, tak programiście wypada znać Lisp i Smalltalk.

Literatura

1. Dijkstra E.W., *Go To Statement Considered Harmful*, „Comm. ACM" 11(3) 1968
2. Dijkstra E.W., *Umiejętność programowania*, WNT, Warszawa 1978
3. Harel D., Feldman Y., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 2008
4. Von Neumann, J., *First Draft of a Report on the EDVAC*, 1945, http://systemcomputing.org/turing%20award/Maurice_1967/TheFirstDraft.pdf
5. Wirth N., *Algorytmy + Struktury Danych = Programy*, WNT, Warszawa 1999

Grzegorz Jakacki ukończył studia na Wydziale Matematyki i Informatyki Uniwersytetu Warszawskiego z tytułem magistra informatyki, przedstawiając pracę magisterską dotyczącą przyrostowego sposobu opisu semantyki języków programowania. Podczas studiów uczestniczył w pracach jury Olimpiady Informatycznej i Zawodów w Programowaniu Zespołowym. Od roku 2000 pracował jako programista, projektant oprogramowania i tutor w USA, Chinach i Polsce. W latach 2000-2001 oraz 2008-2009 był członkiem zespołu opracowującego projekt i implementację fragmentu języka Verilog, służącego do opisu własności układów scalonych. W latach 2001-2006 prowadził społeczny projekt rozwoju kompilatora czołowego OpenC++. Pracował nad systemem analizy kodu źródłowego w języku SystemC (2001-2003) oraz nad interpreterem języka Verilog-AMS (2003-2005). Przełożył na język polski książki: *Nowoczesne programowanie w C++* oraz *Sztuka Programowania* (Tom I). Jest założycielem i prezesem firmy Codility Ltd., zajmującej się automatycznym testowaniem umiejętności poprawnego programowania. Sekretarz społecznego inkubatora technologicznego Warszawski Hackerspace. Prowadzi zajęcia z praktyki programowania na Uniwersytecie Warszawskim.

jakacki@codility.com

Krzysztof Diks, Wojciech Śmietanka
Instytut Informatyki
Uniwersytet Warszawski

Kubełkowe struktury danych

W niniejszym rozdziale przedstawiamy struktury danych, które są proste koncepcyjnie, stosunkowo łatwe w implementacji i nieźle zachowują się w praktyce. Pomimo swojej prostoty i dobrego zachowania, opis tych struktur trudno znaleźć w podręcznikach algorytmiki, dlatego idee przedstawione tutaj powinny zainteresować każdego, kto chciałby poszerzyć swoją wiedzę algorytmiczną. Dla każdego z problemów omawianych w tym rozdziale względnie łatwo zaprojektować algorytm działający w czasie kwadratowym ze względu na rozmiar danych. Idee tutaj zaprezentowane umożliwiają otrzymanie algorytmów działających o czynnik pierwiastkowy szybciej. W szczególności pokazujemy, w jaki sposób obliczyć w czasie $O\left(n\sqrt{n}\right)$ wektor inwersji dla zadanej n-elementowej permutacji oraz jak zaimplementować algorytm Dijkstry, żeby działał w czasie $O\left(m+n\sqrt{c}\right)$, gdzie n to liczba wierzchołków grafu, m liczba jego krawędzi, a c jest całkowitoliczbowym ograniczeniem górnym na długości krawędzi, przy czym zakładamy, że te długości są także całkowitoliczbowe. Oprócz przedstawienia ciekawych struktur danych i ich zastosowań, nie mniej ważny jest proponowany sposób rozwiązywania problemów algorytmicznych, polegający na formułowaniu najpierw algorytmów jak najbardziej abstrakcyjnie, a następnie poszukiwaniu możliwie najlepszych metod implementacji wykorzystywanych w opisie algorytmów abstrakcyjnych konstrukcji.

1. Wstęp

Z pojęciem **kubełków** każdy interesujący się programowaniem i algorytmiką miał pewnie okazję się spotkać przy okazji poznawania algorytmów sortowania. **Sortowanie kubełkowe** [3, 8] stosuje się wtedy, gdy o elementach porządkowanego ciągu wiemy, że są liczbami całkowitymi z przedziału $[0, c-1]$ (lub danymi, którym można przypisać liczby całkowite, uwzględniając ich względny porządek), dla znanej z góry wartości parametru c. Sortowanie wówczas polega na rozrzuceniu sortowanych liczb do kubełków indeksowanych liczbami z przedziału $[0, c-1]$ w taki sposób, że liczby o wartości i wpadają do kubełka z indeksem i. Żeby otrzymać posortowany ciąg wystarczy teraz przejrzeć kubełki w kolejności rosnących indeksów i wypisać ich zawartości. Taki algorytm działa w czasie $O(n+c)$. Jeśli c jest rzędu co najwyżej n, to otrzymujemy algorytm sortowania działający w czasie liniowym. Niektórzy mogą poczuć się zaskoczeni, przypominając sobie, że każdy algorytm sortowania wymaga czasu rzędu co najmniej $n\log n$ [3, 8]. Trzeba jednak pamiętać, że ta dolna granica obowiązuje dla sortowania, w którym informację o względnym porządku elementów uzyskuje się tylko porównując porządkowane elementy między sobą. W sortowaniu kubełkowym elementy nie są ze sobą porównywane i wykorzystuje się wiedzę na temat natury ich wartości. W szczególności to, że te wartości mogą być indeksami tablicy (kubełków). Jeśli zakres tych indeksów jest nieduży, to otrzymujemy bardzo szybki algorytm sortowania, który może być alternatywą dla znanych, najszybszych algorytmów sortowania z użyciem porównań.

Jednym z najważniejszych praktycznych zastosowań kubełków jest **haszowanie** [3], wykorzystywane w bazach danych do szybkiego wyszukiwania interesujących nas danych. Dane przechowujemy w kubełkach poindeksowanych od 0 do $c-1$, dla pewnego parametru c. Z każdymi danymi związany jest klucz jednoznacznie je identyfikujący. Dane o kluczu k przechowywane są w kubełku o indeksie $h(k)$, gdzie h jest funkcją odwzorowującą klucze w przedział indeksów $[0, c-1]$. Jeśli funkcję h dobierzemy w taki sposób, że jest ona szybko obliczalna i równomiernie rozrzuca dane po kubełkach, to dla dobrze dobranego parametru c do każdego kubełka wpadnie niewiele danych, co gwarantuje, że każde interesujące nas dane można odzyskać w czasie stałym, czyli niezależnym od liczby elementów w bazie danych.

W tym rozdziale idea kubełków jest wykorzystana w efektywnej implementacji algorytmów dla trzech problemów, opisanych kolejno w punktach 2, 3 i 4. Zaproponowane struktury danych są proste koncepcyjnie, łatwe w implemen-

tacji, dobrze zachowują się w praktyce i są alternatywą dla struktur danych polegających na (często niełatwej) adaptacji zrównoważonych drzew wyszukiwań binarnych lub wykorzystaniu złożonych implementacji kolejek priorytetowych. Oprócz przedstawienia konkretnych rozwiązań, które mogą znaleźć zastosowania także w innych problemach, proponujemy też pewną metodologię atakowania problemów algorytmicznych, która polega na zapisania algorytmu w sposób jak najbardziej abstrakcyjny, operując takimi pojęciami jak zbiór, ciąg, funkcja itp., a dopiero potem na podawaniu szczegółów implementacyjnych odnoszących się do używanych obiektów abstrakcyjnych i wykonywanych na nich operacjach.

2. Kodowanie permutacji

Nasze rozważania rozpoczniemy od rozwiązania nietrudnego, ale użytecznego zadania, które to rozwiązanie wprowadzi nas w świat kubełkowych struktur danych.

Zadanie 2.1. Kodowanie permutacji

Każdą permutację $A = [0..n-1]$ liczb 0, 1,..., $n-1$ można zakodować za pomocą tablicy $B = [0..n-1]$, w której dla każdego $i = 0, 1,..., n-1$ $B[i]$ jest równe liczbie wszystkich takich j, że $0 \le j < i$ oraz $A[j] > A[i]$. Należy zaprojektować algorytm, który dla danej permutacji A znajduje jej kod B.

Przykład 2.1.

Dla $n = 10$ i $A = [2, 6, 0, 9, 7, 3, 1, 5, 4, 8]$ kodem permutacji A jest tablica $B = [0, 0, 2, 0, 1, 3, 5, 3, 4, 1]$.

W literaturze kombinatorycznej kod B jest nazywany **wektorem inwersji permutacji** A [6]. Suma elementów wektora inwersji mówi, ile jest wszystkich nieuporządkowanych (rosnąco) par elementów w permutacji, którą ten wektor koduje. Taka liczba jest pewną miarą uporządkowania tablicy A i wiadomo na przykład, że algorytm sortowania przez wstawianie (w implementacji ze strażnikiem) wykonuje $n-1+\sum B[i]$ porównań, żeby posortować rosnąco permutację A. A zatem, im ciąg ma mniej inwersji, tym szybciej algorytm sortowania przez wstawianie poradzi sobie z jego uporządkowaniem.

Wymyślenie algorytmu dla naszego zadania nie jest specjalnie trudne. Jest on ukryty w samej definicji problemu. Zapiszmy go jednak. Pamiętajmy przy tym, żeby pierwszy algorytm był jak najbardziej abstrakcyjny, operował pojęciami matematycznymi takimi jak zbiór, ciąg, funkcja oraz operacjami na tych obiektach. Przy pierwszym podejściu nie myślmy nad szczegółami implementacyjnymi. Im ogólniejszy zapis algorytmu, tym większa potem swoboda w doborze tych szczegółów. Oto pierwszy algorytm:

```
Algorytm Kodowanie_1
    for i := 0 to n-1 do
        B[i] := liczba takich j, że 0 ≤ j < i orazA[j]>A[i];  (**)
```

Bardzo łatwo zaimplementować ten algorytm wykonując operację z wnętrza pętli **for** w następujący sposób: przeglądamy po kolei elementy $A[0], A[1],..., A[i-1]$ i każdy z nich porównujemy z $A[i]$, zliczając te $A[j]$, które są większe od $A[i]$:

```
licz := 0;
e := A[i];
for j := 0 to i-1 do
    if (A[j] > e) then
        licz := licz + 1;
    B[i] := licz;
```

Wyznaczenie wartości $B[i]$ wymaga i porównań elementu $A[i]$ z elementami go poprzedzającymi. Zatem łączna liczba porównań konieczna do wyznaczenia kodu B zaproponowanym algorytmem wynosi $\sum_{i=0}^{n-1} i = n(n-1)/2$, niezależnie od zawartości tablicy A. Algorytm ten jest algorytmem kwadratowym, czyli o złożoności $O(n^2)$.

Spróbujmy teraz znaleźć szybsze rozwiązanie. W takim przypadku zazwyczaj mamy dwie możliwości. Możemy zastanowić się nad zupełnie nowym algorytmem albo przyśpieszyć działanie algorytmu, który znamy, poprzez odpowiedni dobór struktur danych. Zastosujemy to drugie podejście. Ponieważ ten rozdział adresujemy do początkujących algorytmików, nasze propozycje nie będą najbardziej optymalne, ale ich zaletą będzie prostota oraz stosunkowo niezłe zachowanie się w praktyce.

Spróbujmy wyrazić operację (**) trochę inaczej. Przez $S_i = \{A[0],..., A[i-1]\}$ oznaczmy zbiór elementów permutacji A z pozycji $0,..., i-1$. Zbiór S_i jest i-elementowym podzbiorem zbioru $S = \{0,..., n-1\}$, ale nie zawiera $A[i]$. Każdy podzbiór $S' \subseteq S$ można reprezentować jako zero-jedynkową tablicę $V[0..n-1]$ taką, że $V[i] = 1$, jeśli tylko $i \in S'$, natomiast $V[i] = 0$, gdy $i \notin S'$.

Przykład 2.2.

Dla $n = 10$ i $S' = \{1, 3, 6, 7, 9\}$ mamy $V = [0, 1, 0, 1, 0, 0, 1, 1, 0, 1]$.

Jeśli V reprezentuje zbiór S_{i-1}, to $B[i]$ jest po prostu równe liczbie jedynek w tablicy V na prawo od pozycji o indeksie równym $A[i]$, czyli znajdujących się w V na pozycjach $A[i] + 1, A[i] + 2,..., n-1$. Przejście od reprezentacji zbioru S_{i-1} do reprezentacji zbioru S_i polega po prostu na ustawieniu jedynki w tablicy V na pozycji $A[i]$, tzn. $V[A[i]] := 1$. Zauważmy jeszcze, że S_0 jest zbiorem pustym, a jego reprezentacją jest $V = [0, 0,..., 0]$. Nasz algorytm można teraz przepisać następująco:

```
Algorytm Kodowanie_2
    {* inicjowanie pustego zbioru S_0 *}
    for i := 0 to n - 1 do
        V[i] := 0;
    (* obliczanie kodu B *)
    for i := 0 to n - 1 do
        B[i] := Jedynki(A[i]);
        Ustaw(A[i]);
```

W tym algorytmie funkcja `Jedynki(k)` oblicza liczbę jedynek w tablicy V na pozycjach o indeksach większych od k, natomiast procedura `Ustaw(k)` ustawia jedynkę w tablicy V na pozycji k-tej. Jeśli zliczanie jedynek zaimplementujemy w naiwny sposób polegający na przeglądaniu wszystkich pozycji o indeksach większych od k, to złożoność algorytmu nadal pozostanie kwadratowa. Pokażemy teraz, w jaki sposób znacząco przyśpieszyć obliczenia, stosując chwyt, którego opis i zastosowania są przedmiotem rozważań w tym rozdziale.

Niech m będzie dodatnią liczbą całkowitą nie większą od n i niech $s = \lfloor (n-1)/m \rfloor + 1$. Zauważmy, że jeśli m dzieli całkowicie n, to $s = n/m$, a w przeciwnym razie $s = \lceil n/m \rceil$. Na przykład dla $n = 10$ i $m = 2$ mamy $s = 5$, natomiast dla $n = 10$ i $m = 3$ mamy $s = 4$.

Aby przyspieszyć algorytm, podzielimy tablicę V na s rozłącznych bloków $V_0, V_1, \ldots, V_{s-1}$, gdzie $V_i = V[im..\min((i+1)m - 1, n)]$. W naszym przykładzie dla $n = 10$ i $m = 3$ mamy $V_0 = V[0..2]$, $V_1 = V[3..5]$, $V_2 = V[6..8]$, $V_3 = V[9..9]$. Zauważmy, że dla $i < s - 1$ każdy blok V_i ma długość m, natomiast długość bloku V_{i-1} wynosi $n - (s-1)m$ i jest nie większa niż m. Wprowadźmy dodatkową tablicę $K[0..s-1]$, która będzie zawierała informacje o liczbach jedynek w blokach V_i. To znaczy chcemy, żeby $K[i]$ było równe liczbie jedynek w bloku V_i. Dla $n = 10$, $m = 3$ i $V = [0, 1, 0, 1, 0, 0, 1, 1, 0, 1]$, mamy $K = [1, 1, 2, 1]$. Korzystając z tablicy K, wartość funkcji `Jedynki(j)` można teraz obliczyć następująco. Najpierw obliczamy indeks bloku zawierającego pozycję j. Łatwo przekonać się, że wynosi on $l = \lfloor j/m \rfloor$. Teraz sumując wartości $K[l+1], \ldots, K[s-1]$, otrzymujemy liczbę wszystkich jedynek w blokach na prawo od bloku l, czyli bloku zawierającego pozycję j. Niech tą liczbą będzie x. To, co nam jeszcze pozostaje, to obliczyć liczbę jedynek na prawo od j w bloku V_l. W tym celu przeglądamy kolejne pozycje $j+1, j+2, \ldots, \min((l+1)m - 1, n)$. Takich pozycji jest mniej niż m.

Zastanówmy się teraz, ile nas kosztuje obliczenie wartości `Jedynki(j)`. Sumowanie wartości z tablicy K zabiera mniej niż s dodawań, natomiast zliczanie jedynek w bloku V_i wymaga przejrzenia mniej niż m elementów w tym bloku. Zatem łączny koszt wykonania algorytmu jest rzędu co najwyżej $O(m + s)$. Zapytajmy teraz, w jaki sposób dobrać m, żeby osiągnąć jak najlepszy efekt złożonościowy? Ponieważ $s = \lfloor (n-1)/m \rfloor + 1$, to widać, że najszybszy algorytm dosta-

niemy wtedy, gdy m i s są mniej więcej takie same. Zatem za m najlepiej wziąć $\lfloor\sqrt{n}\rfloor$, a wówczas funkcja `Jedynki` jest wykonywana w czasie $O(\sqrt{n})$.

Procedurę `Ustaw(j)` zaimplementować bardzo łatwo. Wystarczy ustawić j-tą pozycję w tablicy V na 1 oraz zwiększyć liczbę jedynek w bloku zawierającym j o 1, $K[\lfloor j/m \rfloor] := K[\lfloor j/m \rfloor] + 1$.

Podsumowując, wykonujemy n operacji obliczania liczby jedynek i n operacji ustawienia nowej jedynki, zatem obliczanie kodu permutacji można wykonać w czasie $O\left(n\sqrt{n}\right)$, czyli zdecydowanie szybciej niż w czasie $O(n^2)$. Dla przykładu, jeśli przyjmiemy realistycznie, że stałe ukryte w notacji $O(.)$ wynoszą odpowiednio 2 i 1/2, to dla $n = 1\,000\,000$, $2n\sqrt{n} = 2\,000\,000\,000$, natomiast $n^2/2 = 1\,000\,000\,000\,000/2 = 500\,000\,000\,000 = 250*2\,000\,000\,000$.

Pozostaje formalnie zapisać poszczególne operacje. Niech $m = \lfloor\sqrt{n}\rfloor$, $s = \lfloor (n-1)/m \rfloor + 1$. Poniżej przedstawiamy zapis omówionych właśnie funkcji. Pamiętajmy tylko, że tablica K musi być wcześniej zainicjowana zerami.

```
Algorytm Jedynki(j)
    l := ⌊j / m⌋;
    x := 0;
    for i := l + 1 to s - 1 do
        x := x + K[i];
    for i := j + 1 to min((l + 1) * m - 1, n) do
        if V[i] = 1 then x := x + 1;
    return x;

Algorytm Ustaw(j)
    V[j] := 1; l := ⌊j / m⌋;
    K[l] := K[l] + 1 ;
```

3. Odwracanie

W tym punkcie zajmiemy się rozwiązaniem zadania często spotykanego w praktyce. Operujemy na zbiorze danych, które podlegają różnorodnym modyfikacjom. Należy tak zorganizować ten zbiór, żeby móc szybko odpowiadać na pytania dotyczące zawartych w nim elementów. Nasze kolejne zadanie będzie właśnie takiego typu.

Zadanie 3.1. Odwracanie bloków

Dana jest dodatnia liczba całkowita n i ciąg elementów $C = [c_1, c_2, ..., c_n]$ dowolnego, ale ustalonego typu. Dla ustalenia uwagi przyjmijmy, że elementy ciągu C to małe litery alfabetu angielskiego. Blokiem nazwiemy każdy podciąg ciągu C złożony z kolejnych elementów. Na ciągu C wykonujemy następujące operacje:

1. `Odwróć(i,j)` – odwróć kolejność elementów w ciągu C na pozycjach od i do j, dla $1 \leq i \leq j \leq n$;
2. `El(j)` – podaj wartość j-tego elementu w aktualnym ciągu C, gdzie $1 \leq j \leq n$.

Należy zaprojektować strukturę danych, która służyć będzie do szybkiego wykonania m operacji powyższego typu na dynamicznym ciągu C. Zauważmy, że z punktu widzenia użytkownika ważne są pytania o elementy ciągu. Tak więc operacji `Odwróć` nie musimy fizycznie wykonywać, ważne jest tylko, żeby odpowiedzi na pytania o elementy ciągu były takie, jak byśmy wykonali wszystkie poprzedzające je odwrócenia.

Przykład 3.1.

Załóżmy, że na początku ciąg C ma postać [a, b, c, d, e, f, g, h], $n = 8$ i na C wykonujemy kolejno operacje: `Odwróć(2,4), El(2), El(5), Odwróć(4,7), El(5)`. Wówczas odpowiedziami dla kolejnych wywołań funkcji `El` są odpowiednio d, e, f.

Dla dalszych rozważań przyjmijmy, że przetwarzany ciąg jest zapisany w tablicy $C[1..n]$. Pierwsze narzucające się rozwiązanie jest oczywiste. Każde wywołanie operacji `Odwróć(i,j)` pociąga za sobą fizyczne odwrócenie zawartości podtablicy $C[i..j]$. Ponieważ odwrócenie kolejności elementów w podciągu o długości $j - i + 1$ pociąga za sobą $\lfloor (j - i + 1)/2 \rfloor$ zamian elementów parami, to w pesymistycznym przypadku wykonanie całego ciągu m operacji zabrałoby czas $O(mn)$.

Nasze rozwiązanie będzie polegało na odwlekaniu fizycznej reorganizacji tablicy C – wykonaniu fizycznych odwróceń – tak długo, jak to tylko możliwe. Czy rzeczywiście, żeby odpowiadać na pytanie o elementy ciągu C musimy wykonać fizycznie wszystkie wcześniejsze odwrócenia? Gdyby na przykład przedziały, w których dokonujemy odwrócenia były parami rozłączne, wystarczyłoby tablicę C podzielić na spójne bloki, które odpowiadają maksymalnym fragmentom, które są w całości odwrócone lub w całości nieodwrócone, i dla każdego bloku mieć znacznik informujący o jego statusie: odwrócony, nieodwrócony. Kiedy pytamy o k-ty element ciągu, który wpada do bloku o indeksach od i do j, odpowiedzią jest element $C[k]$, gdy blok nie jest odwrócony, natomiast element $C[j - (k - i)]$, gdy blok został odwrócony. Niestety odwrócenia mogą być wykonywane na blokach, które nie są rozłączne. Szczęśliwie, idee opisane wcześniej znajdują zastosowanie w ogólnym przypadku.

Zawartość tablicy C po wykonaniu pewnej liczby operacji `Odwróć` będziemy reprezentowali jako ciąg parami rozłącznych, pokrywających ją bloków $B_1, B_2, \ldots, B_k$, dla pewnego $k \geq 1$. Każdy blok jest zadany jako trójka (a, b, *odwr*), gdzie a jest indeksem początku bloku, b indeksem jego końca, natomiast *odwr* jest logicznym wskaźnikiem informującym o tym, czy elementy w bloku są odwrócone (*odwr* = *TRUE*), czy też nie (*odwr* = *FALSE*). Żeby odczytać ciąg w aktualnej postaci,

należy przejść kolejno po wszystkich blokach, a w każdym bloku odczytywać elementy w C z lewa na prawo, gdy blok nie jest odwrócony, natomiast z prawa na lewo, gdy jest odwrócony. Na samym początku mamy tylko jeden blok o parametrach (1, n, *FALSE*).

Przykład 3.2.

Niech $C = [a, b, c, d, e, f, g, h]$ i niech (5, 7, T), (2, 4, F), (8, 8, T), (1, 1, T) będzie ciągiem czterech bloków. Zawartość tablicy C po fizycznym wykonaniu odwróceń miałaby postać $[g, f, e, b, c, d, h, a]$.

Zacznijmy od operacji `El(j)`. Oznaczmy przez s_i rozmiar i-tego bloku. Oczywiście $s_i = b_i - a_i + 1$. Żeby wyznaczyć j-ty element w aktualnym ciągu, przeglądamy bloki po kolei w poszukiwaniu pierwszego takiego, dla którego suma rozmiarów wszystkich bloków go poprzedzających oraz jego własnego rozmiaru wynosi co najmniej j. Niech tym blokiem będzie B_i i niech suma rozmiarów wszystkich bloków go poprzedzających wynosi s. Wówczas poszukiwanym elementem jest element na pozycji $j - s$ w tym bloku, gdy blok nie jest odwrócony. W niezmienionej tablicy C ten element znajduje się na pozycji $a_i + j - s - 1$. Jeśli blok jest odwrócony, to poszukiwanym elementem jest ten z pozycji $s_i + 1 - (j - s)$ w bloku B_i, czyli $C[a_i + s_i - (j - s)]$. Oto formalny zapis funkcji `El`.

```
Algorytm El(j)
    s := 0;
    i := 1;
    while (s + s_i < j) do
        i := i + 1;
        if (odwr_i) then
            return C[a_i + s_i - (j - s)]
        else
            return C[a_i + j - s - 1];
```

Zauważmy, że koszt pesymistyczny tego algorytmu jest rzędu liczby wszystkich bloków k.

Trochę trudniejsza do wykonania jest operacja `Odwróć(i,j)`. Jak się za chwilę okaże, najbardziej problematyczny jest przypadek, gdy fragment, który chcemy odwrócić, wpada do jednego bloku. Jeśli jest to cały blok, to wystarczy tylko zmienić jego wskaźnik odwrócenia na przeciwny. Załóżmy zatem, że ten fragment jest podblokiem pewnego bloku B zadanego przez trójkę (a, b, *odwr*). Najpierw obliczamy początek c i koniec d podbloku bloku B, który odpowiada przedziałowi $[i, j]$ i którego elementy chcemy odwrócić.

Pozycje c i d obliczamy w sposób opisany w funkcji `El`. Teraz blok B dzielimy na (co najwyżej) trzy bloki $X = B[a..c - 1]$, $Y = B[c..d]$, $Z = B[d + 1..b]$. Przedziały

X i Z mogą być puste i wtedy pomijamy je w naszych rozważaniach. Wskaźnikom odwrócenia bloków X i Z nadajemy taką samą wartość, jaką ma wskaźnik bloku B, natomiast wskaźnik odwrócenia bloku Y przyjmuje wartość przeciwną. Po dokonaniu tych czynności w ciągu bloków $B_1, B_2,..., B_k$, w miejsce bloku B, wstawiamy kolejno (tylko niepuste) bloki X, Y, Z, gdy wartość wskaźnika odwrócenia bloku B wynosi *FALSE*, natomiast Z, Y, X, gdy tą wartością jest *TRUE*.

A co, gdy przedział $[i, j]$ zawiera więcej niż jeden blok? Niech $B_l, B_{l+1},..., B_p$ będą tymi blokami. Niech c będzie pozycją w bloku B_l odpowiadającą początkowi przedziału, który chcemy odwrócić, czyli i. W zależności od wskaźnika odwrócenia $odwr_l$ dla bloku B_l dzielimy ten blok na dwa podbloki $B_l^{(1)}$ i $B_l^{(2)}$ w następujący sposób:

- jeśli $odwr_l = FALSE$, to $B_l^{(1)} = B[a..c-1]$, $B_l^{(2)} = B[c..b]$ i dla obu bloków wskaźniki odwrócenia ustawiamy na $FALSE$;
- jeśli $odwr_l = TRUE$, to $B_l^{(1)} = B[c+1..b]$, $B_l^{(2)} = B[a..c]$ i dla obu bloków wskaźniki odwrócenia ustawiamy na $TRUE$.

Jeśli teraz w miejsce bloku B_l wstawimy bloki $B_l^{(1)}$ i $B_l^{(2)}$ (tylko niepuste), to możemy dalej pracować przy założeniu, że blok B_l (teraz jest to blok $B_l^{(2)}$) całkowicie wpada w przedział $[i, j]$. Podobnie możemy postąpić z blokiem B_p. Zostawiamy to jako ćwiczenie dla Czytelnika.

W tym momencie możemy założyć, że cały przedział $[i, j]$ to tak naprawdę suma bloków $B_l,..., B_p$, być może z odwróconymi kolejnościami wystąpień elementów w pewnych z nich. Aktualizacja struktury jest teraz bardzo prosta. W miejsce ciągu $B_l, B_{l+1},..., B_p$ w naszej strukturze wstawiamy ciąg $B_p, B_{p-1},..., B_l$ pamiętając przy tym, żeby wskaźnik odwrócenia każdego bloku w tym ciągu zamienić na przeciwny. Nietrudno zauważyć, że koszt operacji aktualizacji naszej struktury jest rzędu co najwyżej liczby bloków w niej zawartych, czyli wynosi $O(k)$.

Pokazaliśmy, że obie operacje `Odwróć` i `El` w pesymistycznym przypadku wykonują się w czasie proporcjonalnym do liczby bloków w strukturze. Duża liczba operacji odwracania może doprowadzić do dużej liczby bloków, w pesymistycznym przypadku nawet liniowej ze względu na n. Dlatego warto dbać o to, żeby liczba bloków nie była za duża. Przyjmiemy, że gdy ta liczba przekroczy $\sqrt{n}$, to odtworzymy aktualny ciąg C na podstawie zawartości bloków. To nie zajmie więcej niż czas liniowy i znowu będziemy mogli wystartować z jednym blokiem z parametrami $(1, n, FALSE)$. Zatem koszt wykonania ciągu m operacji wynosi co najwyżej $O(m\sqrt{n})$. Pomijamy tu koszt inicjacji danych, który nie przekracza $O(n)$ – zainicjowanie tablicy C i jednoelementowej struktury bloków.

Zadanie omawiane w tym rozdziale pochodzi z zawodów programistycznych CERC 2007 [9], a autorem pomysłu przedstawionego rozwiązania jest reprezentant Uniwersytetu Warszawskiego na tych zawodach Piotr Niedźwiedź.

4. Algorytm Dijkstry

Edsger W. Dijkstra (1930-2002), światowej sławy holenderski informatyk. Jak sam wspominał, algorytm dla problemu najkrótszych ścieżek z jednym źródłem wymyślił w 20 minut podczas zakupów w centrum handlowym w Amsterdamie w roku 1956. Publikacja zawierająca opis algorytmu ukazała się trzy lata później: Dijkstra, E. W., *A note on two problems in connexion with graphs*, **„Num. Math." 1(1959), 269-271. Sam problem pojawił się jako ilustracja zastosowania komputera ARMAC dla niespecjalistów. Dijkstra pokazywał, w jaki sposób najszybciej przejechać z Rotterdamu to Groningen. Na owe czasy takie obliczenia z użyciem komputera były prawdziwym wyzwaniem. Warto tutaj dodać, że w tej samej pracy Dijkstra zamieścił również opis algorytmu dla problemu najkrótszego drzewa rozpinającego, bazujący na podobnej idei zachłanności, w myśl której w kolejnym kroku algorytm przemieszcza się do najbliższego wierzchołka wśród jeszcze nieodwiedzonych. Warto jednak zauważyć różnice między tymi dwoma problemami i algorytmami – pozostawiamy to jako ćwiczenie dla własnych przemyśleń** [7].

W tej części zajmiemy się efektywną implementacją jednego z najsłynniejszych algorytmów – algorytmu Dijkstry dla problemu znajdowania najkrótszych ścieżek z jednym źródłem [3, 6]. Krótko sformułujemy problem i podamy dla niego abstrakcyjny algorytm. Nie będziemy się zajmować jego poprawnością, ponieważ opis algorytmu Dijkstry można znaleźć w każdym porządnym podręczniku algorytmiki. Podamy za to jego podstawowe charakterystyki, które mają wpływ na implementację, a następnie zaproponujemy struktury danych, które są nie tylko łatwe w realizacji, ale także efektywne w praktyce.

Załóżmy, że mamy dość szczegółową mapę drogową Polski. Szczegółowość oznacza tu, że odległości między sąsiednimi miejscowościami na mapie (bezpośrednio połączone drogą, nieprzechodzącą przez żadną inną miejscowość uwzględnioną na mapie) nie przekraczają powiedzmy 10 km. Naszym celem jest obliczenie długości najkrótszych tras prowadzących z Warszawy do wszystkich miejscowości uwzględnionych na mapie. Dla uproszczenia przyjmijmy, że wszystkie drogi są dwukierunkowe.

Mapę można modelować za pomocą grafu nieskierowanego, w którym wierzchołki odpowiadają miejscowościom na mapie, natomiast krawędzie bezpośrednim drogom łączącym sąsiednie miejscowości. Z każdą drogą (krawędzią) wiążemy dodatnią liczbę całkowitą równą długości tej drogi. W naszym przypadku przyjmujemy, że długości są dodatnimi liczbami całkowitymi nie większymi od zadanej, dodatniej liczby całkowitej c. Dodatkowo w grafie (na mapie)

wyróżniamy jeden wierzchołek (np. stolicę), dla którego chcemy policzyć odległości (długości najkrótszych tras) do wszystkich pozostałych wierzchołków (miejscowości). Problem, o którym mówimy, w literaturze spotyka się pod nazwą **problemu najkrótszych ścieżek** z jednym źródłem i jest jednym z najczęściej badanych algorytmicznych problemów optymalizacyjnych. Istnieją setki prac na temat rozwiązania tego problemu, a większość z nich to wariacje na temat algorytmu zaproponowanego przez wybitnego holenderskiego informatyka Edsgara Dijkstrę. Należy tutaj zaznaczyć, że w ogólnym sformułowaniu problemu, o długościach krawędzi zakłada się tylko, że są dodatnie i nie narzuca się na te długości żadnego górnego ograniczenia. Sformułujemy teraz nasz problem w oderwaniu od terminologii grafowej, ale w sposób, który będzie przydatny w naszych rozważaniach.

Zadanie 4.1. Algorytm Dijkstry

Danych jest n obiektów ponumerowanych od 1 do n. Obiekty utożsamiamy z ich numerami. Z każdym obiektem i związany jest podzbiór $N(i)$ obiektów różnych od i. Elementy zbioru $N(i)$ nazywamy **sąsiadami** obiektu i. Dla każdego obiektu-sąsiada $j \in N(i)$ znamy odległość $d_i[j]$ pomiędzy obiektami i oraz j. Każda odległość jest dodatnią liczbą całkowitą, nie większą niż zadana z góry dodatnia liczba całkowita c. Naszym zadaniem jest zaprojektowanie „szybkiego" obliczania tablicy $d[1..n]$, zgodnie z następującym algorytmem:

```
Algorytm AD
    (* Inicjacja *)
    d[1] := 0;
    for i := 2 to n do
        if (i ∈ N(1)) then
            d[i] := d1[i];
        else
            (* górne ograniczenie na długość najdłuższej trasy *)
            d[i] := c * (n - 1);
    S := {2,3,...,n};
    (* główne obliczenia *)
    for i := 1 to n - 1 do
        j := obiekt w zbiorze S z najmniejszym d[j]; (* Min *)
        S := S\{j}; (* Usuń Min *)
        for k ∈ N(j) do
            (* Zmniejsz Priorytet *)
            if (d[k] > (d[j] + dj[k])) then d[k] := d[j] + dj[k];
```

Algorytm AD jest tak naprawdę algorytmem Dijkstry zapisanym w abstrakcyjny sposób. Poniżej podajemy kilka ważnych własności tego algorytmu, w których uwzględniamy ograniczenie c na odległości. Będą one przydatne w jego wydajnej implementacji.

1. Niech $j_1, j_2, ..., j_{n-1}$ będą kolejnymi obiektami obliczanymi w wierszu (* `Min` *) i weźmy $j_0 = 1$. Wówczas $d[j_0] = 0 < d[j_1] \le d[j_2] \le ... \le d[j_{n-1}]$. Innymi słowy w algorytmie Dijkstry obliczamy odległości wierzchołków od źródła w kolejności od najbliższych do najdalszych.
2. Dla każdego $k = 1, 2, ..., n-1$, $d[j_k] - d[j_{k-1}] \le c$ – kolejny wierzchołek jest odległy od źródła o co najwyżej c dalej, niż wierzchołek go poprzedzający.
3. $d[j_{n-1}] \le c*(n-1)$ – najbardziej odległy wierzchołek jest położony nie dalej niż $c*(n-1)$.

Przyjrzyjmy się teraz, co wpływa na koszt wykonania algorytmu. Główne operacje są wykonywane na zmieniającym się zbiorze S. Każdy element j w tym zbiorze ma przypisaną liczbę całkowitą $d[j]$, którą nazwiemy **priorytetem**. Wiemy, że każdy priorytet jest liczbą całkowitą z przedziału $[0, c(n-1)]$. Na zbiorze S wykonujemy następujące operacje:

`Inicjacja::` utwórz zbiór S z obiektami 2, 3,..., n o priorytetach zdefiniowanych w części `Inicjacja` algorytmu `AD`.

`Min::` wskaż w zbiorze S obiekt z najmniejszym priorytetem; w przypadku wielu obiektów z takim samym, najmniejszym priorytetem, wskaż dowolny z nich.

`Usuń_Min::` usuń ze zbioru S obiekt wskazany w wyniku wykonania operacji `Min`.

`Zmniejsz_Priorytet(`***k,p***`)::` zmniejsz priorytet obiektu k do nowej wartości p.

W naszym algorytmie wykonujmy $n-1$ operacji `Min`, $n-1$ operacji `Usuń_Min` oraz co najwyżej m operacji `Zmniejsz_Priorytet`, gdzie m jest równe sumie rozmiarów zbiorów sąsiadów $N(i)$ (co odpowiada liczbie krawędzi w grafie). W analizie czasu działania algorytmu musimy też uwzględnić inicjację zbioru S.

W jednej z najbardziej zaawansowanych implementacji algorytmu Dijkstry wykorzystuje się kopce Fibonacciego [3] do reprezentacji zbioru S. W implementacji tej nie zakłada się ograniczenia na długości krawędzi. Algorytm Dijkstry z kopcami Fibonacciego działa w czasie $O(m + n\log n)$. Niestety stopień złożoności tego algorytmu jest tak duży, że jest on trudny w analizie i słabo zachowuje się w praktyce. Najtrudniejsze w tej implementacji jest zapewnienie takiej organizacji zbioru S, żeby operację zmniejszenia priorytetu móc wykonywać w stałym (zamortyzowanym) czasie. Z drugiej strony, w ogólnym przypadku składnika $n\log n$ nie można zmniejszyć, ponieważ algorytm Dijkstry można użyć do sortowania. To ostatnie stwierdzenie pozostawiamy do uzasadnienia Czytelnikowi.

Zapoznamy się teraz z implementacją zbioru S, która jest nie tylko prosta, ale też nieźle zachowuje się w praktyce. Więcej, jej sprytna modyfikacja znajduje zastosowania w analizie różnego rodzaju sieci. W naszych rozwiązaniach istotnie wykorzystamy fakt, że odległości są liczbami całkowitymi ograniczonymi przez stałą c.

4.1. Rozwiązanie 1

Do reprezentacji zbioru S wykorzystamy tablicę $K[0..c(n-1)]$. Elementy tablicy K nazywamy **kubełkami**. Kubełek o indeksie i będzie przechowywał wszystkie obiekty, które aktualnie znajdują się w zbiorze S i których priorytety są równe i. Każdy kubełek reprezentujemy jako listę dwukierunkową po to, żeby nowe obiekty można wrzucać do kubełków w czasie stałym oraz by usuwać z nich w czasie stałym wskazane (na liście) obiekty. Dodatkowo zakładamy, że dana jest tablica $G[1..n]$ taka, że dla każdego obiektu $i \in S$, $G[i]$ wskazuje miejsce wystąpienia obiektu i (na liście) w kubełku $K[d[i]]$. Umożliwia to usuwanie w czasie stałym obiektu z zawierającego go kubełka.

Zastanówmy się teraz, w jaki sposób zaimplementować poszczególne operacje.

`Inicjacja::`

Wszystkie kubełki inicjujemy jako puste. Następnie każdego sąsiada j wierzchołka 1 wrzucamy do kubełka $K[d_1[j]]$, natomiast wszystkie pozostałe wierzchołki wrzucamy do kubełka o numerze $c(n-1)$. Wprowadzamy dodatkową zmienną *ost*, która będzie wskazywała, który z kubełków był oglądany jako ostatni. Zmiennej *ost* na początku nadajemy wartość 0, ponieważ do kubełka o numerze 0 wpadłby wierzchołek 1, choć tego nie czynimy. Koszt inicjacji wynosi zatem $O(cn)$.

`Min::`

Operacja `Min` polega na znalezieniu pierwszego niepustego pudełka z lewej strony, poczynając od pudełka $K[ost]$, i wskazaniu w nim jednego obiektu, np. pierwszego na liście obiektów umieszczonych w tym pudełku. Własność 1 gwarantuje, że do pominiętych kubełków nigdy nie będziemy wracać. Ponadto wskazany obiekt z najmniejszym priorytetem zostanie za chwilę usunięty ze zbioru S, a tym samym z kubełków, i nigdy do nich nie wróci.

`Usuń_Min::`

Operacja ta usuwa obiekt wskazany przez `Min`. Po prostu należy usunąć pierwszy obiekt z listy obiektów w kubełku $K[ost]$. W oczywisty sposób tę operację można wykonać w czasie stałym.

Możemy już podsumować łączny koszt wykonania operacji `Min` i `Usuń_Min`. Wynosi on $O(cn)$. Każda operacja `Usuń_Min` zabiera czas stały. Obiekt usunięty nigdy nie wraca do kubełków. W każdym kubełku spędzamy czas proporcjonalny do liczby zawartych w nim obiektów plus jeden. Ponieważ w poszukiwaniu niepustych kubełków przesuwamy się zawsze w prawo, to łączny koszt wykonania obu operacji jest równy $O(cn)$.

`Zmniejsz_Priorytet(k,p)::`

Operacja ta jest niesłychanie prosta. Wystarczy usunąć obiekt k z kubełka $K[d[k]]$ i przesunąć go do kubełka $K[p]$, zmieniając jednocześnie wartość $d[k]$

na p. Koszt wykonania tej operacji jest stały, a wykonanie m takich operacji zajmuje czas $O(m)$.

Podsumowując, łączny koszt wykonania algorytmu w implementacji rozwiązania 1 wynosi $O(m + cn)$. Algorytm w tej postaci jest znany pod nazwą algorytmu Diala [5]. To rozwiązanie dla małych c jest dość szybkie. Jego główną wadą jest tablica kubełków, która przy dużym, ale ciągle rozsądnym c, może być nieakceptowalna ze względu na swój rozmiar. Pokażemy teraz, w jaki sposób znacząco zmniejszyć rozmiar pamięci potrzebnej do implementacji zbioru S [5].

4.2. Rozwiązanie 2

Dla pokazania, że do implementacji kubełków nie jest do niczego potrzebna tak duża pamięć, wykorzystamy drugą własność algorytmu. Wiemy, że jeśli $K[ost]$ jest ostatnio oglądanym, niepustym kubełkiem, to następny niepusty kubełek będzie na pozycji o numerze co najwyżej $ost + c$. Więcej, jeśli obiekt k jest usuwany z $K[ost]$, to po wykonaniu operacji zmniejszenia priorytetów, wszyscy jego sąsiedzi ze zbioru S znajdą się także w kubełkach o numerach od ost do $ost + c$. Te fakty wykorzystamy do implementacji naszego algorytmu przy użyciu tylko $c + 1$ kubełków $K[0..c]$, po których będziemy wędrować cyklicznie. Jeśli ost jest indeksem ostatnio odwiedzanego, niepustego kubełka, to możemy przyjąć, że wszystkie obiekty ze zbioru S, które sąsiadują z jakimś obiektem spoza S, znajdują się już w kubełkach o numerach ost, $ost + 1 \bmod (c + 1)$,..., $ost + c \bmod (c + 1)$. Ponadto wiemy, że jeśli priorytety obiektów w kubełku $K[ost]$ są równe p, to w kolejnych (cyklicznie) kubełkach znajdują się obiekty o priorytetach odpowiednio $p + 1$, $p + 2$,..., $p + c$. Pozostaje jeszcze jeden mały problem do rozwiązania. W pierwszym podejściu, podczas inicjacji do kubełków wrzucono wszystkie obiekty. Teraz chcemy, żeby w kubełkach były tylko obiekty sąsiadujące z tymi spoza S. W nowym podejściu obiekt będzie się pojawiał w kubełkach dopiero wtedy, gdy wykryjemy po raz pierwszy, że sąsiaduje on z obiektem usuwanym z kubełków w wyniku operacji `Usuń_Min`. Musimy tylko zaznaczyć w jakiś sposób, że obiekt jest w S, ale poza kubełkami. Do tego celu wykorzystamy tablicę odległości d. Wartość – 1 w tablicy będzie oznaczała, że odpowiadający jej obiekt jest poza kubełkami. Oto zapis zaproponowanego rozwiązania.

```
Algorytm AD w małej pamięci
    (* Inicjacja *)
    zainicjuj kubełki K[0..c] jako puste;
    d[1] := 0;
    ost := 0;
    for i:= 2 to n do
        if (i ∈ N(1)) then
            d[i] := d1[i];
            umieść obiekt i w kubełku K[d[i]];
        else
```

```
        d[i] := -1;    (* d[i] = -1 oznacza obiekt zbioru S
                          nieumieszczony jeszcze w żadnym kubełku *)
(* główne obliczenia *)
for i := 1 to n - 1 do
    (* cykliczne poszukiwanie pierwszego niepustego kubełka *)
    while (kubełek K[ost] jest pusty) do
        ost := (ost+1) mod (c+1);
    (* Min *)
    j := obiekt z kubełka K[ost];
    (* Usuń Min *)
    usuń obiekt j z kubełka K[ost];
    for k ∈ N(j) do
        if (d[k] = -1) then
            umieść k w kubełku K[(ost+d_j[k]) mod (c+1)];
        else
            if (d[k] > (d[j] + d_j [k])) then
                (* zmniejsz priorytet *)
                usuń obiekt k zkubełka K[(ost+(d[k]-d[j])) mod (c+1)];
                d[k] := d[j] + d_j[k];
                umieść obiekt k w kubełku K[(ost+d_j[k]) mod (c+1)];
```

W ten sposób zmniejszyliśmy liczbę kubełków z $cn - 1$ do $c + 1$, zachowując (asymptotyczny) czas działania samego algorytmu. W następnym kroku pokażemy, w jaki sposób przyśpieszyć sam algorytm.

4.3. Rozwiązanie 3

Wróćmy do rozwiązania 1. Pokażemy teraz, że można przyśpieszyć opisany w nim algorytm, wykorzystując chwyt z zadania o kodowaniu permutacji. Dla prostoty opisu przyjmijmy, że liczba c jest kwadratem liczby naturalnej a, czyli $c = aa$. Niech $s = an$. Podzielmy kubełki $K[0..cn - 1]$ na s bloków $K_0, K_1,..., K_{s-1}$, każdy o długości a, gdzie $K_i = K[ia..ia + a - 1]$. Z każdym blokiem K_i wiążemy superkubełek S_i, który będzie służył do przechowywania obiektów z odległościami $d[i]$ wpadającymi do przedziału związanego z tym kubełkiem, czyli $[ia, ia + a - 1]$. W poszukiwaniu obiektu z najmniejszą odległością d przeglądamy superkubełki z lewa na prawo, aż znajdziemy pierwszy niepusty superkubełek. Przyjmijmy, że jest to kubełek S_i. Po znalezieniu niepustego superkubełka S_i, wszystkie zawarte w nim obiekty umieszczamy w kubełkach drugiego poziomu $L[0..a - 1]$, które w tym celu inicjujemy jako puste. W tym przypadku kubełek $L[0]$ służy do przechowywania obiektów z priorytetami równymi ia, kubełek $L[1]$ zawiera obiekty z priorytetami równymi $ia + 1$ itd. Teraz poszukiwania obiektów z najmniejszymi priorytetami dokonujemy w kubełkach L, aż do ich wyczerpania. Nowe obiekty wstawiamy albo do kubełków z L, jeśli związane z nimi odległości są z przedziału $[ia, ia + a - 1]$, albo do superkubełków z numerami większymi od i, gdy te odległości są większe od $ia + a - 1$. Podobnie postępujemy z obiektami, dla których wykonujemy operację zmniejszenia priorytetów. Zauważmy, że kubełki pomocnicze L są przetwarzane co najwyżej

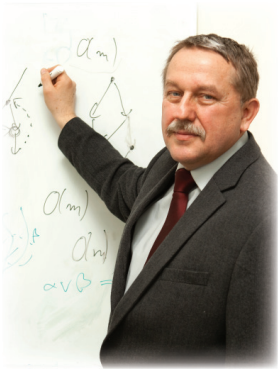
n razy – używamy ich tylko wtedy, gdy superkubełek jest niepusty, a to może się zdarzyć nie więcej niż n razy, bo tyle jest wszystkich obiektów. Zatem łączny koszt wykonania algorytmu przy użyciu dwupoziomowej struktury kubełków wynosi $O(m + na)$. Przeglądamy na superkubełków. Jeśli superkubełek jest pusty, to przechodzimy do następnego superkubełka. Jeśli superkubełek nie jest pusty, to inicjujemy kubełki L jako puste, umieszczamy w nich obiekty z superkubełka i przeglądamy kubełki L po kolei w poszukiwaniu obiektu z najmniejszym priorytetem. Kubełki L przetwarzamy nie więcej niż n razy, a czas potrzebny na ich przetworzenie jest rzędu an plus $n - 1$, bo $n - 1$ razy obiekty zostaną usunięte z kubełków L. Operację `Zmniejsz_Priorytet` wykonujemy w czasie stałym, podobnie jak w rozwiązaniu 1. Należy tylko uwzględnić dwupoziomową strukturę kubełków. Zauważmy na koniec, że możemy użyć tylko $O(a)$ kubełków, jeśli zastosujemy metodę z rozwiązania 2. Podsumowując, przedstawiliśmy implementację algorytmu Dijkstry działającą w czasie $O\left(m + n\sqrt{c}\right)$ i przy użyciu tylko $O\left(\sqrt{c}\right)$ kubełków. Ta implementacja pochodzi od Denardo i Foxa [4]. Dalsze rozwijanie opisanych tu pomysłów oraz zastosowanie wielopoziomowej struktury kubełków prowadzi do implementacji algorytmu Dijkstry działającej w czasie $O(m + n\log c)$ i przy użyciu tylko $O(\log c)$ kubełków [1].

5. Podsumowanie

W tym rozdziale przedstawiliśmy rozwiązania trzech problemów algorytmicznych z wykorzystaniem struktur kubełkowych. Dla każdego z tych problemów znane są asymptotycznie szybsze algorytmy, ale struktury danych w nich wykorzystywane są znacznie bardziej złożone, zarówno jeśli chodzi o ich implementację, jak i analizę. W przypadku kodowania permutacji są to wzbogacone drzewa wyszukiwań binarnych lub drzewa przedziałowe [3, 6], w przypadku zadania odwracanie – są to drzewa typu „splay" [2], a przypadku algorytmu Dijkstry – znajdują zastosowanie kopce Fibonacciego [3]. Każda z tych struktur danych wymagałaby odrębnego artykułu na jej opisanie. Co więcej, analiza ich zachowania wymaga dość złożonych technik analizy algorytmów. Struktury przedstawione w tym rozdziale wymagają tylko znajomości tablic oraz list, z którymi to strukturami ma szanse zapoznać się każdy początkujący programista. Mamy nadzieję, że metody projektowania algorytmów zaproponowane tutaj okażą się Czytelnikowi przydatne.

Literatura

1. Ahuja R.K., Mehlhorn K., Orlin J.B., Tarjan R.E., *Faster Algorithms for the Shortest Path Problem*, „J. ACM" 37(1990), 213-223
2. Banachowski L., Diks K., Rytter W., *Algorytmy i struktury danych*, WNT, Warszawa 2001
3. Cormen T.H., Leiserson Ch.E., Rivest R.L., Stein C., *Wprowadzenie do algorytmów*, WN PWN, Warszawa 2012
4. Denardo E.V., Fox F.L., *Shortest-route methods: Reaching, pruning, and buckets*, „Op. Res." 27(1979), 131-186
5. Dial R., *Algorithm 360: Shortest path forest with topological ordering*, „Comm. ACM" 12(1969), 632-633
6. Diks K., Malinowski A., Rytter W., Waleń T., *Moduł Algorytmy i struktury danych*, portal wazniak.mimuw.edu.pl
7. Misa T.J., *An Interview With Edsger W. Dijkstra*, „Comm. ACM" 53(2010), 41-47
8. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
9. Zadania z CERC 2007, http://contest.felk.cvut.cz/07cerc/home/cteam06.zip, Praga 2007



autor zdjecia: Adrian Krawczyk

Prof. dr hab. Krzysztof Diks

jest pracownikiem Instytutu Informatyki Uniwersytetu Warszawskiego. Specjalizuje się w algorytmice, prowadząc przedmiot Algorytmy i struktury danych. Od roku 1994 jest związany z Olimpiadą Informatyczną, a od 1999 roku jest przewodniczącym Komitetu Głównego Olimpiady. W roku 2005 był przewodniczącym Komitetu Organizacyjnego Międzynarodowej Olimpiady Informatycznej, która odbyła się w Polsce. W roku 2012 współorganizował Finały Akademickich Mistrzostw Świata w Programowaniu Zespołowym. Jest pomysłodawcą i współorganizatorem Potyczek Algorytmicznych – najpopularniejszego konkursu algorytmicznego w Polsce. Autor wielu zadań konkursowych i popularyzator informatyki. Od roku 1999 wspólnie z prof. Janem Madeyem jest opiekunem reprezentacji Uniwersytetu Warszawskiego w konkursach programistycznych. Był współopiekunem mistrzów świata w programowaniu zespołowym w latach 2003 i 2007, oraz wicemistrzów świata z roku 2012.

diks@mimuw.edu.pl

Wojciech Śmietanka

w 2007 roku rozpoczął Jednoczesne Studia Informatyczno-Matematyczne na Uniwersytecie Warszawskim. Od początku studiów startował w zawodach programistycznych ACM ICPC zostając trzykrotnym drużynowym mistrzem Polski (2009, 2010, 2011), dwukrotnym mistrzem Europy Środkowej (2010, 2011) i wicemistrzem świata (2012). Te sukcesy, z wyjątkiem mistrzostwa z 2009 roku, odniósł wspólnie z Tomkiem Kulczyńskim i Kubą Pachockim, a nad przygotowaniami do zawodów ACM ICPC czuwali profesorowie Krzysztof Diks i Jan Madey oraz doktorzy Marek Cygan i Jakub Radoszewski. W czasie studiów zajmował się siatkówką, koszykówką i tańcem towarzyskim. Oprócz sukcesów drużynowych, kwalifikował się w roku 2010 do finałów konkursów Google Code Jam i TopCoder Open. Był także stażystą w firmach Google (Kraków) i Facebook (Palo Alto, CA).

wojciech.smietanka@gmail.com

Jarosław Grytczuk

Wydział Matematyki i Informatyki
Uniwersytet Jagielloński

Czy wszystko można obliczyć?
Łagodne wprowadzenie do złożoności obliczeniowej

Frank Wilczek, laureat Nagrody Nobla z fizyki z roku 2004, został kiedyś zapytany: „Jeśli mógłbyś zadać jedno fundamentalne pytanie jakiejś nadprzyrodzonej, superinteligentnej istocie, to jak by ono brzmiało?". Jego odpowiedź zapewne zaskoczyła rozmówcę: „Czy P = NP? To pytanie zawiera w sobie wszystkie inne pytania, czyż nie?".

Niniejszy artykuł opowiada o jednym z siedmiu problemów milenijnych – zagadnieniu **Czy P = NP?**, stanowiącym jedno z największych wyzwań współczesnej nauki. Pytanie to jest centralnym problemem otwartym teorii obliczeń – dziedziny leżącej na styku matematyki i informatyki. Jednak jego znaczenie wykracza daleko poza te dyscypliny, dotykając fundamentalnych, filozoficznych pytań o naturę świata i ludzkiego umysłu. Jest przy tym rzeczą zaskakującą, że w istocie problem Czy P = NP? można sprowadzić do prostych układanek, takich jak popularne sudoku.

Czy istnieje *efektywny* sposób rozwiązania sudoku dowolnego rozmiaru? Na te i podobne pytania wciąż nie znamy odpowiedzi. Dzięki osiągnięciom teorii obliczeń wiemy jednak, że znalezienie takiego sposobu przyniosłoby zarazem rozwiązanie wszystkich podobnych problemów na świecie. Wydaje się to zatem niemożliwe, lecz dowodu matematycznego owej niemożliwości jak dotąd nie znaleziono.

1. Sudoku i inne układanki

Historia sudoku

Początków sudoku należy szukać nie tyle w Japonii, co w średniowiecznej Arabii. To tam ówcześni matematycy badali kwadraty liczbowe, których wiersze i kolumny nie zawierają powtarzających się elementów. W XVII wieku kwadraty takie także badał wybitny matematyk Leonhard Euler nazywając je kwadratami łacińskimi.

Kwadrat łaciński jako łamigłówka pojawił się po raz pierwszy w roku 1895 we francuskiej gazecie „Echo Paryża", ale nie spotkał się z uznaniem czytelników. Był to diagram 9x9 z polami do uzupełnienia, ale bez charakterystycznego podziału na 9 sektorów. W obecnej postaci łamigłówka pojawiła się po raz pierwszy w roku 1979 w amerykańskiej gazecie „Dell Magazine". Kilka lat później zawędrowała do Japonii, gdzie nadano jej obecną nazwę. W Polsce sudoku pojawiło się po raz pierwszy w roku 1996 w krzyżówkowym dodatku do „Super Expresu".

Działo się to wszystko na dobrych kilka lat przed erupcją popularności krzyżówki, którą sprokurował wielki miłośnik tego typu rozrywek – prawnik z Nowej Zelandii, Wayne Gold. On to natknął się w trakcie pobytu w Japonii na niewielką książeczką o sudoku. Po kilku latach intensywnych i pasjonujących badań zaproponował tygodnikowi „The Times" opublikowanie swoich sudoku. Stało się to w roku 2004. W ciągu niespełna roku szaleństwo sudoku ogarnęło niemal cały świat. W Polsce przyczynił się do tego tygodnik „Polityka" publikując w roku 2005 dodatek z wieloma zadaniami sudoku; patrz również [8].

Sudoku to popularna japońska krzyżówka liczbowa, która zrobiła w ostatnich latach prawdziwą furorę, dostarczając umysłowej rozrywki milionom ludzi na całym świecie. Reguła zabawy jest prosta: w puste pola kwadratu 9 x 9 należy wpisać cyfry od 1 do 9 tak, aby w każdym wierszu, w każdej kolumnie, i w każdym z 9 sektorów nie powtórzyła się żadna cyfra (rys. 1). Niewielu jednak wie, że ta niewinna układanka kryje w sobie jedną z największych tajemnic nauki, od rozwiązania której zależy być może przyszłość naszej cywilizacji. Nie dziwne więc, że na śmiałka, który sprosta wyzwaniu i odkryje ową tajemnicę czeka spora nagroda wysokości 1 000 000 dolarów, ale o tym później.

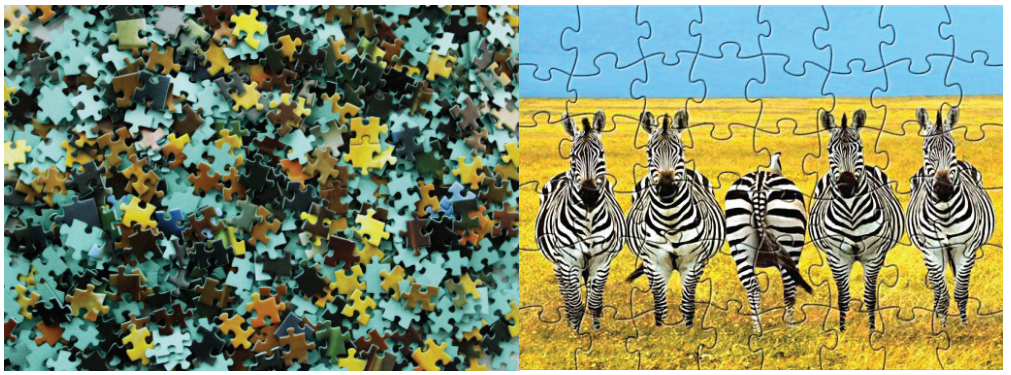
	5	4	6					
				7		9		
			8			3		6
								8
7	6							5
				5				2
9			2					
3		1					7	

1	5	4	6	9	3	2	8	7
6	3	2	5	7	8	9	4	1
8	9	7	1	2	4	6	5	3
5	1	9	8	4	7	3	2	6
2	4	3	9	6	5	7	1	8
7	6	8	3	1	2	4	9	5
4	8	6	7	5	9	1	3	2
9	7	5	2	3	1	8	6	4
3	2	1	4	8	6	5	7	9

Rysunek 1. Sudoku: przed wypełnieniem i wypełnione

Źródło: http://en.wikipedia.org/wiki/Sudoku.

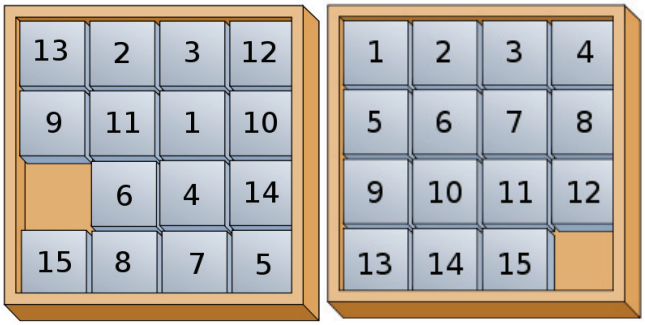
Zacznijmy od prostej obserwacji na temat natury zwykłych układanek, czyli popularnych puzzli. Kto zabawiał się układaniem obrazka z rozsypanych na stole kawałeczków, ten wie, że jest to zadanie raczej trudne, wymagające czasu, cierpliwości i koncentracji. Ale kiedy owe puzzle są już złożone, wystarczy właściwie rzut oka, by upewnić się, że wszystko się zgadza (rys. 2).



Rysunek 2. Puzzle obrazkowe

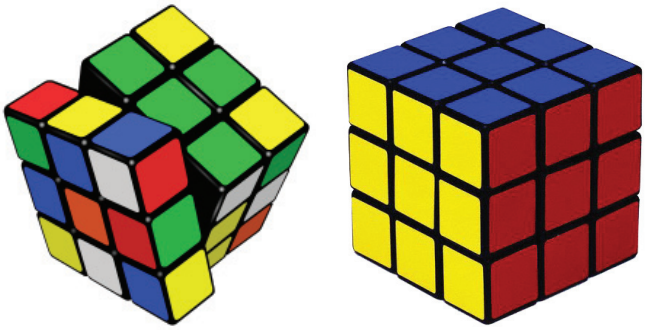
Źródło: http://www.dottysvirtualjigsaws.com/Jigs@wCreateOwnPuzzle.asp.

Podobnie rzecz się ma w przypadku nieco bardziej wyrafinowanych popularnych zabawek, jak układanka Loyda (rys. 3), kostka Rubika (rys. 4), czy też tytułowe sudoku: *sprawdzenie poprawności rozwiązania jest łatwe, niemal natychmiastowe, natomiast znalezienie owego rozwiązania jest raczej trudne, kosztuje sporo wysiłku.*



Rysunek 3. Układanka Loyda

Źródło: http://en.wikipedia.org/wiki/Fifteen_puzzle.



Rysunek 4. Kostka Rubika

Źródło: http://en.wikipedia.org/wiki/Rubik's_Cube.

Kostka Rubika

Ta popularna układanka została wynaleziona w roku 1974 przez Węgra Erno Rubika. Podobny wynalazek opatentował dwa lata później japoński inżynier Terutoshi Ishige. Podstawowa kostka ma wymiary 3x3x3 i składa się z 26 małych sześcianików zamocowanych na obrotowym przegubie. Zabawa polega na obracaniu ściankami kostki tak, aby stały się one jednobarwne. Wszystkich możliwych ustawień kostki jest aż 43 252 003 274 489 856 000. Nie przeszkadza to jednak wielu amatorom tej rozrywki w osiągnięciu imponującej szybkości rozwiązania. Obecny rekord należy do Australijczyka Feliksa Zemdegsa i wynosi 5,66 sekundy. Drugie miejsce w rankingu światowym zajmuje Polak Michał Pleskowicz z czasem 6,11 sekundy. Zawody w układaniu kostki Rubika rozgrywane są po dziś dzień na całym świecie. Istnieje także wiele odmian kostki Rubika o rozmaitych kształtach i wymiarach; patrz również [5].

Nie brakuje przykładów podobnych sytuacji również w matematyce. Znalezienie rozwiązania równania często bywa bardzo złożone, natomiast sprawdzenie jego poprawności przychodzi bez większego trudu. Poniżej równanie, którego rozwiązaniami są liczby 1, –3, 5.

$$x^3 - 3x^2 - 13x + 15 = 0$$

2. Sudoku dla komputera

To, co trudne, a nawet niewykonalne dla człowieka, bywa igraszką dla komputera. W istocie, każdą z wymienionych łamigłówek komputer rozwiąże w ułamku sekundy. Oczywiście dzieje się tak nie dlatego, że komputer jest mądrzejszy od człowieka, ale dlatego, że dysponuje nieporównanie większą mocą obliczeniową. Komputer to po prostu bardzo sprawny mechanizm, który jest w stanie sprawdzić wszystkie możliwości w bardzo krótkim czasie i w ten prymitywny, acz skuteczny sposób znaleźć rozwiązanie.

			14			8	5		10	2	4		6		
		13	10	9	11	12			6		16	15	5		
3	12	5	2	14	6		16						1	11	
	7				2		15		12		3		14	4	16
					8	7	9			6		16	3	1	5
9	1		12		15	3		4	16	7	14				13
2						13	14	8	9			12	15	7	
13	3		6	11							15		8	9	
	2	9		13							5	4		3	6
	16	10	3			9	6	7	4						1
7				8	16	5	3		15	1		9		14	12
6	8	12	1		7			3	14	11					
15	13	14		1		2		16		4				6	
	10	3						14		12	13	1	9	15	2
		2	7	4		15			3	5	10	14	16		
		11		7	3	14		9	1			8			

Rysunek 5. Sudoku 16 x 16

Źródło: http://www.sudoku.4thewww.com/other.php.

Układanka, która miałaby stanowić wyzwanie dla komputera musi mieć większy rozmiar. Wyobraźmy sobie nieco większą tabliczkę sudoku, powiedzmy o wymiarach 16 x 16 (rys. 5). Czy teraz komputer równie szybko znajdzie rozwiązanie? Na pewno nie, ale chyba czas jego poszukiwań nie zwiększy się istotnie. Być może urządzenie poda rozwiązanie po kilku czy kilkunastu sekundach, a nawet jeśli mielibyśmy poczekać parę minut, to i tak nie będzie to żaden dramat.

3. Struś Pędziwiatr

Sprawdźmy, ile czasu zajmie rozwiązanie sudoku o rozmiarach 16x16 jednemu z najszybszych komputerów na świecie, przy zastosowaniu prymitywnej metody przeszukiwania wszystkich możliwości. Komputer ten nazywa się **Struś Pędziwiatr** (ang. *Roadrunner*). Został skonstruowany w laboratoriach firmy IBM w Los Alamos. Właściwie jest to klaster (rys. 6) zajmujący powierzchnię 560 m^2, na który składa się z blisko 19 000 procesorów! W roku 2008 Struś Pędziwiatr pobił rekord świata w szybkości obliczeniowej przekraczając magiczną granicę jednego **petaflopsa**, czyli wykonując 10^{15} operacji (arytmetycznych czy binarnych) na sekundę! Dodajmy jeszcze, że kosztował on firmę IBM, bagatela, 133 000 000 dolarów.

Rysunek 6. Komputer Roadrunner

Źródło:http://en.wikipedia.org/wiki/IBM_Roadrunner.

Superkomputery

Struś Pędziwiatr królował jako najszybszy komputer świata niespełna dwa lata. W roku 2010 pokonał go Jaguar, inny amerykański superkomputer, który z kolei oddał prowadzenie na rzecz chińskiego komputera o nazwie Tianhe-1. Obecny rekord

(2012) należy do Sekwoi – również amerykańskiego komputera firmy IBM. Więcej informacji o pasjonującej rywalizacji maszyn liczących, a także o zastosowaniach ich potężnej mocy obliczeniowej, można znaleźć na stronie Top500: www.top500.org.

W naszym rachunku przyjmiemy kilka uproszczeń. Przypuśćmy, że tabliczka sudoku zawiera jedynie 25 pustych pól. W każde z nich komputer musi wpisać jedną z 4-bitowych liczb od 1 do 16. Przy pojedynczej próbie rozwiązania komputer wpisuje zatem 25 ciągów po 4 bity każdy, a więc łącznie ciąg zerojedynkowy długości 100. Wszystkich możliwości jest zatem 2^{100}, lecz tylko jedna z nich jest właściwym rozwiązaniem układanki. (W oryginalnym sudoku zdarza się, że istnieje więcej poprawnych rozwiązań, my, dla prostoty obliczeń, przyjmiemy założenie o jednoznaczności rozwiązania). Załóżmy dalej, że do sprawdzenia, czy dane uzupełnienie jest tym właściwym, potrzeba komputerowi tylko jednej operacji bitowej. Oczywiście może się zdarzyć przypadkiem, że pierwsza z brzegu ewentualność okaże się dobra, ale może być i tak, że dopiero ostatnie badane rozwiązanie jest właściwe. Zatem, w najgorszym wypadku Struś Pędziwiatr wykona

$$2^{100} = 1\ 267\ 650\ 600\ 228\ 229\ 401\ 496\ 703\ 205\ 376$$

operacji bitowych zanim znajdzie rozwiązanie. Ile czasu mu to zajmie? Hm... policzmy: 10^{15} operacji na sekundę, to daje

$$2^{100}/10^{15} = 1\ 267\ 650\ 600\ 228\ 229,\ 401\ 496\ 703\ 205\ 376 \text{ sekund.}$$

Jedna minuta ma 60 sekund, w jednej godzinie jest 60 minut, jedna doba to 24 godziny, zaś jeden rok to 365 dni. Czyli 1 267 650 600 228 229 to w przybliżeniu... 40 000 000 lat!

Oczywiście przeszukiwanie wszystkich możliwości nie jest najlepszą metodą znalezienia rozwiązania. Od czego mamy rozum, spryt i pomysłowość? Chyba istnieje jakiś sposób pozwalający rozwiązać tak niewinną układankę, jaką jest sudoku, nawet rozmiaru 100x100, w znacznie krótszym czasie. Być może taka metoda istnieje, ale jak na razie nikt jej nie znalazł. Co więcej, jeśliby się to komuś udało, to za jednym zamachem znaleziono by sposób na wszystkie układanki świata!

4. Wyścigi algorytmów

Czas wyjaśnić nieco dokładniej, o czym jest tu mowa. Przede wszystkim interesują nas **problemy obliczeniowe**, czyli takie, których rozwiązanie może

być znalezione za pomocą komputera. Do rozwiązania problemu potrzebny jest **algorytm**, w najgorszym razie przeszukujący wszystkie możliwości (to, jak widzieliśmy, może być zgoła niepraktyczne). Dla jednego problemu może istnieć wiele algorytmów różniących się pod różnymi względami, ale my skupimy się wyłącznie na porównywaniu ich szybkości.

Zabawa jest prosta i przypomina zwykłe wyścigi. Zilustrujmy to na przykładzie problemu **znajdowania największego wspólnego dzielnika** dwóch liczb. Pierwszy sposób polega na znalezieniu rozkładu na czynniki pierwsze obu liczb, a następnie wybraniu jak największej liczby wspólnych dzielników (liczby pierwsze to takie liczby naturalne, które nie dzielą się przez żadną liczbę różną od 1 i od siebie samej, przy czym jedynki do liczb pierwszych nie zaliczamy, rys. 7).

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Rysunek 7. Liczby pierwsze (w żółtych polach)

Źródło: http://www.matemaks.pl/rodzaje-liczb.php.

Jak wiadomo, każda liczba naturalna większa od 1 rozkłada się na iloczyn liczb pierwszych i to w sposób jedyny, jeśli nie zwracać uwagi na kolejność czynników. Na przykład, 84 = 2*2*3*7, zaś 234 = 2*3*3*13. Chcąc znaleźć największy wspólny dzielnik 84 i 234 wystarczy wybrać wspólne czynniki pierwsze z obu rozkładów:

$$\mathrm{NWD}(84, 234) = 2*3 = 6.$$

No tak, ale skąd wziąć rozkład na czynniki pierwsze danej liczby? Okazuje się, że z tym problemem jest podobnie jak z sudoku: nikt nie znalazł jak dotąd znacząco szybszego sposobu niźli prymitywna metoda prób i błędów.

Istnieje jednak sposób obliczenia największego wspólnego dzielnika dwóch liczb niewymagający znajdowania rozkładu na czynniki pierwsze. Jest to znany już w starożytności **algorytm Euklidesa**, polegający na sukcesywnym wykonywaniu dzielenia z resztą. Algorytm ten wykonuje z grubsza 5n operacji bitowych na danych

długości *n* nim poda wynik. Oznacza to, że z problemem rozmiaru naszej nieszczęsnej tabliczki sudoku 16 x 16 Struś Pędziwiatr poradzi sobie w ułamku sekundy![1]

Algorytm Euklidesa

Algorytm Euklidesa jest jednym z najstarszych znanych ludzkości algorytmów. Został wynaleziony przez Euklidesa około 300 lat p.n.e. Co ciekawe, pozostaje on do dziś najlepszym narzędziem znajdowania największego wspólnego dzielnika. Jest także składnikiem wielu innych algorytmów w teorii liczb, powszechnie stosowanych w rozmaitych urządzeniach elektronicznych, głównie związanych z zabezpieczeniami.

Pokażemy działanie algorytmu Euklidesa na przykładzie liczb 234 i 84. Najpierw wykonujemy dzielenie z resztą 234 przez 84:

$$234 = 2*84 + 66.$$

Następnie dzielimy 84 przez otrzymaną resztę 66:

$$84 = 1*66 + 18.$$

W kolejnych krokach dzielimy resztę otrzymaną w przedostatnim kroku przez resztę z ostatniego dzielenia:

$$66 = 3*18 + 12,$$
$$18 = 1*12 + 6,$$
$$12 = 2*6.$$

Algorytm kończy się w momencie otrzymania reszty 0. Największym wspólnym dzielnikiem początkowych liczb 234 i 84 jest ostatnia niezerowa reszta, czyli 6. Dzieje się tak dlatego, że pary liczb sąsiednich w ciągu 234, 84, 66, 18, 12, 6 mają dokładnie ten sam zbiór wspólnych dzielników.

5. Problemy łatwe

Algorytmy takie jak algorytm Euklidesa nazywamy **efektywnymi**, lub po prostu **szybkimi**. Istnieje wiele ważnych zagadnień obliczeniowych, dla których znaleziono szybkie algorytmy. To dzięki takim właśnie matematycznym wynalazkom możemy dziś błyskawicznie wyszukać w Internecie interesujące nas hasło, znaleźć najkrótszą drogę przejazdu z miasta A do miasta B, czy też bezpiecznie dokonywać przelewów bankowych.

[1] Więcej na temat algorytmu Euklidesa, w tym wyjaśnienie, dlaczego działa tak szybko, można przeczytać w rozdziale *Historia rachowania – ludzie, idee, maszyny.*

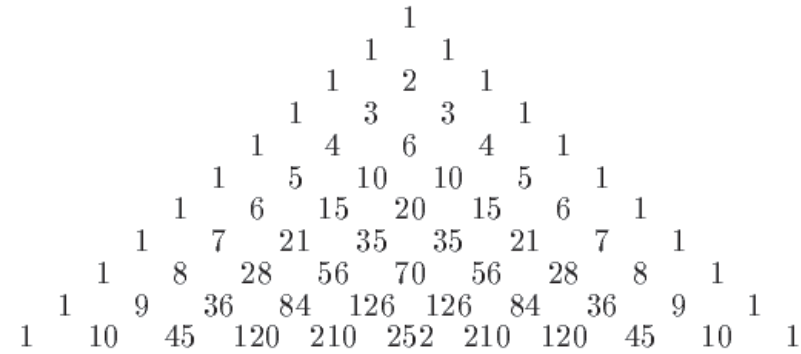
Zauważmy, że nawet jeśli liczba operacji wykonywanych przez algorytm na danych rozmiaru n wzrośnie do n^2 czy n^3, to i tak nie wpłynie to istotnie na realny czas przebiegu algorytmu (nawet dla gigantycznych danych wejściowych). W teorii obliczeń algorytm nazywamy **efektywnym** (lub **wielomianowym**), jeżeli czas jego przebiegu na danych rozmiaru n, mierzony liczbą wykonywanych operacji bitowych, jest ograniczony przez pewną potęgę n^k, gdzie k jest stałą, czyli ograniczony funkcją wielomianową zmiennej n. W żargonie takie algorytmy nazywamy **szybkimi**, a problemy obliczeniowe, które można rozwiązać za pomocą takich algorytmów – **łatwymi**. Podsumowując, problem obliczeniowy jest łatwy, jeżeli istnieje dla niego szybki algorytm.

Klasę problemów łatwych oznaczamy literą **P** (ang. *polynomial* – wielomian).

Problemami łatwymi są na przykład: znajdowanie największego wspólnego dzielnika, wyszukiwanie wzorca w tekście czy znajdowanie najkrótszej drogi w grafie.

Sprawdzanie pierwszości

Łatwy problem nie zawsze łatwo rozpoznać. Na przykład długo nie było wiadomo, czy problem sprawdzania czy dana liczba jest liczbą pierwszą jest łatwy. Dopiero w roku 2002 świat obiegła ekscytująca informacja o dokonaniu trzech Hindusów: Manindry Agrawala, Neeraja Kayala i Nitina Saxeny, którzy odkryli szybki algorytm rozwiązujący to zagadnienie. Ich artykuł noszący znamienny tytuł *PRIMES is in P*, ukazał się w prestiżowym czasopiśmie matematycznym „Annals of Mathematics". Algorytm AKS wykorzystuje nową charakteryzację liczb pierwszych podobną do tej opartej na trójkącie Pascala (rys. 9): Liczba n jest pierwsza wtedy i tylko wtedy, gdy dzieli każdą liczbę w n-tym wierszu trójkąta, za wyjątkiem skrajnych jedynek; patrz również [3].



Rysunek 8. Trójkąt Pascala

Źródło: http://en.wikipedia.org/wiki/Pascal's_triangle.

6. Kwadratura koła

Udowodnienie, że coś jest łatwe bywa czasem dość trudne. Ale jeszcze trudniej wykazać, że coś łatwym nie jest. Problem obliczeniowy nazywamy **trudnym**, jeżeli nie jest łatwy. Aby pokazać, że jakiś problem obliczeniowy jest trudny, należy zatem udowodnić, że nie istnieje szybki algorytm, który służy do jego rozwiązywania. To psychologicznie zgoła odmienna sytuacja.

Zastanówmy się dla przykładu nad problemem **faktoryzacji**, czyli rozkładu na czynniki pierwsze: daną liczbę naturalną rozłożyć na czynniki pierwsze. Na przykład:

$$2013 = 3*11*61.$$

Jak dotąd nikomu nie udało się znaleźć szybkiego algorytmu rozwiązującego to zadanie. To nie oznacza jednak samo przez się, że takiego algorytmu nie ma. Być może problem faktoryzacji jest trudny, ale żeby mieć co do tego absolutną pewność należy udowodnić, że spośród nieskończenie wielu szybkich algorytmów na świecie, żaden nie nadaje się do rozkładania na czynniki pierwsze. Tego jak dotąd również nikomu nie udało się dokonać. Nie wiadomo zatem, czy problem faktoryzacji jest łatwy, czy trudny.

Kwadratura koła

Pierwsze wzmianki o konstrukcjach kwadratu o polu przybliżającym pole danego koła znaleźć można w słynnym papirusie Rhinda z roku 1800 p.n.e. Sam problem znany był zapewne jeszcze wcześniej, w starożytnym Babilonie. W postaci klasycznej konstrukcji geometrycznej sformułowali go po raz pierwszy Pitagorejczycy. Pomimo sporych wysiłków nie udało się jednak greckim matematykom znaleźć poszukiwanej konstrukcji. Musieli oni zadowolić się rozwiązaniami przybliżonymi. Na przykład, Archimedes wpisał w koło i opisał na nim sześciokąt foremny, dziewięciokrotnie podwoił liczbę jego boków, a następnie zamienił ten wielokąt na kwadrat.

Rozstrzygnięcia problemu nie przyniosły także wieki średnie ani epoka nowożytna, choć zajmowali się nim najznakomitsi matematycy tamtych czasów – Fibonacci, François Viete, Gotfried Wilhelm Leibniz czy Leonhard Euler. Jedną z najciekawszych konstrukcji przybliżonych podał polski matematyk i mechanik Adam Kochański (1631-1700).

Ostateczne potwierdzenie narastającego przekonania o niemożliwości kwadratury koła nadeszło od strony algebry. Dzięki genialnemu pomysłowi Kartezjusza, figury geometryczne można opisywać za pomocą równań algebraicznych w prostokątnym układzie współrzędnych. Punkty, które da się skonstruować za pomocą cyrkla i linijki mają charakterystyczną postać tzw. pierwiastników (liczb powstających na bazie

ułamków poprzez wielokrotne nakładanie pierwiastka kwadratowego oraz dodawanie i mnożenie). Dzieje się tak dlatego, że prostą (linijkę) opisuje równanie pierwszego stopnia, zaś koło (cyrkiel) – równanie stopnia drugiego. Współrzędne punktów przecięcia prostych i okręgów powstające w toku konstrukcji muszą być zatem rozwiązaniami równań algebraicznych, których stopień wyraża się potęgą dwójki. Gdyby konstrukcja kwadratury koła była możliwa, to taką liczbą musiałaby być liczba π. Ale w roku 1880 Lindemann udowodnił, że π jest liczbą przestępną – w ogóle nie istnieje równanie algebraiczne, którego π jest pierwiastkiem.

To dokonanie ostatecznie zamknęło kwestię kwadratury koła w świecie nauki. O dziwo, wśród amatorów matematyki do dziś zdarzają się sceptycy poszukujący uparcie tej nieuchwytnej konstrukcji.

Sytuacja przypomina tę sprzed kilkuset lat, kiedy to borykano się ze słynnym zagadnieniem **kwadratury koła**. Problem polegał na znalezieniu geometrycznej konstrukcji (za pomocą cyrkla i linijki) kwadratu o polu równym polu danego koła. Ponieważ przez setki lat od sformułowania tego zadania nikomu takiej konstrukcji nie udało się znaleźć, większość matematyków zaczęła sądzić, że jej po prostu nie ma. W końcu udało się udowodnić, że kwadratura koła jest niemożliwa, przy użyciu metod nowoczesnej na owe czasy algebry. Być może historia powtórzy się w przypadku **problemu faktoryzacji**...

Problem faktoryzacji jest w pewnym sensie odwrotny do problemu sudoku. Mamy tu już złożoną układankę w postaci danej liczby, a szukamy puzzli, które ją tworzą, czyli liczb pierwszych, których iloczyn daje tę liczbę. Jeśli jednak ktoś dostarczy nam rozkładu, to łatwo sprawdzimy czy jest on poprawny wykonując zwykłe mnożenie. Pod tym względem problem faktoryzacji i sudoku są podobne.

W tym miejscu warto przytoczyć zabawną sytuację, która miała miejsce na kongresie matematycznym w 1904 roku. Otóż matematyk amerykański Frank Nelson Cole w czasie swojego wystąpienia podszedł do tablicy i napisał

$$2^{67} - 1 = 147\,573\,952\,589\,676\,412\,927 = 193\,707\,721 * 761\,838\,257\,287,$$

a następnie dowiódł prawdziwości napisanej równości poprzez pomnożenie liczb sposobem pisemnym. Wszystko to odbyło się w pełnej napięcia ciszy. Dopiero na koniec wyznał, że znalezienie owego rozkładu zajęło mu, bagatela, 20 niedziel!

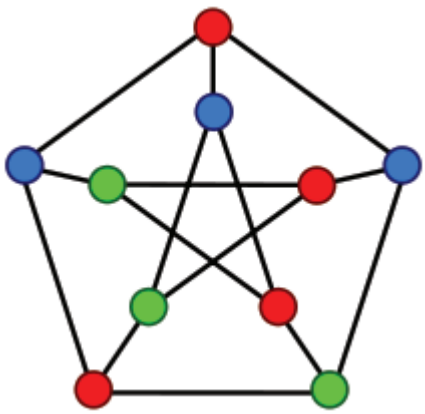
Liczby pierwsze postaci $2^n - 1$ stanowią przedmiot fascynacji matematyków od dawna (noszą one nazwę **liczb Mersenne'a**). Nietrudno wykazać, że jeżeli $2^n - 1$ jest liczbą pierwszą, to i n musi być liczbą pierwszą. Przykład liczby Cole'a pokazuje, że na odwrót nie zawsze to zachodzi. Oczywiście dzisiaj do sprawdzenia, czy $2^n - 1$ jest liczbą pierwszą użyjemy komputera. Od roku 1996 trwa w Internecie wielkie poszukiwanie liczb pierwszych Mersenne'a w ramach projektu GIMPS (http://www.mersenne.org/). Największą liczbą Mersenne'a znalezioną w ten

kolektywny sposób, i zarazem największą znaną obecnie liczbą pierwszą, jest $2^{43\,112\,609} - 1$. Do dziś nie wiadomo, czy liczb Mersenne'a jest nieskończenie wiele.

7. Klasa NP

Istnieje mnóstwo problemów obliczeniowych, o których nie wiemy, czy są łatwe, czy trudne. Najważniejszą klasę tworzą problemy o naturze układanek: łatwo zweryfikować poprawność podanego rozwiązania, natomiast nie zawsze prosto je znaleźć. Klasę tę oznaczamy symbolem **NP** (ang. *nondeterministic polynomial*). W tej klasie znajduje się zarówno problem sudoku i problem faktoryzacji, jak i całe mnóstwo kombinatorycznych zagadnień optymalizacyjnych, spośród których przedstawimy poniżej dwa, polecamy tutaj łagodne wprowadzenie do złożoności obliczeniowej [4].

Pierwsze z nich to problem **3-kolorowania grafu**: czy dany graf da się pokolorować poprawnie trzema kolorami? Kolorowanie grafu jest **poprawne,** jeżeli żadne dwa wierzchołki połączone krawędzią nie są tego samego koloru (rys. 9).



Rysunek 9. Poprawne pokolorowanie grafu

Źródło: http://en.wikipedia.org/wiki/Graph_coloring.

Poprawność pokolorowania grafu jest łatwo zweryfikować, wystarczy przejrzeć wszystkie krawędzie grafu i porównać kolory ich końców. Problem ten jest więc w klasie NP. Ale jak stwierdzić, czy poprawne 3-kolorowanie danego grafu w ogóle istnieje? Można oczywiście sprawdzić wszystkie możliwości, ale to może trwać długo, wszak jest ich aż 3^n (gdzie n oznacza liczbę wierzchołków grafu). Podobnie jak w przypadku faktoryzacji nie wiemy, czy 3-kolorwanie grafu jest łatwe, czy trudne. Nietrudno natomiast przekonać się, że analogiczny problem 2-kolorowania grafu jest łatwy.

Drugim przykładem popularnego problemu z klasy NP jest **problem SAT**, w którym pytamy się, czy dana formuła logiczna jest **spełnialna**, to znaczy, czy istnieje takie podstawienie zer i jedynek w miejsce zmiennych, że wartość logiczna danej formuły wyniesie 1. Na przykład formuła

$$F = (p \vee q \vee r) \wedge (p \vee \neg q \vee s) \wedge (q \vee r \vee \neg s)$$

jest spełnialna, ponieważ przy podstawieniu $p = q = 1$ oraz dowolnych wartościach r i s, w każdym nawiasie pojawi się jedynka:

$$F = (1 \vee 1 \vee r) \wedge (1 \vee 0 \vee s) \wedge (1 \vee r \vee \neg s).$$

Łatwo jest więc zweryfikować poprawność pojedynczego podstawienia, jednak jak dotąd nie wynaleziono szybkiego algorytmu rozstrzygającego istnienie podstawienia spełniającego, ale nie wykluczono też takiej możliwości. Ciągle nie wiadomo, czy problem SAT jest na pewno trudny.

8. Problem milenijny

Czy wobec tego w ogóle znany jest jakiś przykład problemu w klasie NP, o którym wiemy już, że na pewno nie jest łatwy? No właśnie nie! I to jest nasz główny problem. Po wielu latach zmagań i wysiłków, wciąż jesteśmy skazani na spekulacje:

Czy P = NP?

Wiele wskazuje na to, że odpowiedź na to pytanie jest negatywna. Równość P = NP oznaczałaby, że właściwie nie ma naturalnych problemów obliczeniowych, które byłyby trudne, że, cytując słowa Margaret Fuller, *wszystko jest trudne nim stanie się łatwe*. Wszystkie układanki świata da się rozwiązać szybko, wszystkie twierdzenia w matematyce da się udowodnić „mechanicznie". Równałoby się to w gruncie rzeczy ze stwierdzeniem, że ludzką kreatywność dałoby się w pewnym sensie zautomatyzować. Czy możemy wyobrazić sobie komputer komponujący muzykę tak wspaniałą jak muzyka Chopina? Chyba nie, chociaż istnieją już programy komputerowe potrafiące tworzyć muzykę naśladującą style sławnych kompozytorów.

W roku 2000 z okazji przełomu tysiącleci wskazano siedem najważniejszych problemów matematycznych oraz wyznaczono nagrody pieniężne za rozwiązanie każdego z nich, każda w wysokości 1 000 000 dolarów. Fundatorem tych nagród, jak i całego instytutu matematycznego swojego imienia jest ame-

rykański biznesmen Landon T. Clay. Problem rozstrzygnięcia czy P = NP jest jednym z owych siedmiu problemów milenijnych.

W tym miejscu warto wspomnieć, że jak dotąd rozwiązano tylko jeden z problemów milenijnych. Chodzi o słynną **hipotezę Poincarégo**, dotyczącą powierzchni 3-wymiarowych w przestrzeni 4-wymiarowej. Udowodnił ją w roku 2006 Rosjanin Grigorij Perelman. Ku zdumieniu świata, nie zechciał on odebrać swojej nagrody; patrz [7].

Grigorij Perelman (1966-)

Perelman udostępnił w Internecie manuskrypt z rozwiązaniem hipotezy Poincarégo w 2003 roku. Sprawdzenie poprawności wszystkich zawiłych rozumowań zajęło ekspertom trzy lata. W roku 2006 ogłoszono triumfalnie rozwiązanie hipotezy i przyznano Perelmanowi medal Fieldsa – jedną z najbardziej prestiżowych nagród matematycznych. Jednak Perelman odmówił jej przyjęcia, twierdząc, że doniosłość odkrycia matematycznego może być zweryfikowana dopiero po wielu latach od jego dokonania.

Odmowa przyjęcia „matematycznego Nobla" odbiła się głośnym echem, nie tylko w środowisku matematycznym. Skromny, acz ekscentryczny geniusz stał się z dnia na dzień bohaterem licznych medialnych doniesień.

Wrzawa wokół jego osoby wybuchła ponownie po odmowie przyjęcia miliona dolarów nagrody ufundowanej przez Instytut Matematyczny Claya. Tym razem Perelman jako powód wskazał niesprawiedliwe, jego zdaniem, pominięcie Richarda Hamiltona jako matematyka, który przyczynił się w równym stopniu do rozwiązania hipotezy. W istocie faktem jest, że to właśnie Hamilton zaproponował właściwe podejście do hipotezy Poincarégo, a także do ogólniejszej hipotezy Thurstona o klasyfikacji powierzchni trójwymiarowych.

Obecnie Grisza Perelman jest bezrobotnym matematykiem, mieszkającym w skromnym mieszkaniu na petersburgskim blokowisku, rzadko kontaktującym się ze światem zewnętrznym.

9. NP-zupełność

Problem P = NP? może sprawiać wrażenie zagadnienia bardzo rozległego – wszak chodzi tu o porównanie dwóch nieskończonych klas problemów obliczeniowych. Po głębszym badaniu okazało się jednak, że właściwie cała zagadka sprowadza się do pojedynczego problemu, takiego jak np. sudoku. Czyżby los ludzkości zależał od tej niewinnej układanki?

Aby wyjaśnić, w czym rzecz, rozważmy ponownie problem SAT i problem 3-kolorowania grafu. Otóż można udowodnić, że dla każdego grafu G istnieje for-

muła logiczna $F = F(G)$ taka, że G jest 3-kolorowalny wtedy i tylko wtedy, gdy F jest spełnialna. Ponadto, skonstruowanie takiej formuły „kodującej" problem 3-kolorowalności grafu jest możliwe w czasie wielomianowym. Oznacza to, że w istocie problem 3-kolorowalności grafu sprowadza się łatwo do problemu SAT. Zatem, jeżeli problem SAT jest łatwy, to problem 3-kolorowalności też jest łatwy.

W roku 1971 Kanadyjczyk Stephen Cook dowiódł, że właściwie każdy problem z klasy NP w podobny sposób sprowadza się do problemu SAT. Mówimy, że problem SAT jest **NP-zupełny**.

Jest to zadziwiające zjawisko: oto rozstrzygnięcie, czy P = NP sprowadza się w zasadzie do zbadania jednego jedynego problemu obliczeniowego – problemu SAT. Jeśli problem ten jest trudny, to oczywiście mamy P ≠ NP. Jeśli jest łatwy, to wszystkie problemy w klasie NP są łatwe i mamy P = NP. Jest jeszcze trzecia możliwość, ale o tym nieco dalej.

Odnotujmy jeszcze, iż problem SAT nie jest jedynym NP-zupełnym problemem obliczeniowym. W rzeczywistości znaleziono wiele takich problemów w kombinatoryce, a jednym z nich jest także sudoku. Dowiódł tego Japończyk Takayuki Yato w roku 2003. Rezultat ten oznacza, że dowolny problem z klasy NP (np. SAT, 3-kolorowalność, faktoryzację itp.) można „zakodować" w postaci odpowiedniej tabliczki sudoku, której rozwiązanie da pośrednio rozwiązanie danego problemu.

Problemy NP-zupełne

Problem SAT nie jest jedynym znanym problemem NP-zupełnym. Jest nim także problem 3-kolorowania grafu i wiele innych naturalnych problemów o charakterze optymalizacyjnym. Na przykład, słynny problem komiwojażera, w którym mamy graf pełny z liczbami na krawędziach, a zadanie polega na znalezieniu cyklu przechodzącego przez wszystkie wierzchołki grafu tylko raz, którego suma jest minimalna[2].

Problemy NP-zupełne to w pewnym sensie najtrudniejsze problemy w klasie NP – podanie algorytmu wielomianowego dla jednego z nich pociąga za sobą istnienie algorytmów wielomianowych dla wszystkich problemów NP-zupełnych. Ustalenie, czy dany problem jest NP-zupełny bywa czasem dość trudne. Nie wiadomo na przykład, czy problem faktoryzacji jest NP-zupełny, czy nie.

[2] W sytuacji, gdy nie jest znany wielomianowy, czyli szybki algorytm rozwiązywania problemu optymalizacyjnego, konstruowane są algorytmy, które dostarczają rozwiązania przybliżone, a więc nie zawsze najlepsze.

10. Między prawdą a nieprawdą

Wydawać by się mogło, że problem Czy P = NP? może mieć jedynie dwa rozstrzygnięcia: albo P = NP, albo P ≠ NP. Okazuje się, że istnieje jeszcze trzecia możliwość...

W roku 1938 **Kurt Gödel** dokonał odkrycia, które wstrząsnęło fundamentami matematycznego gmachu. Wykazał mianowicie, że istnieją w matematyce hipotezy, których nigdy nie da się rozstrzygnąć – ani potwierdzić, ani obalić! Od tej pory matematycy liczą się z tym, że właściwie każdy problem otwarty może okazać się **nierozstrzygalny**. Rozważmy dla przykładu słynną **hipotezę $3x + 1$**. Dotyczy ona liczbowej zabawy, którą możemy rozpocząć od dowolnej liczby naturalnej n. Jeżeli n jest parzyste, to dzielimy n przez 2. Jeżeli n jest nieparzyste, to mnożymy n przez 3 i dodajemy 1. Z nową liczbą postępujemy podobnie, z kolejną tak samo, i tak dalej. Na przykład, jeżeli $n = 7$, to w pierwszym kroku dostajemy 3*7 + 1 = 22. Ponieważ 22 jest liczbą parzystą, dzielimy 22 przez 2 i dostajemy 11. Ta ostatnia liczba jest z kolei nieparzysta, zatem obliczamy 3*11 + 1 = 34. Znów dzielimy przez 2 i dostajemy 17. W kolejnych krokach dostajemy ciąg liczb: 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, i w końcu 1. Hipoteza $3x + 1$ głosi, że zaczynając tę procedurę od dowolnej liczby naturalnej n, zawsze dojdziemy na końcu do 1. Tę zabawę można zapisać w postaci następującego algorytmu:

```
while n ≠ 1 do
    if n mod 2 = 0 then n:= n/2
    else n:= 3*n + 1
```

Jest rzeczą zadziwiającą, że tak prostej zagadki najtęższe matematyczne umysły nie potrafią rozwiązać. W czym leży trudność tego problemu? Niektórzy eksperci sądzą, że być może przyczyną jest nierozstrzygalność. Być może to, czego szukamy, czyli matematyczny dowód prawdziwości tej hipotezy, po prostu nie istnieje. Ale czy to nie oznacza, że hipoteza jest po prostu fałszywa? Że musi istnieć liczba n, z której nigdy do 1 nie dojdziemy? No właśnie, że nie! Równie dobrze może być prawdą, że i dla przeciwnej hipotezy nie ma matematycznego dowodu. Tego właśnie dowiódł Gödel: istnieją w elementarnej arytmetyce zdania, których nie da się udowodnić, ale zarazem nie da się udowodnić ich negacji. Być może takim zdaniem jest hipoteza $3x + 1$, ale tego też jak dotąd nikomu nie udało się udowodnić.

Podobnie rzecz się ma z problemem Czy P = NP?. Czyżby jego rozstrzygnięcie leżało poza zasięgiem matematycznej dedukcji?

Kurt Gödel (1906-1978)

Odkrycia Gödla zalicza się do najdonioślejszych dokonań myśli ludzkiej XX wieku. Jego największym osiągnięciem jest z pewnością wspomniane twierdzenie o niezupełności arytmetyki, gwarantujące istnienie zdań, których nie da się udowodnić ani obalić. Główny pomysł przypomina słynny paradoks kłamcy: „To zdanie jest fałszywe". Gödel wyszedł od podobnego zdania: „To zdanie nie ma dowodu", umiejętnie je matematycznie formalizując. Należy jeszcze dodać, że rezultaty te są słuszne jedynie przy założeniu niesprzeczności arytmetyki, która oznacza, że nie jest możliwe, aby jakieś twierdzenie i jego zaprzeczenie jednocześnie można było udowodnić. Jeden z największych matematyków wszechczasów David Hilbert postulował udowodnienie niesprzeczności arytmetyki. Jednak projekt ten okazał się w świetle wyników Gödla niemożliwy do realizacji.

Gödel zajmował się również teorią względności. Wykazał on m.in., że teoria Einsteina dopuszcza możliwość podróży w czasie. Niedawno odkryto również, że to on właściwie jako pierwszy, w liście do Johna von Neumanna, sformułował problem równoważny z problemem Czy P = NP?

11. Czy wszystko można obliczyć?

Dokonania Kurta Gödla w dziedzinie podstaw matematyki odegrały także znaczącą rolę w narodzinach informatyki. To właśnie one zainspirowały **Alana Turinga** do sformułowania pojęcia **maszyny Turinga** – fundamentu współczesnej teorii obliczeń stanowiącego matematycznie ścisłą formalizację pojęcia algorytmu. Szkoda, że ten wybitny matematyk nie dożył czasów, kiedy to na każdym niemal kroku natknąć się można na fizyczną realizację jego *automatic machine*.

Jedna z prostych, acz nieoczywistych rzeczy wynikająca z tej formalizacji to istnienie problemów decyzyjnych algorytmicznie nierozstrzygalnych. Cóż to jest takiego problem decyzyjny? To po prostu problem dotyczący liczb naturalnych, w którym jedyne możliwe odpowiedzi to TAK lub NIE. Na przykład: „Czy n jest liczbą pierwszą?". Jeżeli $n = 17$, to odpowiedź brzmi TAK, jeśli $n = 18$, to wówczas odpowiedź brzmi NIE. Problem decyzyjny nazywamy **rozstrzygalnym algorytmicznie**, jeżeli istnieje algorytm (w sensie Turinga), który dla zadanego n znajduje prawidłową odpowiedź – TAK lub NIE.

Zamieniając TAK na 1 i NIE na 0, widzimy, że każdy problem decyzyjny możemy uważać za nieskończony ciąg zerojedynkowy. Początek tego ciągu dla problemu pierwszości to 0110101000101... Nie wchodząc w szczegóły definicji pojęcia algorytmu zgodzimy się zapewne, że jego opis (np. w postaci programu komputerowego) to pewien **skończony** ciąg zerojedynkowy. Z kolei skończony ciąg zerojedynkowy możemy traktować jako binarne przedstawienie pewnej liczby natu-

ralnej. Widzimy zatem, że wszystkich możliwych algorytmów jest nie więcej niż liczb naturalnych. Natomiast wszystkich możliwych problemów decyzyjnych jest więcej – nie da się bowiem wszystkich **nieskończonych** ciągów zerojedynkowych ponumerować liczbami naturalnymi. W istocie, wyobraźmy sobie, że jednak to się udało i mamy listę wszystkich takich ciągów C_1, C_2, C_3,... . Pomyślmy teraz o ciągu X, którego pierwszy wyraz różni się od pierwszego wyrazu ciągu C_1, drugi wyraz różni się od drugiego wyrazu ciągu C_2, trzeci wyraz różni się od trzeciego wyrazu ciągu C_3, i tak dalej. Ciąg X jest więc ściśle określonym nieskończonym ciągiem zerojedynkowym, powinien więc znajdować się na naszej liście. Ale którym ciągiem z listy może być X? Pierwszym? Nie, bo różni się od C_1 na pierwszej pozycji. Drugim? Też nie – przecież różni się od niego na drugiej pozycji. Setnym? Nie, albowiem setny wyraz ciągu C_{100} jest inny niż setny wyraz X. Ciągu X nie ma na liście. Zatem lista taka, obejmująca wszystkie ciągi, nie może istnieć.

Powyższy eksperyment myślowy pokazuje, że istnieją problemy decyzyjne, których nie da się rozwiązać za pomocą żadnego algorytmu. Przykładu takiego problemu dostarczył jako pierwszy sam Turing w swojej fundamentalnej pracy z roku 1936 *On computable numbers, with an application to the Entscheidungsproblem*. Jest to tzw. **problem stopu** – na wejściu dostajemy algorytm A wraz z pewnymi danymi wejściowymi (np. algorytm $3x + 1$ i liczbę $n = 7$) i mamy stwierdzić, czy algorytm A zatrzyma się na tych danych. Jak wykazał Turing problem ten jest nierozstrzygalny – nie istnieje algorytm, który rozwiązuje tak postawione zagadnienie.

Problem stopu

Podamy proste uzasadnienie, że problem stopu nie jest rozstrzygalny, czyli że nie jest możliwe napisanie programu, który może zbadać każdy inny program i stwierdzić w każdym przypadku, czy po uruchomieniu zatrzyma się on, czy też wejdzie w nieskończoną pętlę.

Załóżmy jednak, że istnieje funkcja logiczna `T(R)`, której argumentem `R` jest jakikolwiek program i `T(R) = True`, jeśli program `R` kończy swoje działania, `T(R) = False`, jeśli program `R` nie kończy działania. Rozważmy teraz następujący podprogram `P`:

```
proc P;
   while T(P) do;
   return
```

Łatwo zauważyć, że jeśli `T(P) = True`, to program `P` zapętla się, a kończy działanie tylko wtedy, gdy `T(P) = False`. W obu przypadkach funkcja `T(P)` ma złą wartość i ta sprzeczność pokazuje, że funkcja `T` nie może istnieć.

Jak to jednak często bywa, o konkretnym problemie decyzyjnym nie zawsze łatwo rozstrzygnąć, czy jest rozstrzygalny, czy też nie. Jednym z najsławniejszych zagadnień tego typu był dziesiąty problem Hilberta dotyczący rozwiązalności równań diofantycznych (takich jak np. słynne równanie Fermata $x^n + y^n = z^n$). Dopiero po 70 latach Rosjanin Yuri Matiyasevich udowodnił, że problem Hilberta jest nierozstrzygalny algorytmicznie.

Alan Turing (1912-1954)

Stworzenie teoretycznego modelu współczesnego komputera to niejedyne osiągnięcie Alana Turinga, które istotnie wpłynęło na losy świata. Podczas II wojny światowej pracował w ośrodku w Bletchley Park pod Londynem w zespole brytyjskich kryptologów nad łamaniem szyfrów Enigmy, niemieckiej maszyny szyfrującej. Dzięki wcześniejszym pracom polskich kryptologów – Mariana Rejewskiego, Henryka Zygalskiego i Jerzego Różyckiego, którzy już w roku 1932 złamali kod Enigmy starszego typu – zespół Turinga skonstruował specjalne urządzenia (tzw. bomby kryptologiczne), również bazując na pomyśle polskich kryptologów, które pomagały rozszyfrowywać niemieckie depesze praktycznie przez cały okres działań wojennych od roku 1940. W Bletchley Park od roku 1943 pracował również pierwszy elektroniczny komputer Colossus.

Po wojnie Turing pracował m.in. nad budową pierwszego brytyjskiego komputera według architektury von Neumanna. W tym samym czasie rozpoczyna badania nad analogiami działania maszyny liczącej i ludzkiego mózgu, studiując w tym celu fizjologię i neurologię. W efekcie proponuje swój słynny test Turinga, jako metodę rozstrzygnięcia czy maszyna potrafi „myśleć". Test ten polega na rozmowie sędziego – człowieka w języku naturalnym z pozostałymi stronami poprzez ekran komputera. Jeśli sędzia nie jest w stanie określić, czy któraś ze stron jest maszyną czy człowiekiem, wtedy mówi się, że maszyna przeszła test. Zakłada się, że zarówno człowiek, jak i maszyna próbują przejść test zachowując się w sposób możliwie zbliżony do ludzkiego.

Od roku 1991 urządzane są zawody o nagrodę Loebnera bazujące na teście Turinga. Jak dotąd żadnemu programowi nie udało się zdobyć złotego medalu Loebnera.

12. Epilog

W opinii większości ekspertów mało prawdopodobne jest, abyśmy ujrzeli rozwiązanie problemu Czy P = NP? w najbliższej przyszłości. Nie jest to jednak powód do pesymizmu. W gruncie rzeczy obecna sytuacja sprzyja rozwojowi teorii obliczeń – nic tak bowiem nie stymuluje badań naukowych jak twardy

opór materii problemu. Chęć pokonania trudności, sprostania wyzwaniu, zaspokojenia nabrzmiałej ciekawości – to od wieków główny motor napędowy nauki. I często ważniejsze i donioślejsze okazują się od samej odpowiedzi na dręczące nas pytanie, dokonania służące jej znalezieniu. Jakiż pożytek płynie dla ludzkości, lub choćby dla samej matematyki, z faktu, że suma dwóch n-tych potęg liczb naturalnych nigdy nie jest n-tą potęgą dla $n > 2$? Nie wydaje się, aby poza doznaniem estetycznym, własność ta miała jakiekolwiek znaczenie. Jednakowoż, matematycy zrobili wiele, aby swoją pewność w tym względzie posiąść, budując przez setki lat potężny aparat matematyczny, którego wykorzystanie wykracza daleko poza elementarną arytmetykę. Niechaj więc jak najdłużej pytanie Czy P = NP? odpiera ataki ludzkiego geniuszu, wciągając nas w głąb tajemniczego świata obliczeń.

Literatura

1. Bartnicki T., *Jak wygrać milion dolarów w Sapera*, „Matematyka-Społeczeństwo-Nauczanie" 2008, nr 40
2. Conway J.H., Guy R.K., *Księga liczb*, WNT, Warszaw 2006
3. Czerwiński W., *Test na liczbę pierwszą*, „Delta" czerwiec 2012
4. Harel D., *Rzecz o istocie informatyki*, WNT, Warszawa 2001
5. Hintze W., *Magiczna kostka*, PWN, Warszawa 1987
6. Kordos M., *Wykłady z historii matematyki*, WSiP, Warszawa 1994
7. Strzelecki P., *Hipoteza Poincarego*, „Delta" styczeń 2004
8. Wilson R.J., *Jak rozwiązywać sudoku*, Dom Wydawniczy Rebis, Poznań 2005
9. Yan S.Y., *Teoria liczb w informatyce*, PWN, Warszawa 2006

Prof. nzw. dr hab. Jarosław Grytczuk

ukończył studia matematyczne w Wyższej Szkole Pedagogicznej w Zielonej Górze (obecnie Uniwersytet Zielonogórski). Doktorat z matematyki został mu nadany w roku 1996 na Uniwersytecie Adama Mickiewicza w Poznaniu. Stopień doktora habilitowanego uzyskał w roku 2006, również na UAM. Obecnie Jarosław Grytczuk jest profesorem nadzwyczajnym na Wydziale Matematyki i Informatyki Uniwersytetu Jagiellońskiego oraz na Wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej. Stale współpracuje z czołowymi przedstawicielami matematyki dyskretnej i informatyki teoretycznej zarówno w kraju, jak i zagranicą. Jest współorganizatorem Polskich Konferencji Kombinatorycznych odbywających się cyklicznie od roku 2006. Prowadzi również działalność popularyzatorską, występując często na wykładach dla młodzieży. W swoim dorobku naukowym posiada około 40 prac opublikowanych w czasopismach o zasięgu międzynarodowym.

grytczuk@tcs.uj.edu.pl
http://tcs.uj.edu.pl/Grytczuk

Krzysztof Stencel
Instytut Informatyki
Uniwersytet Warszawski

Homo informaticus colligens, czyli człowiek zbierający dane

Dane zgromadzone przez przedstawicieli *Homo informaticus* są niewyobrażalne. Mierzy się je setkami eksabajtów. Jeden eksabajt to aż 260 bajtów, czyli około 1 tryliona bajtów (1018). *Colligo, ergo sum* (czyli gromadzę więc jestem) to jedno z ulubionych powiedzonek niektórych *Homo informaticus*, którzy stanowią istotny podgatunek zwany *Homo informaticus colligens*. Aby *Homo informaticus* mógł robić to, co umie najlepiej, czyli przetwarzać dane, ktoś musi tymi danymi zarządzać, przechowywać je oraz zabezpieczać je przed niepowołanym dostępem. Tym właśnie zajmuje się jego bliski ziomek – *Homo informaticus colligens* (znany także jako *Infocolligens*), o którym jest ta opowieść.

W tym rozdziale zajmiemy się opisem zadań, wyzwań i metod działania *Infocolligensa*. Do gromadzenia i udostępniania danych używa on systemów baz danych. Omówimy różne rodzaje baz danych, od historycznych (choć wciąż jeszcze używanych) do obecnie powszechnie stosowanych, aż po nowinki, które już zadomowiły się w gawrze *Infocolligensa*. Przedstawimy koncepcje baz sieciowych, hierarchicznych i obiektowych, których czas popularności już minął. Zajmiemy się dokładniej obecnie najpowszechniej stosowanymi bazami relacyjnymi i ich podstawowym językiem SQL. Przekonamy się, że mimo całej złożoności, jest to język bardzo użyteczny i ostatecznie nie tak trudny do opanowania. Na końcu opiszemy bazy nurtu NOSQL, które przebojem wdarły się do wielkich firm i mają obecnie bardzo wiele zastosowań: bazy klucz-wartość, dokumentowe i grafowe. Dynamika rozwoju dziedziny baz danych zapiera obecnie dech w piersiach. Może i Ty chcesz wspomóc *Infocolligensa*?

1. Homo informaticus colligens

Homo informaticus będący przedmiotem niniejszego tomu jest silnie uzależniony od symbiozy z pewną odmianą swojego gatunku, egzystującą cichutko na skraju świata technologii informacyjnych (IT). Ową tajemniczą odmianą jest *Homo informaticus colligens*[1], *Homo informaticus* gromadzący (dane). Czymże bowiem zajmuje się *Homo informaticus*? Wykorzystuje lub buduje algorytmy przetwarzające dane. Mogą to być dane (wejściowe), które skądś trzeba pobrać, lub wyniki (dane wyjściowe), które gdzieś należy odesłać w celu wykorzystania lub przechowania do przyszłego użycia. Jakkolwiek można sobie wyobrazić algorytm, który nie potrzebuje danych (np. generator liczb losowych), to algorytm nigdy nieprodukujący żadnych wyników[2] jest całkowicie bezużyteczny. Dane przetwarzane przez *Homo informaticus* trzeba więc dostarczyć, odebrać i przechować. W tym miejscu widać wyraźnie rolę *Homo informaticus colligens* (w skrócie: *Infocolligens*). To właśnie on w złożonym ekosystemie świata technologii informacyjnych zajmuje się składowaniem i udostępnianiem danych.

Ilość danych, jakimi obecnie dysponuje ludzkość, mierzy się setkami eksabajtow. Jeden eksabajt to 2^{60} bajtów, czyli 1 152 921 504 606 846 976, tj. około tryliona bajtów (10^{18} = 1 000 000 000 000 000 000). *Colligo, ergo sum* (pl. gromadzę więc jestem) – to na pewno maksyma wielkich korporacji informatycznych, dysponujących tak ogromnymi zbiorami danych. Głównymi zasobami takich firm jak Google, Facebook, Amazon itd. nie jest już wcale oprogramowanie. O ich pozycji świadczy ilość i jakość zgromadzonych przez nie danych oraz doskonałość metod ich przetwarzania.

W ekosystemach przyrodniczych nadmierne rozmnożenie jednego gatunku stanowi nie lada problem. Zasoby pożywienia są ograniczone, więc równowaga ekosystemu jest po pewnym czasie mniej lub bardziej brutalnie przywracana. O dziwo, tej właściwości nie ma ekosystem IT. Nowi *Homines informatici* zasiedlający ten ekosystem produkują takie ilości nowych danych, że nie może być mowy o ich deficycie. Zadaniem *Infocolligensa* jest znaleźć sposób przechowywania tej rosnącej wciąż liczby. Gawrę *Infocolligensa* zwykliśmy nazywać **bazą danych**. Tam właśnie gromadzi on dane i stamtąd udostępnia je użytkownikom.

[1] Określenie *Homo colligens* zostało zaczerpnięte z książki Claire Chavarin, *Homo colligens: essai sur l'imaginaire de la collection au 19e siecle*, 2001.

[2] Wyniki mogą mieć rozmaitą postać – na przykład informacji ostrzegawczych z oprogramowania monitorującego pewne urządzenie. Gdy urządzenie pracuje poprawnie, żadnych ostrzeżeń nie ma. Wyniki pojawią się, gdy ulegnie ono awarii.

2. Bazy danych

Zawartość i struktura baz danych[3] zmieniała się rozmaicie na przestrzeni ostatnich kilkudziesięciu lat. Jedno pozostało jednak niezmienione – pojęcie **rekordu**, jako podstawowej jednostki danych. Rekord był, jest i zapewne będzie kolekcją **pól**. Każde pole ma nazwę i wartość. W rozmaitych modelach baz danych zmieniały się wymagania odnośnie rekordów co do nazw pól (czy pola w rekordzie mogą się powtarzać?) i typów danych umieszczanych w polach (jak złożone mogą być, czy tylko proste, a może też złożone?). Zmieniały się także dodatkowe cechy, którymi mógł być obdarzony rekord (np. pojęcie sąsiedztwa z innymi rekordami, które służyło nałożeniu na rekordy struktury grafowej). Poszczególne elementy znikały i pojawiały się zależnie od potrzeb i mód. Rekord pozostał jednak zawsze tym samym – zestawem pól. Pod tym względem *Info-colligens* nie zmienił się ani na jotę.

W pierwszych bazach danych, tzw. **sieciowych** i **hierarchicznych**, rekordy mogły być powiązane w sieci lub hierarchie. Każdy rekord oprócz wartości swoich pól (np. rekord z danymi osobowymi zawierał pola, takie jak nazwisko, imię, data urodzenia) mógł mieć także zdefiniowanych sąsiadów (następniki). Przykładowo rekord z danymi osobowymi Kowalskiego był połączony krawędzią z rekordem działu, w którym pracował, czy z rekordem jego szefa. W modelu hierarchicznym posługiwano się pojęciem rekordu podrzędnego (potomnego), a rekordy łączono w hierarchie. Poszczególne rekordy mogą jednak występować w kilku hierarchiach, więc żeby móc je zapisać w hierarchicznej bazie danych, dodano do jej modelu pojęcie **rekordu wirtualnego** znajdującego się w liściu hierarchii. Taki rekord był po prostu traktowany jako odwołanie do dowolnego innego rekordu.

Rekordy o tych samych właściwościach (np. dane osobowe, działy i samochody służbowe) były przechowywane w plikach. **Plik** jest jednocześnie jednostką organizacyjną składowiska fizycznego – dysku lub taśmy. Z bazy danych korzystano metodą „jedna operacja – jeden rekord". Interfejs bazy danych przewidywał następujące instrukcje: weź pierwszy rekord z pliku, weź następny rekord z pliku, weź pierwszego sąsiada (w bazie hierarchicznej było to dziecko), weź kolejnego sąsiada (dziecko). Posługiwano się także indeksami, z tym że operacje były równie jednorekordowe: weź pierwszy rekord o danej wartości klucza wyszukiwania, weź kolejny itd.

[3] Spośród książek dostępnych w języku polskim najwięcej informacji o bazach danych można znaleźć w klasycznym podręczniku [2].

IMS (ang. *Information Management System*) firmy IBM to jeden z pierwszych hierarchicznych systemów baz danych. Prace nad nim rozpoczęto w roku 1966, a pierwszą wersję uruchomiono w 1968 roku. System ten nadal jest pielęgnowany i użytkowany. Dlaczego nie? Przecież działa. Należy pamiętać, że lepsze jest wrogiem dobrego.

Przełom zaczął się w roku 1970, gdy Edgar Frank Codd przedstawił relacyjny model danych. Zaproponował on uproszczenie rekordów poprzez likwidację struktury grafowej ponad nimi. Okazało się, że mniej znaczy więcej. Dzięki temu uproszczeniu można było zdefiniować nowy abstrakcyjny interfejs do bazy danych działający na zasadzie jedna operacja – jedna relacja. **Relacja** to nic innego jak kolekcja takich uproszczonych rekordów. Operowanie na relacjach, a nie na pojedynczych rekordach było strzałem w dziesiątkę, m.in. ze względu na właściwości dysków magnetycznych. Wirujące dyski magnetyczne mają bowiem to do siebie, że odczyt pojedynczego sektora zajmuje niemal tyle samo czasu co odczyt całego cylindra[4]. Przetwarzanie danych całymi relacjami (tj. dużymi zbiorami rekordów) umożliwia wykorzystanie tej właściwość dysków. Od roku 1970 *Homo informaticus colligens* ma nad biurkiem transparent *Interfejs, głupcze!* Przemysł zaś już od roku 1986 przyjął oficjalny standard języka zapytań dla relacyjnych baz danych. Jest nim **SQL** (ang. *Structured Query Language*), o którym więcej piszemy w dalszej części tego rozdziału. SQL to właśnie język przetwarzania relacji, a nie rekordów.

Pierwszy relacyjny system zarządzania bazą danych (SZBD) powstał też w laboratoriach IBM w roku 1973. Dalej w tym rozdziale znajdują się informacje o jego optymalizatorze zapytań. Algorytmy zaimplementowane w tym optymalizatorze nadal są wykorzystywane we współczesnych bazach danych. Pierwszy komercyjny system relacyjny zbudowano w roku 1979 w firmie Relational Software, która obecnie nosi nazwę Oracle.

Z kolei nad drzwiami *Infocolligensa* wisi napis *Abstrakcja, głupcze!* Pojawienie się relacyjnych baz danych poskutkowało wprowadzeniem **fizycznej niezależ-**

[4] Cylinder składa się ze wszystkich sektorów wszystkich ścieżek o takim samym rzucie na podstawę urządzenia dyskowego.

ności danych. Struktury fizyczne (pliki) zostały oderwane od struktur logicznych (relacji). Projektant bazy danych tworzy jej schemat w postaci zestawu relacji; programista aplikacyjny implementuje aplikację odwołując się do relacji. Pliki natomiast są dla niego niewidoczne. Nie zna ich położenia (mogą być nawet na zdalnej maszynie) oraz przypisania plików do relacji. Dzięki temu projektant bazy danych, a później jej administrator, może przenosić dane z jednego fizycznego położenia do innego bez wpływu na działające aplikacje.

Wspomniane powyżej oddzielenie aspektów fizycznych od logicznych ma jeszcze jedną kluczową konsekwencję. Jest nią możliwość zdefiniowania języka zapytań abstrahującego od fizycznych struktur składowania i konkretnych algorytmów przetwarzania danych. O tym wspominamy w następnym punkcie.

Pierwszą wersję języka SQL opublikowano w roku 1986, a następne wersje pojawiały się w latach 1989, 1992, 1999, 2003, 2006 i 2008. Część z nich zawierała przełomowe zmiany. Wersja SQL:1999 to już pełny język programowania, w których można zapisać dowolny algorytm, wcześniej ten język miał istotne ograniczenia siły wyrazu. Ta wersja to też pierwszy standard obiektowo-relacyjny. Z kolei w języku SQL:2003 uwzględniono obsługę danych w formacie XML.

2.1. Obiektowość w bazach danych

Wzrost popularności programowania obiektowego sprawił, że zaczęto zastanawiać się nad nowym modelem danych. Skoro aplikacje przetwarzają obiekty, to dlaczego baza danych ma składować krotki relacji? Odpowiedzią na to pytanie było pojawienie się **obiektowych baz danych**. Rekord w takiej bazie jest obiektem, który może mieć złożone typy pól oraz pola referencyjne odwołujące się do innych obiektów. Tak jak w programowaniu obiektowym, obiekty-rekordy mogą mieć metody. W latach 90. XX wieku powstały liczne obiektowe systemy zarządzania bazami danych (**OSZBD**), a nawet pierwsza propozycja standardu wydana przez konsorcjum ODMG (rozwiązane w roku 2001). Po kilkunastu latach rozwoju baz obiektowych zainteresowanie nimi świata naukowego osłabło i nie zdobyły one dużego udziału w rynku. Przyczyn doszukiwano się rozmaitych. Mogły nimi być słabość kapitałowa i technologiczna firm wytwarzających OSZBD, słabość merytoryczna konsorcjum standaryzacyjnego czy mniejsza od oczekiwań wygoda stosowania OSZBD. Przecież miały one ułatwić życie programistom, więc dlaczego były o niebo mniej przyjazne niż systemy relacyjne? Nie bez zna-

czenia jest też duże przywiązanie obiektowych produktów bazodanowych do konkretnych języków programowania.

W roku 2006 konsorcjum OMG (ang. *Object Management Group*, m.in. właściciele standardu UML) próbowało ożywić koncepcję standardu obiektowych baz danych. Powołano odpowiednią grupę roboczą. Zebrano ze świata propozycje i po kilku spotkaniach zdecydowano się przyjąć **architekturę stosową** Kazimierza Subiety [4] za punkt wyjścia do definicji nowego standardu. Ten duży sukces polskiej koncepcji niestety spotkał się z bojkotem wielkim firm informatycznych i niechęcią środowiska programistów, jako coś zupełnie nowego i zupełnie nieznanego. Grupa robocza przestała się spotykać i obecnie słuch o niej zaginął.

Reakcja wytwórców relacyjnych SZBD miała dodatkowe ostrze. Była nim koncepcja obiektowo-relacyjnego modelu danych oraz odwzorowania obiektowo-relacyjne. Nowy model danych miał na celu pogodzenie ognia z wodą, bo oba modele mają raczej mało wspólnego ze sobą. Wynikiem jest m.in. nowy standard SQL. Począwszy od SQL:1999, ten podstawowy język programowania baz danych nie jest już relacyjny, a obiektowo-relacyjny. Wtręty obiektowe do SQL okazały się umiarkowanym sukcesem i nie są zbyt powszechnie używane. Z drugiej strony systemy odwzorowania obiektowo-relacyjnego (np. system Hibernate) podbiły umysły wielu programistów i architektów oprogramowania. Systemy takie umożliwiają tworzenie aplikacji obiektowo, zakładając że wskazane obiekty będą trwale składowane w jakiejś bazie danych. Dostarczają także mechanizmów odwzorowania danych obiektowych na relacje bazy danych. Programista definiuje więc tylko obiekty, które chce mieć składowane w bazie, a system odwzorowania martwi się już o wszystko inne. Hibernate i inne systemy tego typu sprawiają, że obiektowa baza danych nie wydaje się już ani trochę atrakcyjna.

Na przełomie wieków XX i XXI wydawało się, że to już koniec historii rozwoju baz danych. Relacyjne bazy danych miały być narzędziem do składowania i przetwarzania wszelkich danych. Kto tak myślał, pomylił się tak samo jak Francis Fukuyama ogłaszający koniec historii w polityce. Jego teorie runęły razem z wieżami WTC tragicznego 11 września 2001 roku. Niestety mniej więcej w tym samym czasie na przełomie wieków doszło też do dramatycznych zmian w dziedzinie baz danych. Okazało się, że relacyjne bazy danych i SQL nie rozwiązują wszystkich problemów. Największym z nich była skalowalność. W roku 2000 Eric Brewer sformułował tzw. hipotezę CAP (ang. *Consistency-Availability--Partition tollerance*), według której nie jest możliwe uzyskanie naraz spójności danych (C), pełnej dostępności (A) i poprawności partycjonowania (P). Możliwe są kombinacje jedynie dwuliterowe. Hipoteza Brewera została udowodniona w roku 2002 i nosi teraz zasłużone miano **twierdzenia CAP**.

Konsekwencją potrzeb *Homo informaticus colligens* oraz prawdziwości CAP jest nowy prąd w dziedzinie baz danych, tzw. **ruch NOSQL**. Należy go rozumieć, jako *Not-Only-SQL* (nie tylko SQL), a nie *No-SQL* (tylko nie SQL!), ponieważ w wielu zastosowaniach klasyczne relacyjne bazy danych są nadal najlepszym narzędziem. Najbardziej znaczącymi systemami baz danych NOSQL są: Big-Table, Cassandra czy HBase. W ten sposób *Infocolligens* przeszedł od ery *one size fits all* (do wszystkiego jest dobry jeden „rozmiar" – relacyjna baza danych) do ery *right tool for the job* (do każdego zadania wybierz najlepsze narzędzie: bazę relacyjną lub NoSQL), [3].

Konsorcjum ODMG zawiązano w 1991 roku. W czasie swojej działalności ODMG wydało pięć wersji standardu dla obiektowych baz danych. Niestety nigdy nie stał się on równoważnikiem SQL i zebrał wiele krytycznych ocen. W roku 2001 konsorcjum ogłosiło swój sukces i zostało rozwiązane, gdyż po takim sukcesie nie pozostało już nic do roboty. Sukces był niestety wyłącznie marketingowy, bo standardy ODMG nigdy nie zostały szeroko rozpowszechnione ani zaimplementowane.

3. Relacyjne bazy danych

Od wielu lat gawrę *Infocolligensa* wypełnia głównie serwer relacyjnej bazy danych, chociaż, jak już wiemy, nowe elementy zaczynają się też tam pojawiać. Relacyjna bazy danych jest oparta na pojęciu relacji, pochodzącym z teorii zbiorów (teorii mnogości). Szczęśliwie można jednak objaśnić jej działanie bez odwoływania się do matematyki. Zaczniemy od tego, że zamiast o relacjach będziemy mówić o **tabelach**. Relacyjna baza danych przechowuje pewną liczbę nazwanych tabel. Każda tabela ma stałą liczbę nazwanych kolumn. Dla każdej kolumny jest określony prosty typ danych, które można w nich umieścić (np. napis, liczba, data). Zestaw nazw tabel, nazw ich kolumn oraz ich typów nazywamy **schematem** bazy danych. Dane mają więc postać wierszy w tabelach i w każdym wierszu jest podana wartość w każdej kolumnie[5].

[5] W standardzie SQL tą wartością może być NULL – wartość pusta lub nieokreślona. Ma ona masę paskudnych właściwości włącznie z tym, że jej pojawienie się wprowadza nas w świat logiki trójwartościowej. NULL to „nie-wiadomo-co", więc porównanie czegokolwiek z NULL daje nam trzecią wartość logiczną „nie-wiadomo-czy" – ani fałsz, ani prawda. W tym rozdziale zbywamy NULL milczeniem tak, jak na to zasługuje.

Osoby			
Id	Imię	Nazwisko	Wiek
1	Adam	Stawski	17
2	Lech	Boruta	13
3	Anna	Welke	14
4	Agnieszka	Nguyen	14
5	Teodor	Markowetz	15

Hobby			
IdOsoby	Nazwa	Początek	Koniec
1	Szachy	2000	9999
1	Brydż	2007	9999
1	Wyścigi	2003	2005
3	Szachy	2009	9999
3	Brydż	1999	2008
4	Filatelistyka	1987	9999
c5	Filatelistyka	1987	1994
5	Brydż	1991	9999
5	Filatelistyka	2000	9999

Znajomi				
IdOsoby	IdZnajomego	Sposób	Początek	Koniec
1	2	kolega	1998	9999
2	1	kolega	1998	9999
1	3	ziom	2004	9999
3	1	ziom	2004	9999
1	5	kolega	2011	9999
2	4	kolega	2002	2007
4	5	przyjaciel	2003	2005
5	4	przyjaciel	2003	2005

Rysunek 1. Trzy przykładowe tabele bazy danych dla sieci społecznościowej

Na rysunku 1 pokazano przykład uproszczonej bazy danych dla sieci społecznościowej. Jeśli podczas lektury na temat historii baz danych zastanawiałeś się, co stało się z zależnościami między rekordami, to teraz masz już odpowiedź. Relacyjna baza danych nie oznacza, że nie można zapisać informacji o takich zależnościach. Po prostu są one reprezentowane inaczej. Tabele mogą (a nawet powinny) mieć tzw. **klucz główny**, czyli jedną kolumnę (lub ich zestaw) jednoznacznie identyfikującą każdy rekord. Klucz w tabeli `Osoby` składa się jedynie z kolumny `Id`. W dwóch pozostałych tabelach rzecz jest bardziej złożona. Klucz główny tabeli `Hobby` musi składać się aż z trzech kolumn: `IdOsoby`, `Nazwa` i `Początek`. Wydawać by się mogło, że wystarczą tylko dwie pierwsze kolumny. To jednak nie wystarczy, może się bowiem zdarzyć, że ktoś porzuci pewnego dnia swoje hobby, aby po latach do niego wrócić. Będzie miał wtedy w tabeli `Hobby` dwa rekordy ze swoim identyfikatorem i nazwą zainteresowania. Nie byłoby to możliwe przy zbyt ubogim kluczu głównym. W bazie ma rysunku 1 nie mogłyby więc istnieć dwa rekordy określające zainteresowania Teodora Markowetza (`Id = 5`) filatelistyką. Kolumna `Początek` musi być zatem dodatkowym elementem klucza głównego. Podobnie jest z tabelą `Znajomi`. Tu też może się zdarzyć, że ktoś kogoś przestanie lubić, a potem polubi ponownie. Lekarstwem na to jest znów dodanie kolumny `Początek` do klucza głównego, który w tej tabeli będzie składał się jeszcze z kolumn `IdOsoby` i `IdZnajomego`.

Tabele z rysunku 1 są ilustracją możliwości przechowywania **danych temporalnych**, tzn. takich, które zmieniają się w czasie. Każdy fakt w tabelach `Hobby` i `Znajomi` ma przypisany okres swojego obowiązywania. Zauważmy, że sprytnie uniknęliśmy konieczności używania wartości pustej. Jeśli jakiś fakt jest nadal prawdą, to za koniec jego ważności uznajemy dostatecznie odległą datę. Przy dzisiejszym postępie technologicznym możemy być pewni, że nasza baza danych nie przetrwa najbliższych dwudziestu lat bez gruntownej modernizacji, nie mówiąc już o roku 9999[6].

Przyjrzymy się teraz możliwościom przetwarzania danych, jakie dają relacyjne bazy danych i wspomniany już język SQL na kilku przykładach zapytań. W założeniach, język SQL miał być przyjaznym i podobnym do naturalnego sposobem zapisywania tego, *co* ma być wynikiem zapytania, a nie *jak* ten wynik należy obliczyć. W jakim stopniu SQL spełnia pokładane w nim nadzieje pozostawiamy do indywidualnej oceny. Osobiście, autor ma wątpliwości co do jego przyjazności. Udało się natomiast doskonale oderwać w SQL realizację zapytań od konkretnych algorytmów. Nawet te najprostsze zapytania mają w SQL po kilka możliwych sposobów realizacji, a te bardziej złożone mają tak

[6] Warto pamiętać o skali czasu. Tak zwana pluskwa milenijna w roku 2000 nie wyrządziła żadnych szkód, podczas gdy sekunda przestępna dodana do zegarów ostatniego dnia czerwca 2012 roku spowodowała awarię licznych serwisów internetowych.

wiele realizacji, że wybór jednej właściwej staje się problemem. Zajmiemy się tym w punkcie 5.

Rozważmy pierwsze zapytanie, które ma wypisać wszystkie kolumny wierszy tabeli Osoby, w których w kolumnie Wiek jest wartość 14. Klauzula SELECT wskazuje kolumny, które mają być wypisane. Gwiazdka w klauzuli jest żądaniem wypisania wszystkich kolumn. Klauzula FROM określa, z których tabel należy pobrać wiersze, a WHERE – zawiera warunek filtrujący wiersze. Zostaną wypisane tylko te wiersze tabeli, które spełniają formułę z klauzuli WHERE. Prawda, że proste?

```
SELECT *
FROM Osoby
WHERE Wiek = 14;
```

A gdybyśmy chcieli wypisać imiona i nazwiska osób mających tyle samo lat co osoba nazwiskiem Nguyen? Proszę bardzo! SQL jest kompozycyjny. Zamiast liczby 14 wystarczy wstawić zapytanie obliczające identyfikator odpowiedniej osoby[7]:

```
SELECT o.Imię, o.Nazwisko
FROM Osoby o
WHERE Wiek = (SELECT Id FROM Osoby WHERE Nazwisko = 'Nguyen');
```

Możemy też zapytać o obecne (tj. w roku bieżącym) hobby każdej osoby. W tym celu trzeba pobrać dane z kilku tabel naraz. Aby tego dokonać, należy w klauzuli FROM podać tabele będące źródłami danych. Zapytanie takie wypisuje wynik tak, jakby brało pod uwagę wszystkie możliwe *n*-tki (w tym przykładzie: pary) wierszy z podanych tabel i następnie filtrowało je pozostawiając tylko te, które spełniają warunek WHERE. Wygląda to więc tak, że bierzemy pod uwagę wszystkie możliwe pary osoba-hobby i potem pozostawiamy tylko te, w których między wartościami kolumn Początek i Koniec mieści się 2012 oraz mają takie same wartości w kolumnach Id w tabeli Osoba i IdOsoby w Hobby.

```
SELECT o.Imię, o.Nazwisko, h.Nazwa
FROM Osoby o, Hobby h
WHERE o.Id = h.IdOsoby
   AND 2012 BETWEEN h.Początek AND h.Koniec;
```

[7] Oczywiście, jeśli osób o nazwisku Nguyen jest w tabeli więcej, to wykonanie zapytania zakończy się sygnalizacją błędu. Aby tego uniknąć, można np. wziąć minimalny identyfikator wśród wszystkich tych osób MIN(Id).

Na potrzeby tego zapytania, w klauzuli `FROM` wprowadzono **aliasy** `o` i `h`, zwane też **zmiennymi krotkowymi**, czyli nowe nazwy tabel `Osoby` i `Hobby`. W ramach tego zapytania należy posługiwać się już tylko nazwami `o` i `h`. Można by nie wprowadzać aliasów, jednak ze względu na ewentualne przyszłe zmiany zestawu kolumn w tabelach (tzw. **ewolucję schematu**), zaleca się programistom aplikacyjnym aliasowanie wszystkich tabel.

Obliczenie wyniku zapytania przez system zarządzania bazą danych oczywiście nie polega na wyznaczeniu najpierw wszystkich możliwych par wierszy tych dwóch tabel występujących w zapytaniu (czyli ich iloczynu kartezjańskiego). Dlaczego „oczywiście"? Czasowa złożoność obliczeniowa takiego algorytmu byłaby bowiem funkcją kwadratową, a dokładnie, byłaby iloczynem wielkości obu tabel. Złożoność kwadratowa jest jednak jednym z największych wrogów efektywności obliczeń prowadzonych na bazach danych. Algorytmy stosowane w mechanizmach baz danych muszą mieć złożoność linową ($O(n)$) lub ewentualnie liniowo-logarytmiczną ($O(n\log n)$), gdzie n jest łączną liczbą wierszu w przetwarzanych tabelach. Aparat wykonawczy zapytań omówiono w punkcie 5.

Język SQL umożliwia też wyliczanie pewnych podsumowań. Oto zapytanie, które dla każdej osoby wyznacza liczbę jej rożnych hobby, uwzględniając także chwilowe zainteresowania. W poniższym zapytaniu jest wykonywane standardowe, poznane wcześniej złączenie tabel `Osoby` i `Hobby`, a następnie grupowanie `GROUP BY` par wierszy według wartości kolumn `Id`, `Imię` i `Nazwisko`. W jednej grupie wszystkie pary wierszy mają te same wartości kolumn grupujących. Jedna grupa w tym zapytaniu odpowiada jednej osobie. Następnie w ramach każdej grupy wyznaczana jest liczba rożnych (`DISTINCT`) wartości kolumny `Nazwa` z tabeli `Hobby`. Przed obliczeniem tego podsumowania należy usunąć duplikaty wśród nazw hobby, ponieważ ktoś mógł mieć jakieś hobby, następnie je porzucić i później do niego wrócić. W takim przypadku będzie miał więcej wpisów w tabeli `Hobby` dla jednego swojego zainteresowania, a zatem bez usunięcia duplikatów wynik zapytania byłby niepoprawny. Na końcu zapytania wynik jest sortowany malejąco (`ORDER BY... DESC`) względem liczby różnych hobby, tj. kolumny wyniku `Z`:

```
SELECT o.Id, o.Imię, o.Nazwisko, COUNT(DISTINCT h.Nazwa) Z
FROM Osoby o, Hobby h
WHERE o.Id = h.IdOsoby
GROUP BY o.Id, o.Imię, o.Nazwisko
ORDER BY Z DESC;
```

Zajmijmy się teraz społecznościowym aspektem bazy danych z rysunku 1. Baza ta dopuszcza asymetrię znajomości, tzn. `Adam` może znać `Teodora`, podczas

gdy Teodor może nie znać Adama lub się do tego nie przyznaje. Poniżej znajduje się zapytanie, którego celem jest wyszukanie osób mających obecnie (tj. w 2012) znajomych, którzy nie są ich znajomymi. Tym razem skorzystamy z tabeli Znajomi i złączymy ją dwa razy z tabelą Osoby. Pierwszy raz (o1) będzie to osoba, której znajomych szukamy. Drugi raz (o2), to ów poszukiwany znajomy, nieuznający znajomości z o1. Mamy tu do czynienia z tzw. **samozłączeniem tabel**. Bez aliasów nie udałoby się tego zapisać, gdyż musimy w jakiś sposób rozróżnić te dwie role osób.

```
SELECT DISTINCT o1.Id, o1.Imię, o1.Nazwisko
FROM Osoby o1, Osoby o2, Znajomi z
WHERE o1.Id = z.IdOsoby
    AND o2.Id = z.IdZnajomego
    AND 2012 BETWEEN z.Początek AND z.Koniec
    AND NOT EXISTS (SELECT 1
                    FROM Znajomi ze
                    WHERE ze.IdOsoby = o2.Id
                        AND ze.IdZnajomego = o1.Id
                       AND 2012 BETWEEN ze.Początek AND ze.Koniec);
```

Zapytanie to polega na wyznaczeniu znajomości z osoby o2 przez osobę o1, a następnie stwierdzeniu, że nie istnieje znajomość ze osoby o1 z osobą o2. Ponownie podkreślmy, że jest to tylko opis tego, co zapytanie ma wypisać. Jak poradzi sobie z tym system zarządzania bazą danych, to już całkiem inna historia. Wcale nie musi on wykonywać czynności w taki sposób, który jest dla człowieka najwygodniejszy do opisu semantyki zapytania!

Kolejne zapytanie służy często w portalach społecznościowych do sugerowania nowych znajomości. Znajduje ono pary osób, które same nie są znajomymi, ale mają wspólnego znajomego. To zapytane jest podobne do poprzedniego. Tym razem jednak łączymy aż pięć egzemplarzy tabel, trzy razy tabelę Osoba i dwa razy tabelę Znajomi. Chcemy, by istniała znajomość z12 osoby o1 z osobą o2 oraz znajomość z23 osoby o2 z osobą o3, ale zabronione jest istnienie znajomości z13 między osobami o1 i o3:

```
SELECT DISTINCT o1.Id, o1.Imię, o1.Nazwisko,
                  o3.Id, o3.Imię, o3.Nazwisko
FROM Osoby o1, Osoby o2, Osoby o3, Znajomi z12, Znajomi z23
WHERE o1.Id = z12.IdOsoby
    AND o2.Id = z12.IdZnajomego
```

```
    AND o2.Id = z23.IdOsoby
    AND o3.Id = z23.IdZnajomego
    AND o1.Id!= o3.Id
    AND NOT EXISTS (SELECT 1
                    FROM Znajomi z13
                    WHERE z13.IdOsoby = o1.Id
                        AND z13.IdZnajomego = o3.Id);
```

Tym razem naiwne obliczenie wyniku tego zapytania na pewno nie wchodzi w rachubę. Polegałoby ono na wyznaczeniu wszystkich piątek, a następnie wyselekcjonowaniu tych, które są złożone z pasujących do siebie elementów. To oznaczałoby ogromną złożoność $O(o^3z^2)$, gdzie o to liczba wierszy w tabeli `Osoba`, a z to liczba wierszy w tabeli `Znajomi`. System zarządzania bazą danych musi takie zapytania wykonywać znacznie szybciej.

Ostatnie z przykładowych zapytań jest ilustracją możliwości języka SQL, która pojawiła się dopiero w roku 1999, czyli 13 lat po ukazaniu się pierwszej wersji standardu tego języka. Chodzi tutaj o możliwość rekurencyjnego wyszukiwania. Wynikiem poniższego zapytania ma być krąg przyjaciół osoby o identyfikatorze 1, ich przyjaciół, ich przyjaciół przyjaciół itd., aż do szóstego poziomu.

```
WITH RECURSIVE Krąg (Id, Imię, Nazwisko, Poziom) AS
(
    SELECT o.Id, o.Imię, o.Nazwisko, 0
    FROM Osoby o
    WHERE o.Id = 1
UNION ALL
    SELECT o.Id, o.Imię, o.Nazwisko, k.Poziom + 1
    FROM Krąg k, Osoby o, Znajomi zko
    WHERE k.Id = zko.IdOsoby
        AND o.Id = zko.IdZnajomego
        AND 'przyjaciel' = zko.Sposob
        AND k.Poziom < 6
)
SELECT *
FROM Krag;
```

W tym zapytaniu jest budowana pomocnicza relacja `Krąg`, która w końcowym kroku zapytania jest po prostu wypisywana jako wynik tego zapytania (ostatnia

w zapytaniu klauzula SELECT). Na początku relacja Krąg jest zbiorem pustym. Pierwsze podzapytanie SELECT znajduje osobę o identyfikatorze 1 zaznaczając, że jej poziom to 0 i dodaje ją do relacji Krąg. Drugie podzapytanie SELECT, iteracyjnie dodaje nowych przyjaciół do już istniejącego kręgu. Korzysta się bowiem z już znalezionych wierszy tabeli Krąg i łączy istniejących członków kręgu k znajomością zko z osobą o. Nowo dodana osoba ma poziom o jeden większy niż jej poprzednik. Obliczenia te są prowadzone aż do wyczerpania wszystkich możliwości, to jest do czasu, aż to drugie podzapytanie SELECT zwróci pusty wynik. Kiedyś musi to nastąpić, ponieważ każde obliczenie rekurencyjne zwiększa poziom o jeden, a jest ograniczenie, że poziom krotki k musi być mniejszy niż 6.

Możliwości takich rekurencyjnych zapytań są niestety bardzo ograniczone. Nie możemy bowiem sprawdzić, czy nowy element relacji rekurencyjnej nie został już wcześniej wygenerowany. Bardzo łatwo jest więc napisać zapytanie rekurencyjne, które się zapętli. Aby tego uniknąć, można ewentualnie dodać kolumnę licznikową, by ograniczyć liczbę wywołań rekurencyjnych (jak w powyższym przykładzie).

4. Transakcje

Infocolligens dysponuje zatem wygodnym językiem do pobierania danych z bazy, choć jak wynika z ostatniego przykładu, język SQL ma istotne ograniczenia. Baza danych musi także zapewniać bezpieczeństwo danych na wypadek awarii lub dostępu do danych przez wielu użytkowników bazy równocześnie. Dwóch użytkowników modyfikujących te same dane w tym samym czasie może doprowadzić do ich zniszczenia. W praktyce takie sytuacje traktuje się jako szczególny rodzaj awarii.

Aby radzić sobie z awariami systemów zarządzania bazami danych, *Infocolligens* posługuje się pojęciem **transakcji**, czyli zestawu operacji wykonywanych na bazie danych o pewnych szczególnych właściwościach, opisywanych angielskim skrótem **ACID** (ang. *Atomicity-Consistency-Isolation-Durability*). **Atomowość** (**A**) oznacza, że transakcja jest wykonywana jako całość, nie może być zrealizowana częściowo: albo kończy się sukcesem (zatwierdzeniem) albo porażką (wycofaniem). Wycofanie oznacza, że transakcja w ogóle nie miała miejsca i w danych nie pozostaje po niej żaden ślad. **Spójność** (**C**) to wymaganie, aby transakcja, która zastanie bazę danych w stanie spójnym, po swoim zakończeniu również pozostawiła spójny stan danych. **Spójna baza danych** zawiera wyłącznie dane poprawne, tj. zgodne z rzeczywistością. Dozwolone są stany niespójne, ale nie mogą takie pozostać po zatwierdzeniu transakcji. **Izolacja** (**I**) to zbudowana na użytek jednej transakcji iluzja, że ta transakcja ma całą bazę danych wyłącznie do swojej dyspozycji i jest całkowicie odizolowana od pozo-

stałych transakcji, jej działanie nie zależy więc od nich. Najwyższą formą izolacji jest **szeregowalność**. System zarządzania bazą danych może dopuszczać pewne przeploty operacji rożnych transakcji, ale tylko takie, których wynik końcowy będzie taki sam jak przy pewnym szeregowym (niewspółbieżnym) wykonaniu tych transakcji. Ten porządek szeregowy ma istnieć, ale nie jest określone, jaki ma być. Na koniec **trwałość** (**D**) oznacza, że dane wpisane przez zatwierdzoną transakcję mają znaleźć się w bazie nawet po awarii SZBD, nośnika danych itd. Jeśli więc transakcja zakończy się sukcesem, nic nie może stać się jej wynikom.

System zarządzania bazą danych realizuje atomowość i trwałość prowadząc dzienniki. **Dziennik powtórzeń** zawiera zapis wszystkich operacji modyfikujących dane i jest zrzucany fizycznie na dysk po każdym zatwierdzeniu transakcji. Dla bezpieczeństwa może być też replikowany. Gdy SZBD jest uruchamiany, sprawdza, czy został czysto zamknięty. Jeśli wykryje, że jego działanie zostało przerwane awarią, przystępuje do **odtwarzania**, tj. ponawia operacje zapisane w dzienniku od ostatniego czystego zamknięcia bazy, a ściślej od tzw. **punktu kontrolnego**. **Dziennik wycofań** zawiera przepisy na wycofanie poszczególnych operacji i jest wykorzystywany do wycofania operacji nieudanych transakcji. Gdy transakcja się wycofuje (po awarii i wznowieniu dotyczy to wszystkich aktywnych transakcji), dziennik wycofań jest czytany od tyłu i stosowane są zapisane w nim operacje anulujące. Izolację realizuje się poprzez stosowanie blokad/zamków. Gdy transakcja działa na jakiejś danej, zakłada na nią zamek, który wyklucza korzystanie z tych samych danych przez inną transakcję. To może powodować **zakleszczenia**, gdy dwie transakcje trzymają zamki na dwóch danych i obie czekają na daną akurat zamkniętą przez tę drugą transakcję. Wymaganie spójności implementuje się poprzez kontrolę **więzów integralności**, tj. warunków poprawności danych, najpóźniej przy zatwierdzeniu. Wymaganie unikatowości klucza głównego, poprawności klucza obcego, czy nieujemność wieku osoby to przykłady takich więzów.

Widać więc, że zabawki *Homo informaticus colligens* są bardzo złożone. Ich implementacja wymaga wiedzy z każdej dziedziny praktycznej informatyki: budowy sprzętu, systemów operacyjnych, programowania współbieżnego, kompilatorów (zapytania trzeba analizować składniowo i wykonywać), struktur danych, logiki, algorytmiki i sztucznej inteligencji (przy przetwarzaniu zapytań; por. punkt 5).

5. Przetwarzanie zapytań, optymalizator i adaptacyjność

Dla każdego *Homo informaticus*, a więc także dla *Infocolligensa* zdrowe lenistwo infrastruktury komputerowej jest jedną z najważniejszych wartości. Wcale nie chodzi o to, aby komputer się napracował, ale żeby dostarczył poprawny

wynik obliczeń. Im mniej kroków wykona i zasobów zużyje, tym lepiej. W bazie danych mamy jednak do czynienia z inną sytuacją niż w przypadku tradycyjnego programowania. Podstawowy język baz danych, SQL, określa bowiem to, *co* ma być zwrócone, a nie, *jak* to obliczyć. Dla jednego zapytania może istnieć bardzo wiele algorytmów wyliczających poprawnie jego wynik. Przetwarzanie zapytania zaczyna się więc od wyznaczenia planu (algorytmu). Następnie ten plan jest realizowany. Zanim jednak do tego dojdzie, system zarządzania bazą danych robi wszystko, aby jak najmniej się napracować. SZBD w swoich strukturach danych zapamiętuje plany pewnej liczby wykonanych uprzednio zapytań, a także wyniki niektórych z nich. Wybór wyników do zapamiętania odbywa się heurystycznie w ramach dostępnych zasobów (głównie pamięci operacyjnej). Tu po raz pierwszy spotykamy się ze sztuczną inteligencją SZBD.

5.1. Przed poszukiwaniem planu wykonania

Zanim jednak SZBD przystąpi do wyboru planu weryfikuje, czy w ogóle musi to robić. Na początku sprawdza więc, czy otrzymane zapytanie nie zostało już wcześniej zapamiętane. Jeśli ma jego aktualny wynik, to po prostu go odsyła. Jeśli ma jego wcześniej wyznaczony plan, przystępuje do wykonania tego planu. Tożsamość zapytań jest sprawdzana literalnie, co do znaku. W praktyce oblicza się wartość pewnej funkcji haszującej z tekstu zapytania. Programista aplikacyjny nie powinien więc wpisywać wartości stałych do zapytania.

Pierwsze przykładowe zapytanie z tego rozdziału jest zatem niewłaściwe:

```
SELECT * FROM Osoby WHERE Wiek = 14;
```

Programista powinien użyć tzw. **zmiennej wiązania** (parametru poprzedzanego dwukropkiem) i przy wywołaniu tego zapytania podać wartość `:W`:

```
SELECT * FROM Osoby WHERE Wiek = :W;
```

Osiągamy dzięki temu dwa cele. Po pierwsze SZBD może to zapytanie rozpoznać jako tożsame, gdy zostanie ono wysłane wielokrotnie, i unikać w ten sposób kosztownego szukania planu. Po drugie uniemożliwiamy w ten sposób ataki **wstrzykiwania SQL** (ang. *SQL injection*), które polegają na wpisywaniu do formularza WWW fragmentów zapytań z użyciem znaków specjalnych, głównie apostrofów i odwrotnych ukośników. W wyniku takiego ataku znaczenie zapytań przekazywanych do bazy danych z aplikacji WWW może ulec modyfikacji, co z kolei może prowadzić do uszkodzenia danych lub niepożądanego ujawnienia tajemnic.

Niestety, nauczenie programistów aplikacyjnych tak prostej rzeczy, jak nieużywanie stałych w zapytaniach, okazuje się być zadaniem niewykonalnym. Z tego powodu w SZBD wprowadzano pewne udogodnienie. Po włączeniu odpowiedniej opcji konfiguracyjnej, SZBD analizuje wstępnie wszystkie przychodzące zapytania i zamienia w nich stałe na zmienne wiązania. To oczywiście nie rozwiązuje problemu ataku wstrzykiwanym SQL, ale za to zwiększa ponowne użycie raz wyznaczonych planów. Nie odbywa się to jednak bez kosztów. Po pierwsze SZBD wykonuje dodatkowe czynności przy przetwarzaniu zapytania, co powoduje zużycie zasobów. Po drugie i znacznie ważniejsze, tracimy w ten sposób pewną możliwość. Może się bowiem zdarzyć, że dla pewnych wartości parametrów chcemy planu wykonania innego niż dla pozostałych. Taka sytuacja ma miejsce, gdy dane w jakiejś kolumnie są rozłożone nierównomiernie (np. w bazie danych polskich żołnierzy pole płeć będzie zawierało zdecydowanie więcej liter M niż K). Bez włączenia opcji usuwania stałych, po prostu użyjemy parametru dla wartości standardowych, a stałej dla wartości szczególnych. Po włączeniu tej opcji, cały ten wysiłek jest jałowy. SZBD wszystkie wartości potraktuje tak samo.

5.2. Algebra relacji

Zajmijmy się teraz sytuacją, w której SZBD nie ma gotowego planu i musi go utworzyć. Proces ten rozpoczyna się od analizy składniowej zapytania, która prowadzi do wytworzenia wstępnego planu logicznego. Taki plan to drzewo wyrażenia **algebry relacji**, która stanowi podstawę teoretyczną przetwarzania zapytań w bazach danych. Nośnikiem tej algebry są skończone relacje, a działaniami są następujące operatory. W nawiasach podano ich tradycyjnie używane symbole.

Selekcja(σ) to wybór pewnego podzbioru wierszy z zadanej tabeli na podstawie warunku logicznego.
Rzutowanie(π) to wybór pewnego podzbioru kolumn z zadanej tabeli.
Zmiana nazw(ρ) kolumn w zadanej tabeli.
Iloczyn kartezjański($\times$) dwóch tabel to zbiór wszystkich możliwych par ich wierszy.
Złączenie($\bowtie$) dwóch tabel to zbiór wszystkich pasujących do siebie par ich wierszy. Dopasowanie wierszy jest zadane formułą logiczną.
Suma($\cup$) mnogościowa dwóch tabel.
Różnica($-$) mnogościowa dwóch tabel.
Przecięcie($\cap$) mnogościowe dwóch tabel.

Ten zestaw operatorów jest nadmiarowy, ponieważ przecięcie jest szczególnym przypadkiem złączenia. Iloczyn kartezjański to złączenie z warunkiem zawsze prawdziwym. Z kolei złączenie to złożenie iloczynu kartezjańskiego, zmiany nazw i selekcji. Można by więc ten zbiór operatorów ograniczyć do sze-

ściu. Język SQL wymaga też dodatkowych operacji niedefiniowalnych w algebrze relacji (bo relacje są zbiorami), jak sortowanie, grupowanie i agregacja.

Przypomnijmy jedno z pierwszych zapytań z punktu 3.

```
SELECT o.Imię, o.Nazwisko, h.Nazwa
FROM Osoby o, Hobby h
WHERE o.Id = h.IdOsoby
    AND 2012 BETWEEN h.Początek AND h.Koniec;
```

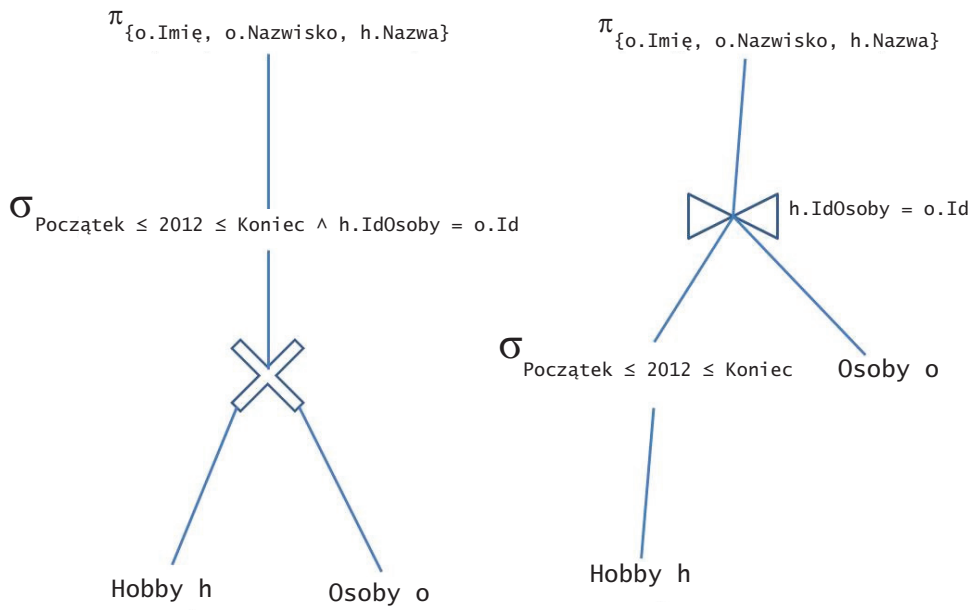
Początkowy plan logiczny realizacji tego zapytania, wynikający li tylko z jego analizy składniowej, znajduje się w lewej części rysunku 2. Zauważmy, że jest on nieefektywny. Przewiduje bowiem produkt kartezjański tabel wejściowych wygenerowany na podstawie klauzuli `FROM`, następnie ma zostać wykonana selekcja (wynik `WHERE`), a na końcu rzutowanie (wynik `SELECT`).

5.3. Optymalizatory

Teraz do dzieła przystępuje **optymalizator regułowy**, który stara się tak przekształcić plan logiczny zapytania, by poprawić jego wydajność. Stosowane reguły muszą być oczywiście poprawne (zachowywać ten sam wynik zapytania). Muszą także zwiększać wydajność w każdych warunkach. Niestety takich reguł jest niewiele. Jedną z nich jest reguła, której efekt jest widoczny w prawej części rysunku 2. Reguła ta polega na rozpoznaniu warunków selekcji jako warunków złączenia i zamianie iloczynów kartezjańskich na złączenia. Po jej zastosowaniu w przypadku powyższego zapytania pozbywamy się iloczynu kartezjańskiego i otrzymujemy szansę na wygenerowanie planu o złożoności mniejszej niż kwadratowa.

Optymalizator regułowy ma niestety niewielkie pole do popisu, choć wyniki jego działania są bardzo ważne. Po nim pałeczkę przejmuje **optymalizator kosztowy**, który znając statystyki dotyczące wielkości tabel, ich przestrzeni składowania, rozkładów wartości w kolumnach itd. dokonuje wyboru najlepszego planu zapytania. Zanim zajmiemy się nim, przyjrzyjmy się jego arsenałowi. Metod wykonywania zapytań jest oczywiście bardzo wiele i nie ma miejsca, żeby je wszystkie omówić. Ograniczymy się do niezwykle ważnych złączeń równościowych. Dlaczego tak ważnych? W relacyjnej bazie danych zależności między rekordami zapisujemy jako wartości kluczy obcych. Odnalezienie połączonego rekordu wymaga więc złączenia równościowego, które jest najlepiej rozpracowanym przez naukowców operatorem algebry relacji i też jako pierwszy został porządnie zaimplementowany przez firmę IBM w pierwszym systemie SZBD o nazwie R.

Rozważania będą ilustrowane zapytaniem, którego plan logiczny jest umieszczony po prawej stronie na rysunku 2. Przedstawimy podstawowe algorytmy wykonania złączenia równościowego z tego zapytania.



Rysunek 2. Początkowy i poprawiony plan logiczny przykładowego zapytania

5.4. Algorytmy realizacji złączenia równościowego

Pierwszy algorytm jest najprostszy; nosi nazwę **zagnieżdżonych pętli** (ang. *nested loops join*, NLJ). Polega on na wczytywaniu kolejnych fragmentów pierwszej tabeli do pamięci operacyjnej i przeglądaniu kolejnych stron drugiej tabeli w celu znalezienia pasujących par. Owszem, asymptotycznie ten algorytm ma złożoność kwadratową, jednak w przypadku baz danych bierzemy pod uwagę najkosztowniejsze operacje, czyli transfery między dyskiem i pamięcią, za to zaniedbujemy operacje w pamięci operacyjnej. Jeśli dostępna pamięć ma wielkość M, a liczby stron złączanych tabel to odpowiednio H i O, to algorytm zagnieżdżonych pętli wczyta OM fragmentów tabeli `Osoby` i dla każdego fragmentu odczyta całą zawartość `Hobby`. To daje nam HOM odczytów z dysku. Zauważmy po pierwsze, że jeśli jedna z tabel w całości mieści się w pamięci, to druga będzie odczytana tylko raz. Po drugie mamy tu do czynienia z odczytem dużych obszarów dysku, które są bardzo szybkie w porównaniu ze swobodnym czytaniem małych fragmentów. Te elementy mogą przesądzić o tym, że ten algorytm jest najlepszy dla konkretnej bazy danych.

Drugi algorytm, **złączenie indeksowe** (ang. *index nested loops*, INLJ) jest udoskonaleniem pierwszego, ale jest możliwy do wykonania tylko wtedy, gdy na kolumnie złączenia w jednej z tabel założono *indeks* umożliwiający wyszukiwanie wierszy o zadanym kluczu w czasie logarytmicznym. Indeks taki ma zwykle strukturę B^+drzewa, tj. zrównoważonego drzewa wyszukiwań, będą-

cego dalekim kuzynem drzew wyszukiwań binarnych (BST). Przypuśćmy, że mamy taki indeks na kolumnie `Id` tabeli `Osoby`[8]. Wtedy wystarczy przeglądać kolejno wiersze tabeli `Hobby` i dla każdego z nich wyszukać w indeksie na `Osoby.Id` wszystkie wiersze spełniające warunek złączenia. „Jaki dobry algorytm!", zachwyci się ktoś. Mamy przecież złożoność liniowo-logarytmiczną $O(H \log O)$. Uwaga! Wcale nie jest tak dobrze. Tym razem bowiem nasze dostępy do dysku są swobodne: za każdym razem pobieramy jeden blok z przypadkowego miejsca na dysku. Dostępy swobodne są jednak znacznie kosztowniejsze od ciągłych. To powoduje, że ten algorytm nie musi być lepszy od algorytmu NLJ.

Dwa pozostałe algorytmy: **złączenie przez scalanie** (ang. *sort merge join*, SMJ) i **złączenie haszowane** (ang. *hash join,* HJ) polegają na wstępnym przetworzeniu plików z tabelami. Algorytm SMJ najpierw sortuje obie tabele po wartości kolumny złączenia, a następnie przebiega obie posortowane kolumny poszukując pasujących par zupełnie tak samo, jak w fazie scalania algorytmu mergesort. Wskaźnik bieżącego rekordu porusza się tylko w jednym kierunku, a więc faza scalania ma koszt liniowy. Koszt złączenia przez scalanie to zatem $O(O \log O + H \log H)$.

Z kolei algorytm HJ czyta obie tabele i za pomocą funkcji haszującej liczonej na wartości kolumny złączenia rozrzuca ich wiersze do kubełków tablicy haszującej. Następnie w drugiej fazie łączy ze sobą zawartości kubełków. Jeśli funkcja haszująca jest dobra i nie tworzy zbyt wielkich kubełków, tj. niemieszczących się w pamięci, to druga faza odbywa się przy tylko jednym odczycie zawartości kubełków. Mamy więc złożoność $O(O + H)$. Ten algorytm ma jeszcze jedną zaletę. Jeśli jedna z tabel jest mała i jej tablica haszująca mieści się w pamięci, to haszujemy ją w pamięci, a następnie czytamy tabelę większą poszukując pasujących par za pomocą znajdującej się w RAM tablicy haszującej.

Dodatkową zaletą obu ostatnich algorytmów jest wykonywanie przez nie wyłącznie ciągłych, a więc bardzo szybkich, odczytów z dysku.

5.5. Optymalizator kosztowy

Ta mała próbka dostępnych algorytmów wykonywania złączenia równościowego pokazuje możliwości, jakimi dysponuje optymalizator kosztowy. Na marginesie zauważmy, jak ważna jest w tym procesie algebra relacji. Umożliwia bowiem podział zapytania na mniejsze fragmenty i korzystanie z pewnej rozsądnej liczby algorytmów dla tych fragmentów. To jednak nie wszystko, co można zrobić przy optymalizacji kosztowej. Optymalizator może wpływać także na kształt drzewa planu logicznego. Złączenie ma dwie ważne właściwości: symetryczność i łączność (jak dodawanie w arytmetyce):

[8] W istocie taki indeks zawsze będzie istniał. System SZBD założy go automatycznie, aby usprawnić kontrolę więzów klucza głównego na tej kolumnie.

$$R \bowtie S = S \bowtie R$$
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Kolejność wykonywania złączeń jest więc zupełnie obojętna. To dobra wiadomość, daje bowiem optymalizatorowi dużą przestrzeń możliwych planów wykonawczych. Jest też niestety poniekąd zła: przestrzeń ta jest tak ogromna, że nie da się jej gruntownie przeszukać. Logiczny plan zapytania ma postać drzewa. Jeśli uwzględnimy w nim tylko złączenia, to będzie to pełne drzewo binarne[9]. Przykłady takich drzew są pokazane na rysunku 3. Jeśli w zapytaniu złączamy *n* tabel, to samych kształtów drzew binarnych o *n* liściach jest bardzo dużo. Ich liczba jest równa liczbie Catalana:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

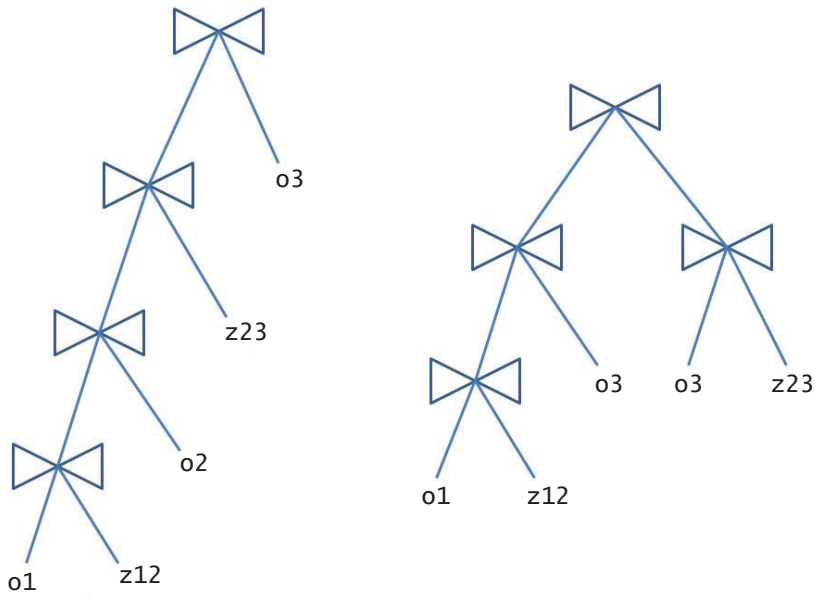
Nas interesuje jednak nie tylko kształt drzewa, ale i rozmieszczenie relacji w jego liściach. Trzeba zatem pomnożyć *Cn* przez *n*!, by poznać liczbę wszystkich możliwych drzew kolejności złączeń. To ogromna liczba. Można wykazać, że problem optymalizacji zapytań rozumiany jako wybór najbardziej efektywnego drzewa kolejności złączeń jest NP-trudny[10]. Optymalizator kosztowy musi sobie więc radzić inaczej. Należy pamiętać także o tym, że wyborowi podlega nie tylko kolejność wykonywania złączeń, ale także konkretne algorytmy ich wykonania, takie jak wspomniane już NLJ, INLJ, SNJ i HJ.

Pierwszy optymalizator kosztowy zrealizowany we wspomnianym już systemie R firmy IBM brał pod uwagę tylko jeden kształt drzew: drzewa skierowane w lewo, takie jak drzewo z lewej części rysunku 3. Ograniczyło to wielkość przestrzeni poszukiwań do zaledwie, bagatela, *n*!. Optymalizator systemu R był zaprogramowany dynamicznie na maskach bitowych. Dla każdego podzbioru już złączanych tabel i dla każdej z pozostałych tabel sprawdzał, czy dołączenie jednej z pozostałych tabel do tego zestawu da lepszy wynik niż do tej pory znaleziony. Ten algorytm ma złożoność wykładniczą $O(n2^n)$. Jest więc sensowny dla małych liczb złączanych tabel. W systemie PostgreSQL[11] jest domyślnie stosowany dla $n \leq 12$.

[9] Każdy wierzchołek **pełnego drzewa binarnego** jest liściem albo ma dokładnie dwóch synów.

[10] O problemach NP-trudnych jest mowa w artykule *Czy wszystko można obliczyć. Łagodne wprowadzenie do złożoności obliczeniowej.*

[11] System PostreSQL ma otwarty kod źródłowy, można więc podpatrzeć, co robi jego optymalizator zapytań. W przypadku produktów komercyjnych algorytmy optymalizacji są pilnie strzeżonymi tajemnicami.



Rysunek 3. Przykładowe kształty drzew planu zapytania

Gdy liczba złączanych tabel jest istotnie większa, algorytm przeszukujący całą przestrzeń planów jest zbyt powolny. Nie możemy przecież dopuścić, by optymalizacja zapytania trwała dłużej niż wykonanie pierwszego z brzegu planu, choćby najgorszego. Z tego powodu, dla większych zapytań stosuje się heurystyczne metody optymalizacji opracowane w dziedzinie sztucznej inteligencji, takie jak programowanie genetyczne czy symulowane wyżarzanie. Ze względu na to, że i tak nie przeszukujemy całej przestrzeni, a drzewa inne niż skierowane w lewo mogą być znacznie lepsze, heurystyczne optymalizatory kosztowe biorą pod uwagę także tzw. **drzewa krzaczaste** (ang. *bushy trees*). Przykład takiego drzewa widać po prawej stronie na rysunku 3. Drzewa z tego rysunku są planami wykonawczymi uproszczonego zapytania pokazanego już w punkcie 3:

```
SELECT DISTINCT o1.Id, o1.Imię, o1.Nazwisko,
                        o3.Id, o3.Imię, o3.Nazwisko
FROM Osoby o1, Osoby o2, Osoby o3, Znajomi z12, Znajomi z23
WHERE o1.Id = z12.IdOsoby
    AND o2.Id = z12.IdZnajomego
    AND o2.Id = z23.IdOsoby
    AND o3.Id = z23.IdZnajomego
    AND o1.Id!= o3.Id;
```

To jednak nie koniec możliwości optymalizacji w bazach danych. *Homo informaticus,* podobnie jak inni *Homo sapiens*, w nocy śpi. Baza danych nie wymaga jednak snu, więc gdy jej użytkownicy śpią, może wykonywać czynności na ich rzecz. Jedną z takich czynności jest optymalizacja *off-line*. W trakcie dnia SZBD może kolekcjonować ważne zapytania, tzn. często wykonywane i/lub pożerające dużo zasobów. Gdy były one wykonywane, były optymalizowane *on-line*. Siłą rzeczy musiała to być optymalizacja szybka, bo przy terminalu na wynik czekał niecierpliwy użytkownik. Optymalizator kosztowy nie mógł więc zbyt gruntownie przejrzeć przestrzeni poszukiwań najlepszego planu.

Gdy jednak zapada noc[12] i po serwerowni zaczynają snuć się wampiry, złe duchy i białe damy, SZBD przystępuje do wykonywania tzw. **zadań wsadowych** (np. generowanie faktur i tworzenie rozbudowanych raportów) oraz zadań administracyjnych, z optymalizacją *off-line* włącznie. Taka optymalizacja jest dokładniejsza, przeszukuje bowiem większy obszar przestrzeni planów wykonania. Rano, gdy użytkownicy przyjdą do pracy, mogą ze zdziwieniem zauważyć, że ich zapytania działają teraz szybciej. Oczywiście, aby ten mechanizm mógł zadziałać, zapytania muszą być rozpoznane przez SZBD jako powtarzalne. To jest możliwe tylko wtedy, gdy programiści będą stosowali zmienne wiązania, a nie będą umieszczali stałych w tekście zapytania – piszemy o tym na początku tego punktu.

Jak widać kwestia doboru planów jest bardzo dokładnie zbadana i dopracowana w systemach zarządzania bazami danych. Wiemy jednak, że wybrany plan wykonania nie musi być optymalny, bo do jego wyboru są stosowane algorytmy przybliżone. Może się więc zdarzyć, że wybrany plan działa bardzo nieefektywnie. Dotychczas omówiona architektura SZBD przewidywała działanie w stylu wybór planu – wykonanie. *Infocolligens* to jednak *Homo sapiens*, więc wziął też pod uwagę możliwość modyfikacji planu w trakcie wykonywania. To doprowadziło do koncepcji **adaptacyjnego przetwarzania zapytań** (ang. *adaptive query processing*)[13], czyli przetwarzania, które dostosowuje się do specyfiki otrzymanych danych.

Istnieją dwa podejścia do adaptacji. Pierwsze jest bardziej oczywiste i polega wykrywaniu wadliwości wykonywanego planu i jego korekcie po wykryciu usterek. Ma ono jednak istotną wadę – zwykle wyniki otrzymane przed korektą są tracone. Znacznie bardziej obiecujące jest drugie podejście, które można nazwać **bezplaniem probabilistycznym**[14]. Bezplanie oznacza brak planu wyko-

[12] Oczywiście mamy na myśli noc wirtualną, tzn. czas mniejszego obciążenia serwerów w związku z zakończeniem dnia roboczego w gospodarce realnej.

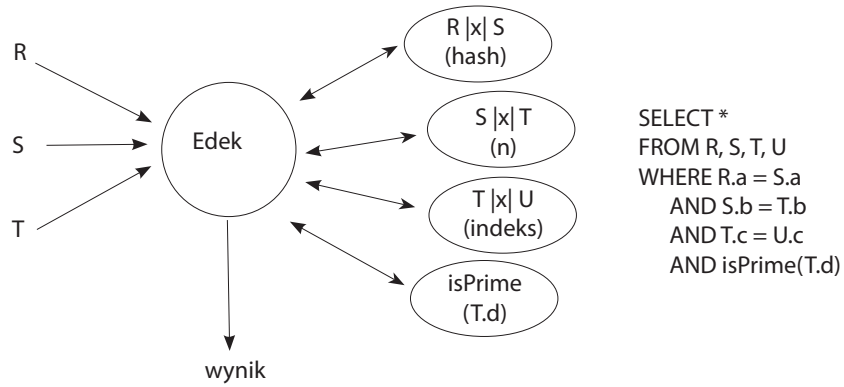
[13] Znakomitym źródłem informacji na temat adaptacyjnego przetwarzania zapytań jest przeglądowy artykuł [1].

[14] Ten termin jest całkowicie inwencją autora, który z góry przeprasza co wrażliwszych czytelników za ewentualny ból zębów w trakcie jego odczytywania. Nie ma on odpowiednika po angielsku, za taki można ewentualnie przyjąć *probabilistic no-plan*.

nania. Probabilistyczne jest dlatego, że kolejne kroki do wykonania są wybierane w wyniku losowania. Prawdopodobieństwa w tym losowaniu dobiera się na podstawie dotychczasowej historii realizacji planu wykonania. Przykładem mechanizmu adaptacyjnego jest metoda oparta na module Edek, zilustrowana na rysunku 4. Na wejściu ten moduł otrzymuje strumień danych wejściowych z relacji *R*, *S* i *T*. Gdy na wejściu pojawiają się krotki tych relacji, to są kierowane do odpowiednich operatorów złączeń i kosztownej selekcji (sprawdzenie, czy kolumna *d* w tabeli *T* jest liczbą pierwszą). Kierowanie odbywa się losowo na podstawie **tablicy trasującej** (ang. *routing table*). Każdy z możliwych celów jest opatrzony prawdopodobieństwem, z jakim krotka ma tam trafić. Gdy krotka wpada do węzła operatora, jest przetwarzana i może zostać odrzucona (jeśli nie spełniła warunku selekcji albo nie złączyła się z żadną z krotek relacji przeciwnej) lub zostać zwrócona do modułu Edek, być może w kilku kopiach (gdy złączona została więcej niż jedną krotką). Gdy krotka odwiedzi wszystkie operatory (kontroluje to towarzysząca jej maska bitowa), zostaje przekazana do strumienia wynikowego.

Gdzie tutaj jest adaptacja? Moduł Edek kontroluje parametry wykonania poszczególnych operatorów, takie jak czas przetworzenia jednej krotki oraz stopień wyjścia, czyli ile krotek wróciło do modułu Edek po wrzuceniu jednej krotki. Na podstawie tych parametrów moduł Edek może zmieniać prawdopodobieństwa trasowania w tablicy. Czasem jakaś ścieżka może mieć prawdopodobieństwo zerowe, jeśli jest całkowicie nieopłacalna. Plan wykonania zapytania realizowany przez moduł Edek jest więc zapisany w prawdopodobieństwach trasowania. Modyfikowanie planu (adaptacja) odbywa się poprzez zmianę tych prawdopodobieństw w trakcie działania. Nie powoduje to utraty wyników dotychczasowych obliczeń.

Adaptacyjne wykonywanie zapytań jest bardziej kosztowne niż klasyczne, jeśli statycznie wybrany plan zapytania jest dobry. Trzeba bowiem wówczas monitorować parametry wykonania i dostosowywać plan. Adaptacyjność jest obiecującą techniką w sytuacjach, w których optymalizator nie radził sobie z wyborem planu dla ogromnego zapytania, miał niedokładne statystyki o danych lub rzeczywiste dane zawierają niespodziewane korelacje, np. oszacowanie wielkości złączenia dwóch tabel było małe, ale w trakcie realizacji zapytania okazało się znacznie większe. Adaptacyjność to też sposób na przetwarzanie zapytań w **strumieniowych bazach danych** (ang. *stream databases*). W takich bazach dane są nieskończonym strumieniem, który nie jest w całości przechowywany. Przetwarzanie zapytań polega na przepuszczeniu tego strumienia przez operatory. W długich okresach charakterystyka strumieni danych zmienia się i mogą to być zmiany bardzo duże i jedynie adaptacyjny procesor zapytań ma szansę dobrze poradzić sobie w takich warunkach.



Tabele wejściowe	Odwiedzone	Następne cele
{R}	{}	(R |x| S, 1.0)
{ST}	{S |x| T, isPrime(T.d)}	(R |x| S, 0.3), (T |x| U, 0.7)
...	...	...

Rysunek 4. Edek. Jedna z metod adaptacyjnego przetwarzania zapytań

6. NOSQL

Bazy danych NOSQL są to narzędzia uzupełniające pewną lukę rynkową, wynikającą z ograniczeń klasycznych relacyjnych baz danych. Dzielimy je na trzy grupy: bazy klucz-wartość, bazy dokumentowe i bazy grafowe.

Bazy danych klucz-wartość są po prostu wielkimi skalowalnymi słownikami. Rejestrują bowiem wartości rekordów pod konkretnymi kluczami wyszukiwania. Pierwsza taka baza BerkeleyDB powstała zanim jeszcze pojawił się termin NOSQL. W bazie tej kluczem wyszukiwania jest ciąg bitów, a wartością... też ciąg bitów. Taka architektura bazy danych umożliwia nieograniczoną skalowalność, ponieważ dane można łatwo rozproszyć na bardzo wielu maszynach na podstawie wartości funkcji haszującej z klucza. Replikacja jest równie łatwa, o ile zrezygnujemy z aksjomatów ACID i zastąpimy je tzw. **spójnością ostateczną** (ang. *eventual consistency*). Gdy system zarządzania bazą danych implementuje taką spójność, to jest pewność, że po skończonym czasie od aktualizacji danych w jednym miejscu ostatecznie wszystkie kopie zostaną też poprawione. Zgodnie z konsekwencjami omówionego już twierdzenia CAP (podrozdział 2), tylko taka uboższa forma spójności jest możliwa. Bazy klucz-wartość eliminują więc ograniczenia klasycznych baz relacyjnych wynikające z rygorystycznego trzymania się aksjomatów ACID.

Obecnie jest bardzo wiele baz tego typu. Warto wspomnieć o bazie BigTable firmy Google, Apache Hbase czy Cassandrze, początkowo stosowanej w portalu Facebook. Przy pracach nad BigTable opracowano też nowy paradygmat przetwarzania rozproszonego, tzw. **MapReduce**. Przetwarzanie tą metodą jest bardzo popularne, ponieważ bardzo dobrze się skaluje i łatwo rozprasza na wiele maszyn. Metodę przetwarzania nazwiemy **skalowalną**, jeśli wzrost wielkości jej danych wejściowych powoduje proporcjonalny wzrost zużycia zasobów (czas działania, pamięć i liczba serwerów). Wyobraźmy sobie, że budujemy aplikację WWW. Będzie ona skalowalna, jeśli n-krotny wzrost liczby aktywnych użytkowników będzie wymagał dołączenia dodatkowo co najwyżej $n - 1$ nowych serwerów.

Pierwszy krok w MapReduce o nazwie **Map** polega na pobraniu wybranych rekordów z baz(y) danych i przekształceniu ich na pary klucz-wartość. Następnie w być może wielokrotnie wykonywanym kroku **Reduce,** pary o tym samym kluczu są grupowane i wyliczane są nowe wartości na podstawie pogrupowanych rekordów. Gdy dla każdego klucza istnieje już tylko jedna para, jest ona zwracana jako wynik. Operacja wykonywana podczas fazy redukcji musi być symetryczna i łączna żeby wynik obliczeń nie zależał od kolejności wykonania kroków Reduce.

Prześledźmy działanie metody MapReduce na przykładzie bazy danych, w której kluczem jest adres URL, a wartością zapamiętany dokument WWW spod tego adresu. Chcemy obliczyć, ile jest dokumentów w takiej bazie danych, w których występują wyrazy ze zbioru S. W fazie Map, dla każdego dokumentu i słowa $v \in S$ jest generowana para $\langle v, 1 \rangle$, o ile v występuje w tym dokumencie. W fazie Reduce pary z tym samym słowem są sukcesywnie grupowane przez serwery wykonawcze, przy tym liczby wystąpień są sumowane. Po zakończeniu tego procesu ostatni serwer generuje jedną parę dla każdego słowa ze zbioru S. Drugi element tej pary to liczba dokumentów, w których to słowo występuje. Wykonywane w fazie Reduce dodawanie liczb jest symetryczne i łączne, a więc niezależnie od postaci kaskady serwerów redukujących, otrzymany wynik będzie zawsze poprawny. Zasady działania metody MapReduce są rzeczywiście proste, ale to tylko pozory. Efektywna implementacja tego mechanizmu wymaga wiele wysiłku ze względu na złożone problemy komunikacji i synchronizacji między uczestniczącymi maszynami (serwerami).

Bazy dokumentowe (ang. *document-oriented databases*) służą do przechowywania dokumentów w rozmaitych formatach: XML, JSON, BSON i tekst. Równie dobrze implementuje się na nich MapReduce. Bardzo wygodne jest też ich skalowanie i replikacja. Często też trudno jest odróżnić bazy dokumentowe i klucz-wartość. Przykładami najbardziej znanych baz dokumentowych są CouchDB, MongoDB, eXist i Redis. Baza MongoDB jest używana przez liczne

firmy na rynku mediów, takie jak Disney (do przechowywania stanów gier *on-line*), CNN, New York Times. Także znane programistom repozytorium kodów źródłowych GitHub używa MongoDB.

Jeśli szukać szczególnej cechy wyróżniającej takie bazy wśród innych baz typu NOSQL, to będzie to istnienie mechanizmów **wyszukiwania pełnotekstowego**, który polega na znajdowaniu dokumentów zawierających zadane słowa lub frazy. Taka funkcjonalność jest realizowana za pomocą **indeksów/list odwróconych** (ang. *inverted lists/indices*), które dla każdego słowa zawierają listę ich wystąpień w dokumentach. Podobne udogodnienie starają się też dostarczyć producenci klasycznych relacyjnych SZBD, jednak testy wykazują, że ich implementacje są istotnie wolniejsze od tych stosowanych w bazach typu NOSQL. Przyczyną jest m.in. wierność aksjomatom ACID.

Ostatnią grupą baz NOSQL, o których tutaj wspominamy, są **bazy grafowe** (ang. *graph databases*). Przykładowymi produktami z tej rodziny są Neo4j, GraphDB, OrientDB. Zapewne najbardziej znaną w Polsce firmą używającą jednej z tych baz (Neo4j) jest Deutsche Telecom, właściciel sieci T-Mobile. Także firma Cisco wykorzystuje bazę Neo4j.

Grafowe bazy danych przechowują i udostępniają strukturę grafową utworzoną przez rekordy bazy, traktowane jako wierzchołki grafu. Z kolei dla każdego rekordu (wierzchołka) jest przechowywana w takiej bazie lista jego sąsiadów. Zwykle rekordy mogą zawierać dowolną liczbę pól i w istocie są dokumentami. To powoduje, że rozróżnienie, czy dana baza NOSQL jest bazą grafową, klucz-wartość czy dokumentową nie jest takie proste. Ale bazy grafowe, tak jak inne bazy NOSQL, mają na celu zaoferowanie czegoś niedostępnego w relacyjnych SZBD. Tym czymś jest możliwość przejścia do sąsiada w czasie stałym. Przypomnijmy, że podobna operacja w bazie relacyjnej wymaga złączenia lub użycia indeksu i ma koszt logarytmiczny. Bazy grafowe są więc rozwiązaniem przystosowanym do przechowywania dużych grafów i wykonywania na nich obliczeń wymagających swobodnego poruszania się między wierzchołkami.

Zaraz, zaraz, stop! *Infocolligensie*, zatoczyłeś koło i wróciłeś do źródeł! Wymyśliłeś na nowo sieciowy model danych! I tak, i nie. Z pewnością model wydaje się podobny, ale są pewne istotne różnice. Po pierwsze wierzchołki w grafowej bazie danych mogą przechowywać dowolne informacje podczas, gdy baza sieciowa miała rygorystycznie przestrzegany schemat. Po drugie baza grafowa ma zaimplementowane mechanizmy replikacji i fragmentacji; dobrze więc skaluje się na wiele serwerów. Bazy sieciowe były tylko scentralizowane. I po trzecie – interfejs, głupcze! Bazy hierarchiczne oferują indeksy odwrócone i języki zapytań grafowych (np. Gremlin). Tego nie było w bazach sieciowych. Możliwości oferowane przez grafowe bazy danych są bardzo obiecujące z punktu widzenia budowniczego portalu społecznościowego. Sieciowa baza danych w ogóle się do tego nie nadaje.

Tak to już jest, że *Homo informaticus* musi pewne idee wykryć ponownie, by je udoskonalić i wprowadzić na nowy, wyższy poziom. Zaczęliśmy od baz hierarchicznych i sieciowych, potem pojawiły się bazy relacyjne, obiektowe, obiektowo-relacyjne, klucz-wartość, dokumentowe i grafowe. W bazie danych każdego z tych rodzajów *Homo informaticus colligens* przechowuje po prostu rekordy. Na szczęście za każdym razem obok marketingu pojawiały się też nowe właściwości i funkcjonalności. Rozwój w dziedzinie baz danych, mimo chwilowego przestoju na przełomie stuleci, jest ponownie niezwykle dynamiczny. Oby tak dalej...

Literatura

1. Deshpande A., Ives Z.G., Raman V., *Adaptive Query Processing*, „Foundations and Trends in Databases" 1(1) 2007, 1-140, http://www.nowpublishers.com/product.aspx?product=DBS&doi=1900000001
2. Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych. Pełny wykład*, WNT Warszawa 2006
3. Stonebraker M., *Technical perspective – One size fits all: an idea whose time has come and gone*, „Comm. ACM", 51(12) 2008, 76
4. Subieta K., *Teoria i konstrukcja obiektowych języków zapytań*, Wydawnictwo PJWSTK, Warszawa 2004

Prof. nzw. dr hab. Krzysztof Stencel

jest profesorem nadzwyczajnym w Instytucie Informatyki Uniwersytetu Warszawskiego. Jego zainteresowania naukowe koncentrują się wokół baz danych oraz inżynierii oprogramowania.

Uczestniczył w zawodach programistycznych na początku ich rozwoju, reprezentując Polskę na I Międzynarodowej Olimpiadzie Informatycznej w 1989 roku. Od dłuższego czasu natomiast zajmuje się sędziowaniem i organizacją zawodów informatycznych. Przewodniczył jury Olimpiady Informatycznej od samego jej początku w roku 1993 aż do 2008 roku. Był też szefem jury zawodów międzynarodowych: Olimpiady Informatycznej Krajów Europy Środkowej (1997, 2004), Bałtyckiej Olimpiady Informatycznej, (2000, 2001, 2008) oraz Międzynarodowej Olimpiady Informatycznej (2005). W roku 2011 był przewodniczącym Olimpiady Informatycznej Krajów Europy Środkowej. W roku 2008 postanowił dla przyjemności rozwiązywać zadania programistyczne na słynnym serwerze Uniwersytetu Valladolid. Od 19 września 2011 roku do chwili pisania tego tekstu rozwiązał ponad 3000 zadań i jest liderem rankingu. Krzysztof Stencel jest autorem licznych publikacji oraz podręczników z baz danych, a także tłumaczem książek z tego zakresu.

stencel@mimuw.edu.pl

Wojciech Cellary

Katedra Technologii Informacyjnych
Uniwersytet Ekonomiczny w Poznaniu

Elektroniczny biznes – mariaż ekonomii i informatyki

Niniejszy artykuł poświęcony jest związkom informatyki z ekonomią. Odpowiada na pytanie, jak bardzo zmieniło się zarządzanie przedsiębiorstwami, ich kontakty z klientami i wzajemna współpraca przedsiębiorstw dzięki zastosowaniu metod i narzędzi informatyki i telekomunikacji. Omówiono tu specyficzne cechy elektronicznego biznesu wychodząc od charakterystyki elektronicznej informacji i komunikacji oraz pojęć produktu cyfrowego i usługi cyfrowej. Tekst przedstawia zalety elektronicznego biznesu typu przedsiębiorstwo – klient. Skupiono także uwagę na elektronicznym biznesie typu przedsiębiorstwo – przedsiębiorstwo, w szczególności na organizacjach wirtualnych, które są nową formą organizacyjną, powstałą dzięki informatyce i telekomunikacji. Następnie przedstawiono dwa nowe paradygmaty informatyczne, które mają bezpośredni wpływ na elektroniczny biznes: architekturę usługową SOA i przetwarzanie w chmurze. Wnioski z artykułu można podsumować następująco: mariaż ekonomii i informatyki prowadzi do elektronicznego biznesu i elektronicznej gospodarki, w której powstają nowe jakościowo miejsca pracy o ogromnych możliwościach.

1. Wstęp

Studiując historię informatyki, łatwo zauważyć, że rozwój komputerów postępował z jednej strony w kierunku ich miniaturyzacji, z drugiej – w kierunku zwiększania ich mocy obliczeniowej, a z trzeciej – w kierunku obniżki ich ceny. W efekcie dzisiaj, czyli około sześćdziesięciu lat od powstania informatyki, mamy do czynienia z małymi i tanimi komputerami o dużej mocy obliczeniowej, które znajdują powszechnie zastosowanie. Warto przy tym zauważyć, że te małe i tanie komputery nie wyparły tych większych, drogich, o bardzo dużych mocach obliczeniowych, tylko je uzupełniły, znakomicie poszerzając zakres zastosowań informatyki na praktycznie wszystkie dziedziny życia i działalności ludzkiej.

Przedsiębiorcy zainteresowali się informatyką natychmiast, jak tylko było ich stać na komputery. Na początku interesowali się nią wielcy przedsiębiorcy, a w miarę postępu – coraz mniejsi.

Informatyka zrewolucjonizowała sposób zarządzania przedsiębiorstwami, oferując możliwość przechowywania w bazach danych w uporządkowany sposób dowolnie szczegółowej informacji o przedsiębiorstwie i o każdym aspekcie jego działalności oraz możliwość automatycznego przetwarzania tej informacji dla celów zarządzania nim. Jednak znaczenie informatyki dla przedsiębiorstw i – szerzej – gospodarki nie ogranicza się do zmian ilościowych – zdolności do przetwarzania większej ilości informacji w krótszym czasie niż miało to miejsce przed wynalezieniem i zastosowaniem komputerów. Informatyka spowodowała fundamentalne zmiany w sposobach prowadzenia biznesu, kontaktów z klientami i współpracy między podmiotami gospodarczymi. Przykładem takiej zmiany w dziedzinie zarządzania przedsiębiorstwami i relacjami z klientami, 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. Taka bardzo daleko posunięta uniformizacja była wówczas głęboko uzasadniona. Miała ona na celu maksymalne obniżenie kosztów produkcji – łatwo zauważyć, że koszt przezbrojenia linii produkcyjnej z koloru czarnego na biały był ogromny, a zysk niewielki. Natomiast niskie koszty produkcji i w konsekwencji niska cena produktu były warunkiem uzyskania odpowiednio szerokiego rynku zbytu, bez którego produkcja nie miałaby ekonomicznego sensu. 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 speł-

nianie potrzeb zróżnicowanych klientów daje natomiast większe możliwości eksploatacji potencjalnych możliwości rynkowych przez przedsiębiorstwa, ze wszystkim pozytywnymi dla nich konsekwencjami takiego stanu rzeczy.

W tym artykule, który jest rozszerzoną wersją [9], wyjaśniamy istotę zastosowania informatyki w działalności gospodarczej, koncentrując się na kwestiach jakościowo nowych.

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 lub cielęcej skóry), a inaczej gdy był to 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 następnie – społecznymi. Internet i telefon komórkowy są wynalazkami, które upowszechniły się bardzo szybko. Obecnie około 2,3 miliarda ludzi korzysta z Internetu, a prawie 6 miliardów z telefonów komórkowych, które można uznać za inną formę dostępu do Internetu [2]. Umasowienie informacji i komunikacji elektronicznej spowodowało pojawienie się na rynku produktów i usług cyfrowych oferowanych i świadczonych przez Internet 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 za pośrednictwem sieci. 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**. Jest to taka gospodarka, w której produkty i usługi cyfrowe 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, organizuje dostawy i dokonuje płatności.

Elektronicznym biznesem nazywamy również 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. Komunikacyjne natomiast dotyczą wymiany informacji między komunikującymi się stronami. Usługi transakcyjne są takimi usługami komunikacyjnymi, które pociągają za sobą skutki prawne. Dla przykładu, jeśli ktoś kupił coś przez Internet, to musi zapłacić, a e-sklep, który coś 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 7 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 sprzyja 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 – organizacje wirtualne.

W następnych rozdziałach scharakteryzujemy te rodzaje elektronicznego biznesu, skupiając się najbardziej na organizacjach wirtualnych, gdyż stanowią one największą innowację w sferze zarządzania przy wykorzystaniu środków informatyki i telekomunikacji.

3. Charakterystyka elektronicznej informacji i komunikacji

Punktem wyjścia do zrozumienia przemian, jakie w organizacji przedsiębiorstw spowodowała informatyka i telekomunikacja, jest analiza cech **infor-**

macji elektronicznej, czyli zdematerializowanej, w porównaniu z informacją zapisaną na nośniku papierowym, czyli materialnym [7].

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 do informacji bez względu na jego własne położenie geograficzne oraz położenie geograficzne 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 wydatek na sam nośnik, 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 niska opłata za dostęp do informacji elektronicznej przez Internet w porównaniu choćby z kosztami delegacji do klasycznych archiwów informacji 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 z tylko jednym wybranym porządkiem – na przykład chronologicznym lub alfabetycznym. Inny porządek 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 znaczny sposób dostęp do informacji, co ma szczególne znaczenie w przypadku dużych archiwów.

Informacja elektroniczna jest wyszukiwana automatycznie. Możliwe jest efektywne przeszukiwanie archiwów informacji elektronicznej wspomagane kompu-

terowo, 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 mu zbytecznej informacji.

Ogólnie 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 **komunikacja elektroniczna** ma wiele cech, które wpływają na sposób prowadzenia działalności gospodarczej. Po pierwsze dzięki upowszechnieniu telekomunikacji mobilnej, dzisiejsze połączenia mają charakter: człowiek – człowiek, a nie aparat telefoniczny – aparat telefoniczny. Telekomunikacja stała się więc 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 zapewnia ona 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". 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 stosunkowo zaawansowanego świata biznesu, ale właśnie 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 ułatwione jest wyszukiwanie informacji o potencjalnych 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 zmienności jest wynikiem stałego udoskonalania i rozwoju oprogramowania systemów komputerowych stosowanych do produkcji i świadczenia usług. Przeobrażeniom ulegają metody pracy, w czym znowu duży udział ma konieczność nabycia przez pracowników umiejętności posługiwania się nowym oprogramowaniem stosowanym w pracy. Zmianie ulega organizacja pracy, w szczególności na stanowiskach, na których wynikiem pracy jest pewna informacja lub komunikacja. Przykładem jest telepraca, czyli świadczenie pracy na odległość przez sieć. Wreszcie, zmiany technologiczne i organizacyjne w gospodarce w naturalny sposób pociągają za sobą przekształcenia prawne, do których przedsiębiorstwa muszą się na bieżąco dostosowywać.

Zróżnicowanie działalności przedsiębiorstw objawia się w skali makro- i mikroekonomicznej. W skali makroekonomicznej mamy 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, mają do czynienia ze zróżnicowaniem prawnym – w różnych krajach panują różne 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 jego podstaw leży przekonanie, że kosztownym elementem każdego procesu biznesowego jest pozyskanie klienta przez przedsiębiorstwo. Dlatego jeśli uda się zdobyć 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. Elektroniczny biznes przedsiębiorstwo – klient

Niewątpliwy sukces elektronicznego biznesu typu przedsiębiorstwo – klient i jego szybka i coraz powszechniejsza akceptacja wynika z tego, że obie strony na nim zyskują. Przedsiębiorstwo prowadząc działalność przez sieć przede wszystkim obniża swoje koszty przerzucając pewne czynności z pracownika na klienta. A klient nie tylko nie protestuje przeciwko temu, ale możliwość samodzielnego wykonywania tych czynności uważa za wielką zaletę. Nikt, kto zaczął korzystać z e-banku, nie wróci do tradycyjnego oddziału banku zlokalizowanego na jakiejś ulicy, do którego trzeba dojechać, zaparkować, a potem stać w kolejce, ponieważ jest to po prostu zbyt uciążliwe. Tradycyjne oddziały banków nadal istnieją, ale zmieniają swoją funkcję z operacyjnej na doradczą. Dla przedsiębiorstw prowadzenie elektronicznego biznesu ma jednak wiele innych zalet niż tylko obniżka kosztów. Po pierwsze rozszerzają zakres swojej działalności, bo uwalniają się od ograniczeń geograficznych. Po drugie prowadzą biznes 24/7. Po trzecie mają znacznie większy kontakt z klientami, a dzięki temu większe możliwości promocyjne i marketingowe oraz możliwość sprzedaży produktów komplementarnych i usług po sprzedaży. Kontakty z klientami są prowadzone wieloma kanałami: przez witryny WWW, mejl, fora dyskusyjne, aplikacje mobilne, SMS, prezentacje wideo i media społecznościowe. Kontakty te – co najmniej niektóre – są dwukierunkowe. Zupełną nowością jest tworzenie się wirtualnych społeczności wokół przedsiębiorstw, ich produktów lub usług [8]. Takie społeczności prowadzą do aktywnego zaangażowania klientów, które może być wykorzystane do skłonienia ich do wielokrotnego powrotu do przedsiębiorstwa, promocji jego marki oraz jego produktów i usług, a także do ich doskonalenia i rozwoju. Dla przedsiębiorstw prowadzących elektroniczny biznes, ich klienci stali się stałym źródłem innowacji.

Szczególnie duże nowe możliwości mają podmioty świadczące usługi elektroniczne. Na standardowych platformach oferowanych przez wielkie, światowe koncerny powstają setki tysięcy aplikacji tworzonych przez niezależne przedsiębiorstwa, często – na początku działalności – nawet jednoosobowe. Najlepiej widać to zjawisko na przykładzie aplikacji mobilnych. Jest to nowy sektor gospodarczy, o bardzo małych barierach wejścia i nieograniczonych możliwościach sukcesu, zależnego przede wszystkim od wyobraźni twórcy i trafienia z nowym produktem lub usługą w konkretne potrzeby ludzi.

6. Od e-biznesu typu przedsiębiorstwo – przedsiębiorstwo do organizacji wirtualnych

Przeglądając Internet widać jak na dłoni miliony ofert elektronicznego biznesu typu przedsiębiorstwo – klient. Paradoksalnie, tego rodzaju e-biznes stanowi tylko 15% całości, natomiast 85% to e-biznes typu przedsiębiorstwo – przedsiębiorstwo. Jego istotą jest współpraca przedsiębiorstw przez Internet w celu wyprodukowania złożonego produktu lub zaoferowaniu na rynku złożonej usługi. Cechy elektronicznej informacji i komunikacji są wykorzystywane do przyspieszenia produkcji i obniżenia jej kosztów, poszerzenia rynków zbytu oraz do doskonalenia i tworzenia nowych produktów i usług. Elektroniczny biznes typu przedsiębiorstwo – przedsiębiorstwo różni się stopniem integracji przedsiębiorstw – od bardzo powierzchownej, na przykład ograniczającej się do wymiany mejli z załącznikami – do najbardziej zaawansowanej, jaką są **organizacje wirtualne** [4, 5, 6]. Organizacje wirtualne są odpowiedzią na gospodarcze wymaganie dynamizmu i zróżnicowanie globalnego rynku. Są one zbiorami współpracujących ze sobą przez Internet jednostek gospodarczych i innych – na przykład administracyjnych lub pozarządowych – występujących na rynku tak, jakby były jednym przedsiębiorstwem. Organizacje wirtualne mogą prowadzić działalność polegającą zarówno na oferowaniu produktów i usług, jak i zamawianiu produktów i usług na swoje potrzeby.

Procesy tworzenia się organizacji wirtualnych mogą przebiegać zarówno od góry do dołu, jak i od dołu do góry. W tym pierwszym przypadku mamy do czynienia z dekompozycją tradycyjnych, dużych organizacji hierarchicznych. Natomiast w drugim przypadku – z integracją małych i średnich przedsiębiorstw.

W epoce, w której dominował papierowy obieg informacji, w szczególności obieg papierowych dokumentów (ta etap 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, 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 prędki obieg informacji nie był im potrzebny, gdyż 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 w organizacje wirtualne małych i średnich przedsiębiorstw. Głównym powodem tej integracji jest przekonanie, że na dynamicznym i zróżnicowanym rynku małe lub średnie przedsiębiorstwo samo nie jest w stanie sprostać wyzwaniom i podołać konkurencji. Dlatego jest koniecznie włączenie się takiego podmiotu w większy organizm gospodarczy. Integracja z organizacją wirtualną jest bardzo dobrym rozwiązaniem, gdyż pozwala małemu lub średniemu przedsiębiorstwu zachować swą podmiotowość, na której często właścicielom firmy bardzo zależy. W procesie integracji, przedsiębiorstwa, o których mowa, mogą albo dołączać do organizacji wirtualnych wynikających z dekompozycji dużych organizacji hierarchicznych, które się na nie otwierają, albo próbować samoorganizować się w organizacje wirtualne w celu oferowania bardziej złożonych produktów i usług oraz poprawy swojej pozycji konkurencyjnej na większym rynku.

7. Cechy organizacji wirtualnych

Organizacje wirtualne charakteryzują się trzema zasadniczymi cechami. Po pierwsze kulturą biznesową i sposobem funkcjonowania ukierunkowanym

na podążanie za nieustannie zmiennymi potrzebami klientów. Po drugie skoncentrowaniem się każdej jednostki wchodzącej w skład organizacji wirtualnej, na doskonaleniu swoich kluczowych kompetencji i opieraniu się na zaufaniu do partnerów w odniesieniu do pozostałych funkcji. Wreszcie po trzecie – standaryzacją danych, systemów informatycznych i procesów biznesowych w skali całej 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, lecz doprowadzane do wdrożeń przemysłowych.

Druga cecha przedsiębiorstw wchodzących w skład wirtualnych organizacji, 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 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 organizacji wirtualnych jest konieczność standaryzacji danych, systemów informatycznych i procesów biznesowych w skali całej 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.

8. Transformacja przedsiębiorstw do organizacji wirtualnych

Efekt koncentracji przedsiębiorstwa na swoich kluczowych kompetencjach uzyskuje się na drodze transformacji funkcjonalnej i operacyjnej [3]. Transformacja funkcjonalna polega na wydzielaniu z przedsiębiorstwa funkcji oraz zadań. W przypadku rozdzielenia 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 zlecenie funkcji jest praktyką biznesową stosowaną od dawna w celu obniżenia kosztów funkcjo-

nowania przedsiębiorstw, to wydzielanie zadań jest charakterystyczne dla 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 źle prowadzone, to bezpośrednio wpływają na biznes firmy, oraz na takie, które nawet jeśli są nieodpowiednie to bezpośrednio na biznes firmy nie mają wpływu. Te pierwsze nazywamy **działaniami krytycznymi** dla misji firmy, a te drugie – **niekrytycznymi** dla misji firmy. Drugim kryterium jest odróżnienie od konkurencji, dzielące działania na bezpośrednio dotyczące przewagi konkurencyjnej firmy i na przewagę konkurencyjną bezpośrednio niewpływające. Te pierwsze to **działania kluczowe**, 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, trzeba rozdysponować 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. Gdyby operator telekomunikacyjne nie łączył (dobrze) rozmów, to klienci odejdą od niego do konkurencji, a finanse firmy załamią się. Wystawianie rachunków klientom jest działaniem kluczowym, bo bez nich firma nie uzyska przychodów, ale niekrytycznym dla misji firmy – klienci operatora telekomunikacyjnego nie odejdą od niego tylko dlatego, że rachunki otrzymują 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.

9. Architektura usługowa SOA

Największym wyzwaniem stojącym przed organizacjami wirtualnymi jest odpowiedź na pytanie, jak wydzielić zadanie, ale zachować kontrolę nad spo-

sobem 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 organizacji wirtualnej. Warunkiem takiej integracji jest odpowiednia architektura rozproszonych systemów informatycznych – w tym przypadku **architektura usługowa SOA** (ang. *Service Oriented Architecture*) [1, 10, 11]. Architektura usługowa SOA jest formą organizacyjną, 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 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 zarządzanie wyłącznie całymi łańcuchami dostaw zapewnia spełnienie całościowych potrzeb klientów.

10. Przetwarzanie w chmurze

Drugą obok architektury usługowej SOA nową formą organizacyjną współczesnej informatyki jest **przetwarzanie w chmurze** (ang. *cloud computing*) [12, 13]. Wyrażenie to wzięło się stąd, że Internet na rysunkach jest przed-

stawiany w postaci chmury. Przetwarzanie w chmurze polega na zastąpieniu lokalnej informatyki w przedsiębiorstwach i urzędach usługami świadczonymi na masową skalę przez Internet przez wyspecjalizowane przedsiębiorstwa informatyczne. Dwie zasadnicze cechy odróżniają przetwarzanie w chmurze od tradycyjnych podejść do przetwarzania danych, z których jedna ma charakter informatyczny, a druga biznesowy. W chmurze – jeden system operacyjny, jedna baza danych i jedna aplikacja obsługują wiele przedsiębiorstw i/lub urzędów, czasami liczonych w tysiącach. Różnice między poszczególnymi klientami są modelowane na poziomie metadanych (danych opisujących dane), a nie w postaci wersji aplikacji. Zatem jeśli trzeba uaktualnić aplikację, to robi się to jeden raz, a nie tyle razy, ile jest przedsiębiorstw lub urzędów korzystających z tej lub podobnych aplikacji. Natomiast każdy klient uaktualniając swoje metadane, może dopasować tę aplikację do indywidualnej specyfiki. Model biznesowy przetwarzania w chmurze polega na **zapłacie za usługę** według wskazań licznika, tak jak płacimy za wodę, gaz, energię elektryczną czy taksówkę. Tak jak – na szczęście – nie musimy budować własnej elektrowni, aby mieć energię elektryczną w domu, tak nie musimy inwestować w serwerownię, aby móc korzystać z usług informatycznych.

Zalety przetwarzania w chmurze są wielorakie i różnej natury. Najważniejszą zaletą jest wyrównywanie szans małych i dużych, biednych i bogatych. Dotychczas na naprawdę porządną informatykę było stać tylko dużych i bogatych – duże przedsiębiorstwa i duże gminy (miasta). Mniejsi i ubożsi stawali przed trudną do pokonania barierą finansową – nie mieli środków na zainwestowanie w profesjonalne centrum przechowywania i przetwarzania danych, na zakup licencji na oprogramowanie, na zatrudnienie zespołu zawodowych informatyków o wysokich kwalifikacjach, których notabene w mniejszych miejscowościach w ogóle nie można znaleźć. Dzięki przetwarzaniu w chmurze, wszystkie te bariery znikają, a zarówno mali, jak i duzi mają dostęp do tych samych usług informatycznych, w dodatku tańszych, co wynika z efektu skali. Ma to znaczący, pozytywny wpływ na rozwój gospodarczy i zapewnia jednolity poziom usług e-administracji, niezależnie od tego, czy ktoś mieszka w dużym mieście, czy w małej wsi.

Drugą ważną zaletą jest redukcja ryzyka biznesowego. W przypadku inwestycji we własną serwerownię ryzyko jest zawsze duże. Jeśli zainwestuje się za dużo w stosunku do liczby przyszłych klientów danego przedsiębiorstwa – która jest zawsze trudna do przewidzenia – to wzrosną koszty i można tych klientów utracić na rzecz tańszej konkurencji. Jeśli zainwestuje się za mało, a zdarzy się dobra koniunktura, to nie można jej wykorzystać, bo na zbyt małym sprzęcie i oprogramowaniu uszytym na zbyt małą miarę, jakość usług informatycznych gwałtownie spada, a klienci się zniechęcają. Przetwarzanie w chmurze zapewnia pełną **skalowalność**. Natomiast dzięki opłatom według użycia, jeśli koniunktura jest dobra, to przedsiębiorstwo potrzebuje więcej usług informatycznych,

ale ma z czego za nie zapłacić. Jeśli koniunktura jest zła, to nie płaci za usługi, których nie potrzebuje. Strona finansowa biznesu staje się przewidywalna w każdych warunkach, czyli ryzyko prowadzenia biznesu obniża się, co oczywiście sprzyja rozwojowi gospodarki.

Wreszcie trzecia bardzo ważna korzyść przetwarzania w chmurze dotyczy energii, a zatem w konsekwencji ekologii. Na świecie jest około 1,5 miliarda komputerów. Wszystkie, choćby chwilowo, są podłączone do serwerów. Te serwery są jednak wykorzystane średnio w siedmiu procentach. To oznacza gigantyczne marnotrawstwo energii używanej do zasilania tych prawie bezczynnych serwerów. Idea przetwarzania w chmurze jest zatem taka: ponieważ Ziemia się obraca, to gdy Azjaci idą spać, budzą się Europejczycy, a gdy Europejczycy idą spać, to jeszcze pracują Amerykanie. W nocy większość urzędów i przedsiębiorstw nie pracuje, a zatem obciążenie serwerów jest mniejsze. Niech więc europejskie dane przetwarzają się częściowo na azjatyckich serwerach w godzinach nocnych w Azji, a za dnia w Europie, a potem, jak Ziemia się obróci, amerykańskie dane na europejskich serwerach itd. W ten sposób można oszczędzić energię, ochronić środowisko i obniżyć opłaty za usługi informatyczne.

Przetwarzanie w chmurze nie ma, niestety, samych zalet. Może nie wadą, ale na pewno problemem jest zapewnienie bezpieczeństwa i prywatności danych w chmurze. Ponieważ dane te cyrkulują pomiędzy różnymi serwerami rozsianymi po całym świecie, to bardzo trudno jest wykryć, a jeszcze trudniej skazać winnego naruszenia prywatności. Przetwarzanie w chmurze naocznie ukazuje, jak bardzo przestarzała stała się koncepcja terytorialnego obowiązywania prawa. Polskie prawo działa od Odry do Bugu i od Bałtyku do Tatr, ponieważ tylko na tym terytorium można je egzekwować. Prawdziwe życie przenosi się jednak do Internetu, w którym takie granice nie istnieją. Globalny Internet wymaga globalnego prawa – w naszym, tak bardzo zróżnicowanym świecie, jest to jedno z najważniejszych wyzwań.

11. Wnioski

Jak wynika z tego artykułu, współczesna gospodarka opiera się na mariażu ekonomii i informatyki. Przyszłość należy do małych i średnich przedsiębiorstw, które są zwinne, innowacyjne i szybko adaptują się do zachodzących zmian, ale tylko takich, które są informatycznie zintegrowane w organizacje wirtualne i korzystają z przetwarzania w chmurze. W globalnej gospodarce małe i średnie przedsiębiorstwa same sobie nie poradzą. Kluczowo ważna jest dla nich współpraca na skalę międzynarodową. Formą takiej współpracy są organizacje wirtualne, a podstawową technologią – technologie elektronicznego biznesu oparte na środkach i metodach informatyki i telekomunikacji. Szansą polskiej gospo-

darki jest duża liczba młodych, dobrze wykształconych osób, które mogą tworzyć innowacyjne przedsiębiorstwa świadczące usługi oparte na wiedzy, stanowiące znaczące komponenty 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 wybierając studia, ponieważ elektroniczny biznes jest jedną z najbardziej obiecujących dziedzin zastosowań informatyki.

Literatura

1. Ambroszkiewicz S., Brzeziński J., Cellary W., Grzech A, Zieliński K. (eds.), *SOA Infrastructure Tools – Concepts and Methods*, Wydawnictwa Uniwersytetu Ekonomicznego w Poznaniu, Poznań 2010
2. (B.a.), *Key statistical highlights: ITU data release June 2012*, ITUWorld Telecommunication/ICT Indicators Database, International Telecommunication Union, http://www.itu.int/ITU-D/ict/statistics/material/pdf/2011%20Statistical%20highlights_June_2012.pdf
3. Brunett K., Fishman G. (eds.), *The NVO Way of Doing Business. The Bridge: Connecting Business and Technology Strategies*, Cisco Systems Internet Business Solution Group 2003
4. Camarinha-Matos L., Afsarmanesh M.H. (eds.), *Collaborative Networks: Reference Modeling*, Springer Verlag, Berlin, Heidelberg 2010
5. Camarinha-Matos L., Afsarmanesh M.H., Ollus M. (eds.), *Virtual Organizations: Systems and Practices*, Springer Verlag, Berlin, Heidelberg 2005
6. Cellary W., *Networked Virtual Organizations: A Chance for Small and Medium Sized Enterprises on Global Markets*, w: Godart C., Gronau N., Sharma S., Canals G. (eds.), *Software Services for e-Business and e-Society*, Springer Verlag, Berlin Heidelberg 2009, 73-81
7. Cellary W., *Globalization from the Information and Communication Perspective*, w: Janowski T., Mohanty H. (eds.), ICDCIT 2007, LNCS No. 4882, Springer Verlag, Berlin, Heidelberg 2007, 283-292
8. Cellary W., *Content Communities on the Internet*, „Computer" 41(2008), 106-108
9. Cellary W., *Czy komputery będą robić biznes*, w: M.M. Sysło (red.), *Podstawy algorytmiki. Zastosowania informatyki*, Vol. 1, Warszawska Wyższa Szkoła Informatyki, Warszawa 2011, 33-45
10. Estefan J.A., Laskey K., McCabe F., Thornton P. (eds.), *Reference Architecture Foundation for Service Oriented Architecture Version 1.0*, OASIS Committee Draft 02, OASIS SOA Reference Model Technical Committee, 2009, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf
11. MacKenzie M., Laskey K., McCabe F., Brown P.F., Metz R. (eds.), *Reference Model for Service Oriented Architecture 1.0*, OASIS Standard, OASIS SOA Reference Model Technical Committee, 2006, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
12. Sosinsky B., *Cloud Computing Bible*, Wiley Publishing, Indianapolis 2011
13. Rhoton R., *Cloud Computing Explained: Implementation Handbook for Enterprises*, Recursive Limited, 2011

Prof. dr hab. inż. Wojciech Cellary

jest kierownikiem Katedry Technologii Informacyjnych na Uniwersytecie Ekonomicznym w Poznaniu. Specjalizuje się w problematyce elektronicznego biznesu, elektronicznej gospodarki opartej na wiedzy, elektronicznej administracji i społeczeństwa informacyjnego. Był konsultantem wielu ministerstw, komisji sejmowych i senackich oraz ekspertem Komisji Europejskiej. Kierował licznymi projektami naukowymi i przemysłowymi zakończonymi wdrożeniami na skalę międzynarodową. Był redaktorem naukowym Raportu o Rozwoju Społecznym: Polska w drodze do globalnego społeczeństwa informacyjnego, opracowanego pod auspicjami Programu Narodów Zjednoczonych ds. Rozwoju (UNDP). Jest członkiem licznych towarzystw i organizacji naukowych. W latach 2010-2011 pełnił funkcję przewodniczącego Rady Informatyzacji przy Ministrze Spraw Wewnętrznych i Administracji, a obecnie jest członkiem Rady Informatyzacji przy Ministrze Administracji i Cyfryzacji oraz członkiem Rady ds. Informatyzacji Edukacji przy Ministrze Edukacji Narodowej. Autor 10 książek i ponad 170 artykułów naukowych.

cellary@kti.ue.poznan.pl
www.kti.ue.poznan.pl

Ryszard Tadeusiewicz

Laboratorium Biocybernetyki
Akademia Górniczo-Hutnicza, Kraków

Informatyka medyczna

Ważnym źródłem sukcesów medycyny jest inżynieria biomedyczna oraz jej dział – informatyka medyczna. Współczesny lekarz może osiągać znacznie lepsze wyniki leczenia, niż jego kolega kilkadziesiąt lat temu, między innymi za sprawą szerokiego wykorzystania w medycynie osiągnięć informatyki. I o tym właśnie mówi ten artykuł. Rozważane są role i zadania jakie nowoczesna informatyka może spełniać we wspomaganiu lekarzy i służby zdrowia. Zadania te odnoszą się do zarządzania placówkami medycznymi. Dlatego omawiana jest rejestracja oraz ewidencja pacjentów oraz usług medycznych. Zadania te wiążą się ze wspomaganiem diagnostyki medycznej ze szczególnym uwzględnieniem systemów komputerowego przetwarzania sygnałów biomedycznych (EKG, EEG itp.) oraz systemów obrazowania medycznego (m.in. tomografia komputerowa). Omawiane są także role komputerów w sterowaniu aparaturą terapeutyczną (roboty chirurgiczne, magnetoterapia itp.). Prezentowane są również możliwości, jakie wiążą się z wykorzystaniem informatyki do zdalnego niesienia pomocy medycznej bezpośrednio w domach pacjentów lub w odosobnionych miejscach ich pobytu (na przykład podczas wypraw wysokogórskich) za pomocą tak zwanej telemedycyny.

1. Wprowadzenie

Informatyka jest dziedziną wiedzy, która dostarcza metod pozwalających rozwiązywać różne problemy. Część z nich jest związana z rozwojem badań naukowych, inne dotyczą zastosowań komputerów w technice i w gospodarce, w fabrykach i w bankach, w szkołach i w urzędach, a także w domach i w czasie podróży. Obserwujemy to na co dzień i do tego zdążyliśmy się już przyzwyczaić. Istnieje jednak na ogół mało znany obszar zastosowań informatyki, który jest związany z potrzebami medycyny. Tu komputer staje się sojusznikiem lekarza w jego walce o zdrowie i życie ludzkie. Dzięki wyposażeniu w odpowiednią aparaturę komputerową współczesny lekarz może lepiej i szybciej rozpoznawać procesy chorobowe trapiące pacjenta, a także trafniej ustalać sposób leczenia wykrytej choroby. Lekarze sprzed dwudziestu czy trzydziestu lat, którzy tych możliwości nie posiadali, gdyż liczba komputerów wtedy nie była duża, a ponad to były trudno dostępne, byli w trudniejszej sytuacji i rzadziej odnosili sukcesy. Dlatego komputery, które obecnie wspomagają pracę lekarzy, pozwalają im działać o wiele skuteczniej i pewniej, eliminując cierpienia pacjentów i oddalając ryzyko ich śmierci. A każdy czytający te słowa musi sobie uświadomić, że nawet jeśli teraz jest młody i zdrowy, to nieuchronnie przyjdzie taka chwila, gdy lekarze wyposażeni w mądrze zaprogramowane komputery będą starali się zmniejszyć JEGO cierpienia i usunąć zagrożenie JEGO życia – lub będą walczyć o życie i zdrowie jego najbliższych. Szanse sukcesu w tej walce może znacząco zwiększyć informatyk dostarczający nowych i doskonalszych narzędzi komputerowych dla medycyny.

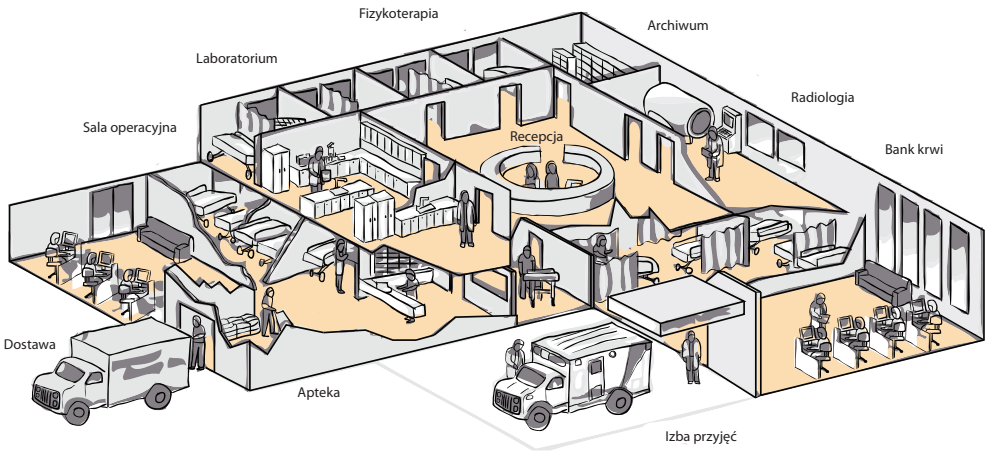
Czy może być szlachetniejszy cel studiowania i rozwijania informatyki?

2. Do czego można zastosować komputer w medycynie?

Komputery są dziś używane wszędzie i do wszystkiego, ponieważ dzięki różnorodnym programom – komputer jest dziś bardziej wielozadaniowy niż najbardziej rozbudowany szwajcarski scyzoryk. Nic więc dziwnego, że komputery pojawiają się także w jednostkach służby zdrowia. Jednak byłoby poważnym błędem oczekiwanie, że komputer w szpitalu będzie można wykorzystywać według tych samych wypróbowanych metod, jak tego typu sprzęt w przedsiębiorstwie handlowym, fabryce, banku lub urzędzie. Owszem, systemy komputerowe w medycynie mogą zbierać i przetwarzać dane o pacjentach, o chorobach i ich objawach, o zastosowanym leczeniu i o uzyskanych wynikach. W tym

zakresie ich zadania są podobne do tych, jakie pełnią komputery wykorzystywane w innych obszarach zastosowań informatyki. Komputerowi jest przecież obojętne, czy musi zapamiętać dane o pacjentach, czy o pasażerach samolotu. Usługi medyczne można więc rejestrować w komputerowych bazach danych podobnie jak usługi turystyczne czy hotelowe – i robi się to między innymi w celach rozliczeniowych.

Co więcej, zbieranie informacji o tym, jakim zabiegom poddawano pacjenta i ich skutkach, jest dodatkowo ważne z medycznego punktu widzenia – ponieważ można na przykład uniknąć błędu powtórnego podania leku, na który pacjent był uczulony albo który już wcześniej okazał się nieskuteczny. Dlatego w medycynie znajdują zastosowania typowe narzędzia informatyczne, takie jak bazy danych czy sieci komputerowe, ułatwiające zdalny dostęp do tych danych. Znajdą się one w każdej przychodni czy szpitalu (rys. 1).



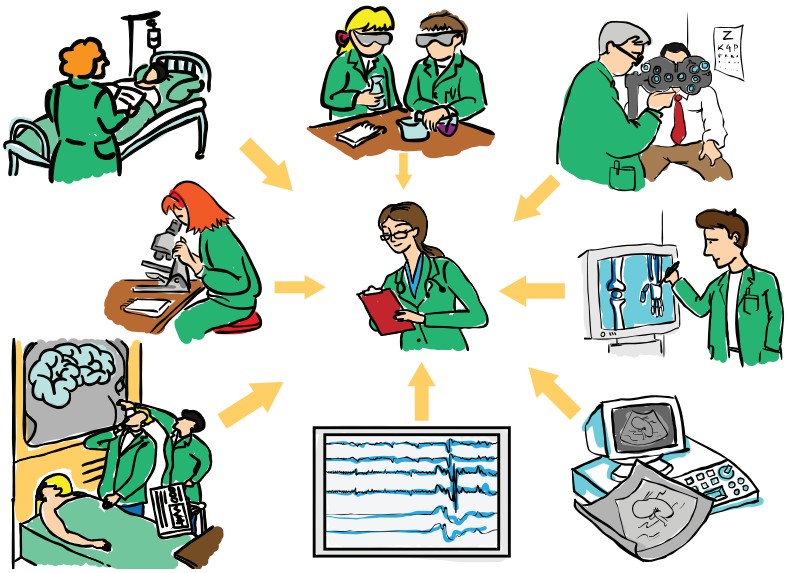
Rysunek 1. Niewielki szpital wyposażony w typowy system informatyczny służący ewidencji

Początki systemów informatyki medycznej wiążą się z pierwszymi zastosowaniami komputerów także i w innych dziedzinach, które miały miejsce w latach 50., 60. i 70. ubiegłego stulecia. Pierwszych, bardzo nieśmiałych i nieporadnych prób komputeryzacji medycyny wyśledzić dziś nie sposób. Dlatego za początek informatyki medycznej uważa się zwykle opublikowanie przez informatyka Roberta Ledleya oraz lekarza (radiologa) Lee B. Lusteda artykułu zatytułowanego *Reasoning Foundations of Medical Diagnosis* **w prestiżowym i szeroko czytanym czasopiśmie „Science". Miało to miejsce**

w roku 1959. Pierwszy szpitalny system informatyczny został eksperymentalnie uruchomiony w roku 1964 w szpitalu El Camino w Kalifornii przez firmę Technicon. Jego twórcą był Martin Lockheed.

W nowoczesnych szpitalach dostęp do danych wszystkich pacjentów jest w każdym gabinecie lekarskim, a w przyszłości będzie zapewne także bezpośrednio przy łóżku chorego. To bardzo ułatwia prowadzenie leczenia i kontrolę tego procesu.

Jednak dla nas bardziej interesujące w tym tekście będą te zastosowania komputerów w medycynie, które są zdecydowanie **odmienne** od wszelkich innych. Spróbujemy pokazać, że wiele problemów związanych z medycznymi zastosowaniami informatyki jest wysoce specyficznych. Wymienimy je teraz skrótowo, a potem omówimy kolejno.



Rysunek 2. Lekarz obecnie podczas stawiania diagnozy wspomagany jest przez wiele różnych rodzajów systemów technicznych informujących o stanie pacjenta. Praktycznie wszystkie te aparaty mają wbudowane komputery

W momencie gdy pacjent zjawia się u lekarza (w przychodni, w szpitalu lub w dramatycznych okolicznościach w karetce pogotowia) – to najważniejszym zadaniem jest ustalenie, co mu dolega i jaka jest tego przyczyna. Innymi słowy, konieczne jest rozpoznanie choroby (lub doznanych obrażeń) i postawienie diagnozy. Dawniej lekarz wykonując to zadanie miał do dyspozycji wyłącznie własne zmysły (oglądał, osłuchiwał, opukiwał pacjenta) i własną wiedzę wyniesioną ze studiów lub wynikającą z osobistego doświadczenia. Obecnie może wyko-

rzystać aparaturę, która dostarcza setek informacji o procesach toczących się w narządach i tkankach badanego pacjenta, co umożliwia dokładne wskazanie źródła i natury choroby (rys. 2). Aparatura ta jest z reguły zinformatyzowana, to znaczy sygnały i inne informacje z ciała pacjenta rejestruje komputer i dopiero w formie przeanalizowanej i przetworzonej (a więc łatwiejszej do interpretacji) przedstawia lekarzowi.

Lekarz przy ocenie stanu pacjenta, stawianiu diagnozy oraz podejmowaniu decyzji dotyczących terapii powinien kierować się zasadą określaną jako *evidence based medicine* (medycyna oparta na dowodach). Nie wnikając w szczegóły, wymaga to skonfrontowania zaobserwowanych u pacjenta objawów z najnowszymi danymi naukowymi i doświadczeniami klinicznymi, by podjąć decyzje gwarantujące osiągnięcie maksymalnej skuteczności, efektywności i bezpieczeństwa zaplanowanego leczenia. Ponadto lekarz może posłużyć się opisanymi osiągnięciami naukowymi oraz przypadkami klinicznymi, dostępnymi przez Internet w bibliotekach medycznych na całym świecie.

Po ustaleniu diagnozy i podjęciu decyzji o wyborze sposobu leczenia pacjenta – trzeba przystąpić do realizacji tego leczenia. Tutaj ponownie rola informatyki jest znacząca. **Elektroniczny rekord pacjenta** (rys. 3) może być bardzo pomocny w nadzorowaniu przebiegu leczenia, a także przy podejmowaniu w trybie roboczym kolejnych decyzji dotyczących stosowanej diety, podawanych leków i stosowanych zabiegów.

Komputer może także pomagać w obserwacji podstawowych parametrów charakteryzujących stan zdrowia pacjenta, takich jak temperatura czy ciśnienie krwi, zastępując tradycyjną kartę umieszczaną przy łóżku pacjenta. Jeszcze bardziej jest przydatne to, że z tego komputera może skorzystać pielęgniarka podająca leki lub kierująca pacjenta na specjalistyczne zabiegi. Do tego celu buduje się obecnie mobilne stanowiska robocze – specjalne terminale komputerowe w postaci wózków, które można łatwo przewozić z miejsca na miejsce lokując je kolejno przy łóżkach kolejnych pacjentów w kolejnych salach szpitalnych, gdzie można wygodnie sprawdzić wszystkie zalecania lekarskie odnoszące się danego pacjenta, można wpisać informacje dotyczące przyjętych leków, zastosowanych zabiegów, ewentualnych uwag pacjenta na temat jego aktualnego samopoczucia itp. Wspomniany wózek pozwala na obsługę komputera w pozycji stojącej (naturalnej w przypadku osoby obsługującej pacjenta), uwalnia ręce pielęgniarki i pozwala na bardzo sprawne wykonywanie jej zadań.

Na koniec warto wspomnieć o jeszcze jednym medycznym zastosowaniu komputerów, mianowicie o ich roli jako narzędzi sterujących medyczną aparaturą terapeutyczną. Coraz większa liczba metod współczesnej terapii przewiduje, że leczącym elementem jest nie tylko tabletka czy iniekcja, ale także oddziaływanie na organizm pacjenta takiej czy innej maszyny. Maszynami takimi mogą być sztuczne serce lub sztuczna nerka, a także urządzenia dozujące do wnętrza

ciała pacjenta w celach terapeutycznych określone czynniki fizyczne – na przykład pole elektromagnetyczne albo promieniowanie jonizujące. Urządzenia takie wymagają precyzyjnego sterowania, a takie sterowanie może zapewnić jedynie odpowiednio zaprogramowany komputer.

Rysunek 3. Przykładowy elektroniczny rekord pacjenta

Źródło: http://www.codeconcept.pl/uhc/img/screens/CliniNET_Screen_02.png, maj 2012.

Pierwsze sztuczne narządy powstały jeszcze w latach 40. poprzedniego wieku. Na początku była to sztuczna nerka, której wynalazcą był Holender – Willem Johan Kolff. Pierwszą pacjentką wyleczoną za pomocą sztucznej nerki była Maria Sofia Schafstadt. Zabieg miał miejsce 11 września 1945 roku. Kolff był także pierwszym człowiekiem, który zbudował i zastosował (w 1957 r.) sztuczne serce, ale jego użycie zakończyło się niepowodzeniem. Pierwsze opatentowane sztuczne serce zostało opracowane w roku 1963 przez Paula Witchella, a do produkcji przemysłowej weszło w roku 1982 sztuczne serce Roberta Jarvika (o nazwie Jarvik 7). Obecnie i sztuczne nerki, i sztuczne serca (a także inne sztuczne narządy) wyposażane są w komputerowe sterowanie.

3. Komputery w diagnostyce medycznej

Jak już wspomnieliśmy – komputery pomagają w rozpoznaniu choroby i postawieniu diagnozy. Nowoczesne przychodnie lub szpitale wyposażone są w mnóstwo aparatury, która pozwala rejestrować różne sygnały z ciała pacjenta. Istnieje cała obszerna i bardzo dziś zaawansowana dziedzina wiedzy nazywana **inżynierią biomedyczną**, która określa między innymi to, jakie sygnały można uzyskać z ciała pacjenta i jak należy je interpretować, by naprawdę pomogły one lekarzowi w podjęciu poprawnej decyzji. Nowoczesne wersje wszystkich tych aparatów diagnostycznych powiązane są z komputerami, ponieważ to znacząco polepsza ich pracę. Metody informatyczne pomagają oczyścić z zakłóceń sygnały odebrane z ludzkiego ciała, następnie automatycznie je przeanalizować, dokładniej zinterpretować, a także zachować w bazie danych w celu przyszłego wykorzystania na przykład przy śledzeniu postępów procesu uczenia.

Elektrokardiografia (EKG) jest najwcześniej odkrytą i najszerzej stosowaną metodą rejestracji i analizy sygnałów elektrycznych towarzyszących procesom życiowym w narządach wewnętrznych. Pierwsze techniczne rejestracje elektrycznej aktywności serca przeprowadził w roku 1887 brytyjski fizjolog Augustus D. Waller, jednak podstawy nowoczesnej diagnostyki EKG stworzył w roku 1889 holenderski fizjolog Willem Einthoven, i to on za swoje prace w 1924 roku otrzymał Nagrodę Nobla. Dzisiaj przy rejestracji i analizie EKG coraz ważniejszą rolę pełnią komputery.

Znajdowanie nowych algorytmów służących do automatycznej analizy różnych sygnałów biomedycznych, gdy już znajdą się w komputerze, jest jednym z ciekawszych obszarów współczesnej informatyki. Możliwość zastąpienia przez dobrze przemyślany algorytm spostrzegawczego oka i mądrego umysłu specjalisty, pozwala na korzystanie z tych badań również przez mniej doświadczonego lekarza. Ma to duże znaczenie dla wczesnego wykrywania chorób, a wczesna diagnoza z reguły radykalnie zwiększa szanse na skuteczne leczenie. Co więcej, dobrze zbudowany program umieszczony przykładowo w smartfonie, wraz z prostą przystawką pozwalającą na wprowadzanie do tego smartfonu na przykład sygnału EKG (albo innych sygnałów życiowo ważnych dla danego pacjenta) – umożliwia stały nadzór nad stanem zdrowia, w razie potrzeby ostrzegając użytkownika o pojawiających się zagrożeniach. Na rysunku 4 pokazano przykładowe rozwiązanie tego typu, dzięki któremu pacjent może przebywać w domu i normalne funkcjonowanie (także w pracy zawodowej) – zamiast leżeć w szpitalu na obserwacji. Serce pacjenta jest stale pod kontrolą oprogramowania

umieszczonego w komputerze typu palmtop lub w smartfonie. Gdy serce pracuje poprawnie, to komputer ogranicza się do nadzoru, ale nie podejmuje żadnych dodatkowych działań. Jednak gdy obserwowany sygnał pracującego serca zacznie wykazywać nieprawidłowości, to program zawarty w pamięci tego wielofunkcyjnego urządzenia podejmie działania ratunkowe. Najpierw ostrzeże użytkownika, że powinien na przykład ograniczyć wysiłek, usiąść, położyć się. Potem, jeśli niepokojące objawy nie ustąpią, może automatycznie połączyć go z lekarzem kardiologiem, który przekaże bardziej szczegółowe zalecenia. Wreszcie w przypadku utrzymującego się lub pogłębiającego kryzysu wskazującego na poważną niesprawność działania serca – smartfon sam wezwie pogotowie, skutecznie informując, gdzie pacjent się obecnie znajduje, co sam ustali nawet w przypadku utraty przytomności przez pacjenta – dzięki sygnałom GPS.

Rysunek 4. Komputerowy system monitorowania serca umieszczony w smartfonie.

Źródło: http://b4tea.com/wp-content/uploads/2010/10/wearable-ecg-system.jpg, maj 2012.

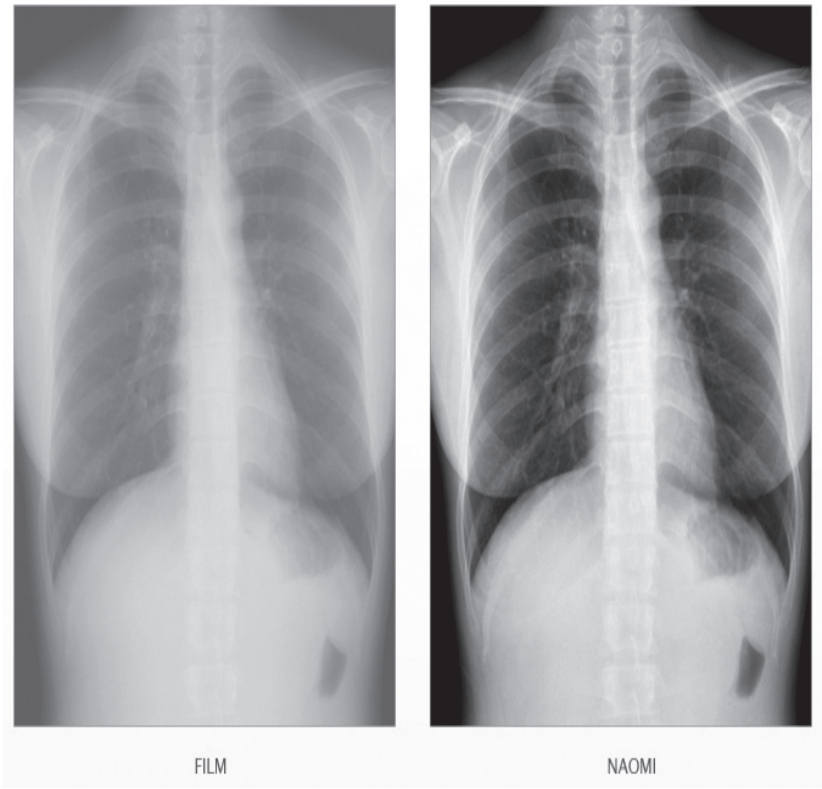
Opisany wyżej przykład zastosowania informatyki w specjalistycznej aparaturze diagnostycznej należy do obszernej dziedziny z pogranicza informatyki i medycyny określanej jako **telemedycyna**, o której jest mowa w dalszej części.

Obok sygnałów związanych z pracą serca, skomputeryzowana aparatura diagnostyczna może odbierać, rejestrować i pomagać interpretować jeszcze mnóstwo innych sygnałów związanych z pracą mózgu, narządów zmysłów (wzrok, słuch, dotyk), mięśni, organów wewnętrznych itp. Jest to obecnie jeden z najszybciej rozwijających się obszarów zarówno samej informatyki, jak i metrologii (nauki o czujnikach pomiarowych) oraz skojarzonej z nimi obiema inżynierii biomedycznej.

4. Komputerowe pozyskiwanie i analiza zobrazowań medycznych

Przez całe stulecia wygląd narządów wewnętrznych badanego pacjenta był dla lekarza zagadką. Metodą palpacyjną (badanie dotykiem), opukiwaniem,

osłuchiwaniem, wnioskowaniem na podstawie różnych danych pośrednich itp. – lekarze tworzyli sobie pogląd na temat tego, jak wygląda taki czy inny narząd. W szczególności starali się ustalić, czy jest on zniekształcony przez chorobę, a jeśli tak, to w jakim miejscu i w jakim stopniu. Jednak było to mało dokładne i często dopiero podczas operacji przekonywali się, czy te domysły były trafne.



Rysunek 5. Przykładowy obraz rentgenowski ujawnia zalety oraz wady tej formy obrazowania

Źródło: http://www.rfamerica.com/images/naomi/tech/minimum/radi_sample_zoom.jpg, maj 2012.

Radykalną poprawę sytuacji przyniósł wynalazek obrazowania rentgenowskiego. Prześwietlenie ciała pacjenta promieniami X umożliwia przedstawienie narządów wewnętrznych w taki sposób, że na matrycy elektronicznych detektorów promieniowania (kiedyś na kliszy fotograficznej) ujawnia się ich zarys w postaci cienia (rys. 5). Obraz ten powstaje na skutek fizycznego zjawiska pochłaniania promieni X przez narządy ciała pacjenta. Tam, gdzie narządy silnie pochłaniają promieniowanie (na przykład kości) – pojawia się widoczny jasny cień. W miejscach słabszego pochłaniania (na przykład płuca) – obraz jest

szary. Tam gdzie promieniowanie praktycznie w ogóle nie przenika (powietrze poza obrysem ciała pacjenta) – obraz jest intensywnie czarny.

Punktem wyjścia do całej nowoczesnej radiologii (czyli nauki o obrazowaniu narządów wewnętrznych) było odkrycie w roku 1895 przez Wilhelma Roentgena tajemniczego promieniowania, które przez odkrywcę nazwane było promieniowaniem X (pod tą nazwą znane jest w większości krajów świata), a w Polsce i w Niemczech nazywane jest promieniowaniem rentgenowskim. Za odkrycie możliwości „zaglądania do wnętrza ciała" za pomocą promieniowania X Roentgen w roku 1901 został uhonorowany pierwszą Nagrodą Nobla w dziedzinie fizyki. Dziś obrazy rentgenowskie są nadal w użyciu, ale rejestracją i obróbką tych obrazów zajmują się komputery.

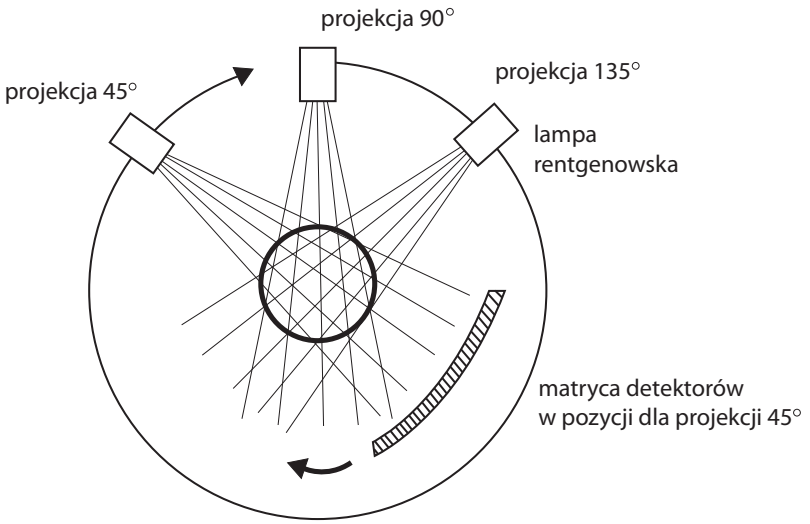
Niestety typowy obraz rentgenowski ma wiele wad. Po pierwsze zarysy narządów są na nim mało wyraźne. W centralnym punkcie obrazu widać kształt serca, ale jest on niewyraźny i mało czytelny. Narządów jamy brzusznej (poniżej przepony, w dolnej części zdjęcia) nie widać prawie wcale. Po drugie narządy przesłaniają się wzajemnie – na rysunku 5 cienie żeber i kręgosłupa nakładają się na obraz serca pogarszając i tak nie najlepszą możliwość oceny prawidłowości jego budowy. Najgorsza sytuacja jest z głową. Masywna czaszka ukrywa wewnątrz najcenniejszy narząd człowieka – mózg. Na tradycyjnym zdjęciu rentgenowskim praktycznie wcale nie można go zobaczyć. Żeby zróżnicować na przykład szarą i białą substancję mózgu trzeba użyć tzw. miękkiego promieniowania rentgenowskiego (powstającego, gdy aparat prześwietlający jest zasilany niższym napięciem). Ale takie miękkie promienie X są w całości pochłaniane przez kości czaszki i nie dostaną się do jej wnętrza. Gdy zaś zastosuje się promieniowanie twarde (powstające przy wysokim napięciu zasilającym aparat) – to przenikną one przez czaszkę, ale będą miały tak dużą energię, że przelecą przez struktury mózgowe praktycznie wcale nie ulegając pochłanianiu, więc nie dając potrzebnego diagnostycznie cienia.

Mimo wymienionych wad techniki rentgenowskiej była ona używana przez ponad pół wieku jako główne źródło wiedzy na temat narządów wewnętrznych ciała człowieka – po prostu nie było innej metody.

Pierwszy użyteczny praktycznie tomograf komputerowy zbudował w roku 1968 Godfrey Newbold Hounsfield, który otrzymał za to w roku 1979 Nagrodę Nobla. To była sensacja – po raz pierwszy Nagrodę Nobla w dziedzinie medycyny otrzymał infor-

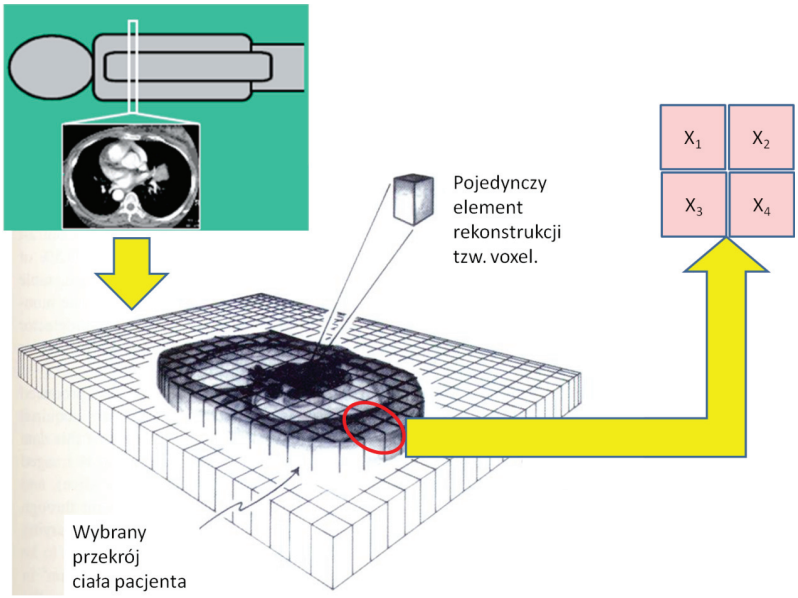
matyk. Co ciekawe, samą zasadę działania tomografu i algorytm jego działania opracował jeszcze w roku 1917 austriacki matematyk Johann Radon, jednak nie mógł swej metody wypróbować w praktyce, gdyż dla uzyskania obrazu tomograficznego trzeba rozwiązać układ kilkuset tysięcy równań o kilkuset tysiącach niewiadomych, co bez komputerów było niewykonalne.

Przełom nastąpił za sprawą informatyki. Mianowicie zbudowano i praktycznie zastosowano tomograf komputerowy. W **tomografie** obraz wnętrza ciała człowieka powstaje poprzez komputerową interpretację wyników wielokierunkowego prześwietlenia ciała pacjenta cienkimi promieniami rentgenowskimi. Przy każdym takim prześwietleniu mierzone jest natężenie promieniowania wysyłanego oraz po przejściu przez ciało pacjenta. W ten sposób otrzymuje się dokładną miarę łącznego stopnia pochłaniania promieniowania X na drodze tego cienkiego promienia. Za to łączne pochłanianie odpowiedzialne są wszystkie narządy, przez które promień przechodzi. Gdyby mieć tylko jeden taki wynik – byłby w istocie mało przydatny. Ale ponieważ tomograf gromadzi bardzo wiele takich danych, pochodzących od prześwietlania ciała pacjenta w różnych kierunkach i pod różnymi kątami – każdy punkt wewnątrz ciała wchodzi (jako niewiadoma) do wielu wyników pomiarów łącznego pochłaniania na różnych drogach (rys. 6).



Rysunek 6. Zbieranie danych do tworzenia obrazu wnętrza ciała w tomografii komputerowej. Opis w tekście

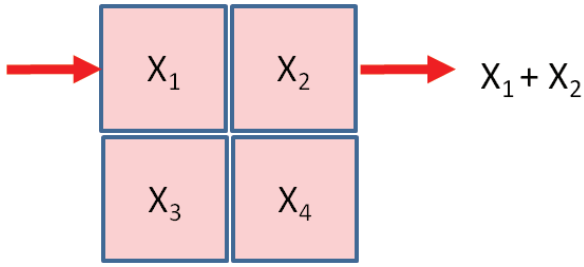
Powstaje w ten sposób układ wielu równań, w których znane są sumaryczne pochłaniania, a nieznane są pochłaniania w różnych konkretnych punktach wewnątrz ciała pacjenta. Rozważmy uproszczony przykład przedstawiony na rysunku 7.



Rysunek 7. Zasada tworzenia obrazu w tomografie komputerowym

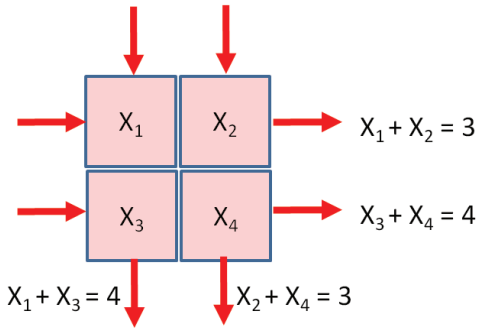
Przedstawiono tym ciało pacjenta, o którym tomograf zbiera informacje w postaci przekroju wybraną płaszczyzną. W tej płaszczyźnie wyróżnia się małe fragmenty ciała człowieka (lub otaczającej przestrzeni), które odpowiadają ogólnie znanym pikselom na obrazach, ale ponieważ są to elementy trójwymiarowe – nazywa się je voxelami (lub wokselami). Komputer opracowujący dane z tomografu ustala, jak dużą zdolność pochłaniania promieniowania rentgenowskiego ma każdy voxel. Pokażemy, jak się to odbywa, ograniczając nasze działania do czterech wyodrębnionych voxeli, pokazanych w prawej górnej części rysunku.

Każdy z rozważanych voxeli cechuje się jakąś (nieznaną!) zdolnością pochłaniania promieniowania rentgenowskiego. Te zdolności pochłaniania oznaczymy jako niewiadome *x1, x2, x3, x4*. Żeby ustalić wartości tych niewiadomych przepuszczamy przez ciało pacjenta w różnych miejscach i w różnych kierunkach wąskie wiązki promieniowania rentgenowskiego, mierząc każdorazowo dokładnie natężenie promieniowania wchodzącego do ciała i natężenie promieniowania wychodzącego po drugiej stronie. W ten sposób można ustalić sumaryczny stopień pochłaniania promieniowania na pewnej wybranej drodze (rys. 8).



Rysunek 8. Sumaryczny stopień pochłaniania promieniowania na drodze obejmującej voxele numer 1 i 2

Jeśli taki sumaryczny stopień pochłaniania promieniowania ustalimy (na drodze pomiaru) na każdej rozważanej drodze – to można będzie ułożyć układ równań, w których jako niewiadome wystąpią wartości stopnia pochłaniania w poszczególnych rozważanych voxelach *x1, x2, x3, x4*, a jako wartości wiadome wystąpią te zmierzone sumaryczne stopnie pochłaniania na poszczególnych drogach (rys. 9). Ponieważ w przykładzie rozważane są zaledwie cztery voxele – wystarczą cztery sondowania (prześwietlenia wąską wiązką promieni w dwóch położeniach i w dwóch kierunkach).



Rysunek 9. Wyniki dwukierunkowego sondowania wybranej grupy voxeli.

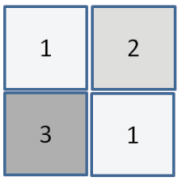
Z wyników sondowań można ułożyć układ równań, którego użyjemy do określenia nieznanych wartości stopnia pochłaniania w poszczególnych rozważanych voxelach *x1, x2, x3, x4* . Oto on:

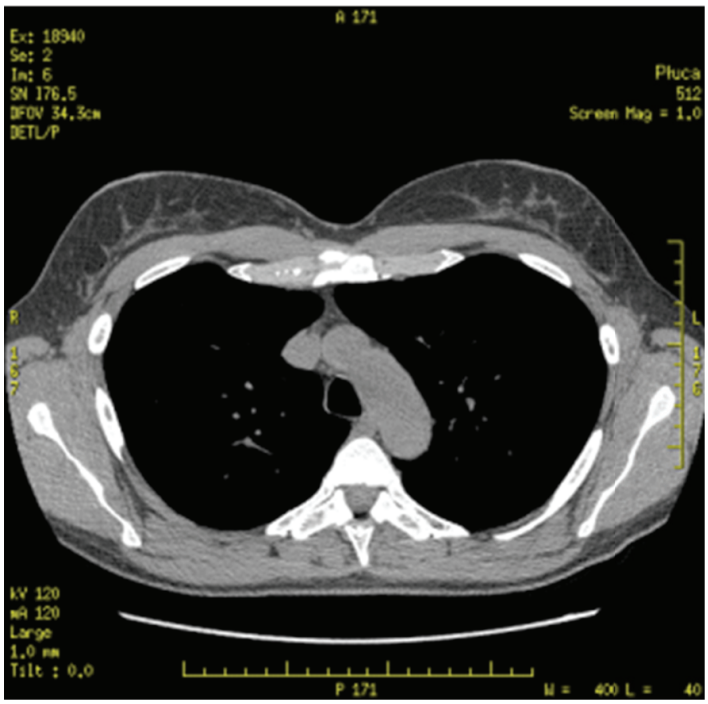
$$x1 + x2 = 3$$
$$x3 + x4 = 4$$
$$x1 + x3 = 4$$
$$x2 + x4 = 3$$

Rozwiązując ten układ równań otrzymamy wartości:

$x1 = 1$
$x2 = 2$
$x3 = 3$
$x4 = 1$

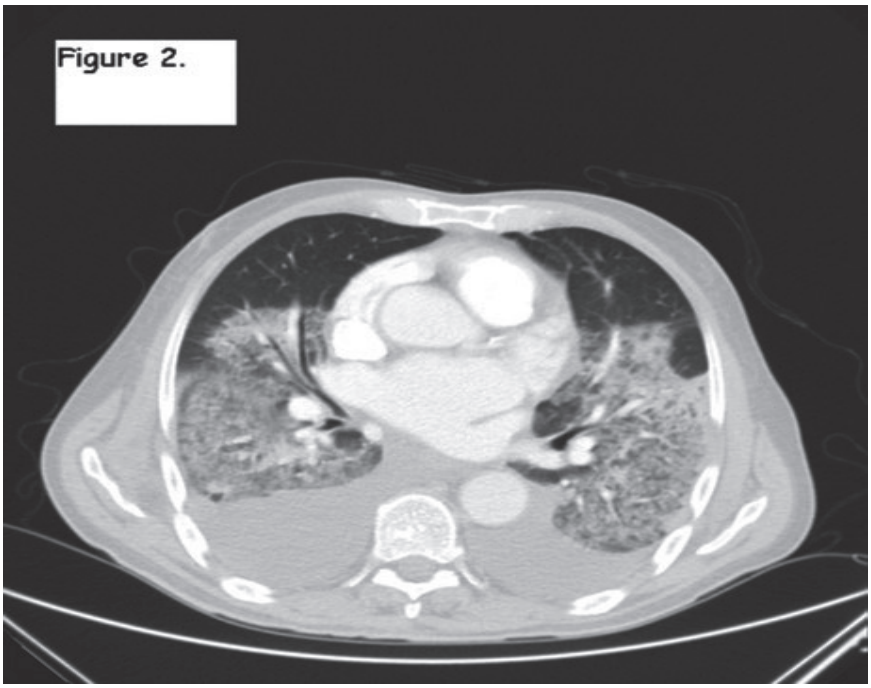
Przedstawiając powyższe wartości jako punkty o zróżnicowanej szarości mamy dla rozważanego maleńkiego kawałka ciała pacjenta obraz pokazany na rysunku 10 po lewej stronie. Wykonując takie same czynności, jak wymienione wyżej, dla dziesiątków tysięcy voxeli wchodzących w skład rozważanego przekroju ciała człowieka musimy użyć dziesiątków tysięcy sondowań, powstających w sposób zasygnalizowany na rysunku 6. Efektem jest układ dziesiątków tysięcy równań. W każdym równaniu niewiadoma oznaczająca pochłanianie promieni rentgenowskich przez pewien konkretny punkt występuje wraz z niewiadomymi oznaczającymi to pochłanianie w innych punktach, ale w każdym równaniu jest brana pod uwagę inna kolekcja punktów, bo każde równanie jest zapisem pomiaru dokonanego przez promień biegnący przez ciało pacjenta w innym kierunku. Po zgromadzeniu wszystkich tych równań i rozwiązaniu wszystkich niewiadomych – wynik tych obliczeń można pokazać jako obraz. W efekcie z wielu punktów o zróżnicowanej szarości powstaje taki obraz, jak przedstawiono na rysunku 10 po prawej stronie.





Rysunek 10. Przekształcenie wyników obliczeń komputerowych na czytelny obraz przekroju ciała

Obraz, który w ten sposób powstaje, jest odmienny od obrazu uzyskiwanego w tradycyjnym aparacie rentgenowskim. Po pierwsze przedstawia on wnętrze ciała człowieka i znajdujące się tam narządy w przekroju. Dzięki temu nic niczego nie przesłania i każdy narząd widoczny jest dokładnie i osobno. Można to zobaczyć na rysunku 11, który przedstawia wnętrze klatki piersiowej (analogicznie jak rys. 5), ale w tym przypadki wszystkie struktury (kręgosłup, żebra, serce, płuca) widać wyraźnie osobno. Co więcej, dzięki zobrazowaniu narządów wewnętrznych w formie przekroju można wyraźnie zobaczyć wewnętrzną budowę serca oraz strukturę płuc, czego na zwykłym zdjęciu rentgenowskim trudno się doszukać.



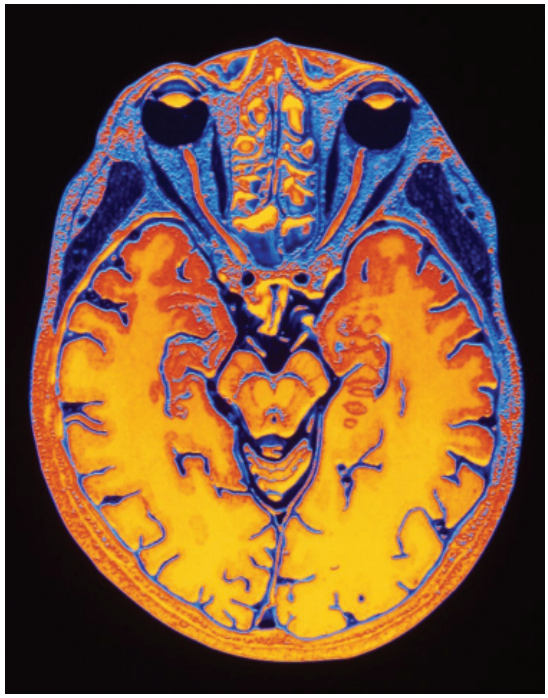
Rysunek 11. Obraz tomograficzny wnętrza klatki piersiowej

Źródło: http://www.ispub.com/journal/the-internet-journal-of-thoracic-and-cardiovascular-surgery/volume-9-number-2/a-fatal-case-with-diffuse-alveolar-hemorrhage-due-to-tirofiban-and-clopidogrel-therapy.article-g02.ns.jpg, maj 2012.

Druga zaleta obrazu uzyskiwanego za pomocą tomografii komputerowej związana jest z tym, że można na nim uzyskać bardzo dokładne zarysowanie także tych obiektów, w których pochłanianie promieniowania rentgenowskiego bardzo nieznacznie różni się od ich otoczenia. Na tradycyjnym zdjęciu rentgenowskim takie obiekty będą bardzo słabo widoczne lub wręcz niewidoczne, bo ich szarość będzie prawie taka sama jak szarość otoczenia. Natomiast w tomografii komputer rozwiązujący odpowiednie równania stwierdzi, że w tym punkcie jest taka wartość pochłaniania, a w innym punkcie wartość ta jest inna, więc

może także tę mocno subtelną różnicę przedstawić bardzo wyraziście, odwołując się (jeśli potrzeba) także do odwzorowania barwnego (różnym stopniom pochłaniania promieniowania rentgenowskiego przyporządkowane zostaną różne barwy). Przykład takiego sztucznie wybarwionego obrazu tomograficznego jest przedstawiony na rysunku 12.

Zastosowanie sztucznego barwienia obrazów z tomografu komputerowego uzasadnione jest tym, że oko ludzkie odróżnia tylko około 60 poziomów szarości, a w wyniku obliczeń komputerowych punkty na obrazie tomograficznym reprezentują zdolność pochłaniania promieniowania wyrażaną liczbowo w przedziale od –1000 do +1000. W związku z tym, na obrazie tomograficznym trzeba odwzorować rozpiętość jasności pikseli wynoszącą 2000 tak zwanych jednostek Hounsfielda (HU), czego samą tyko skalą jasności (albo szarości) zrobić się po prostu nie da. Nawiasem mówiąc, warto zauważyć na rysunku 12 jak perfekcyjnie tomografia komputerowa potrafi odtworzyć niedostępne dla zwykłego rentgena wnętrze czaszki człowieka, prezentując budowę mózgu, a nawet wnętrze gałek ocznych (widoczne są soczewki).

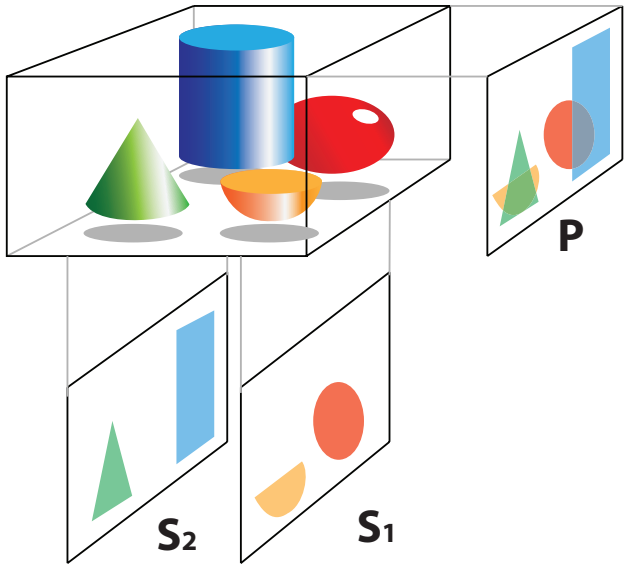


Rysunek 12. Obraz tomograficzny ma w wyniku obliczeń komputerowych tak wiele poziomów wartości pikseli, że do ich odwzorowania używa się nie tylko jasności, ale dodatkowo także kolorów

Źródło: http://i.dailymail.co.uk/i/pix/2008/10/25/article-1080553-02380787000005DC-299_233x310_popup.jpg, maj 2012

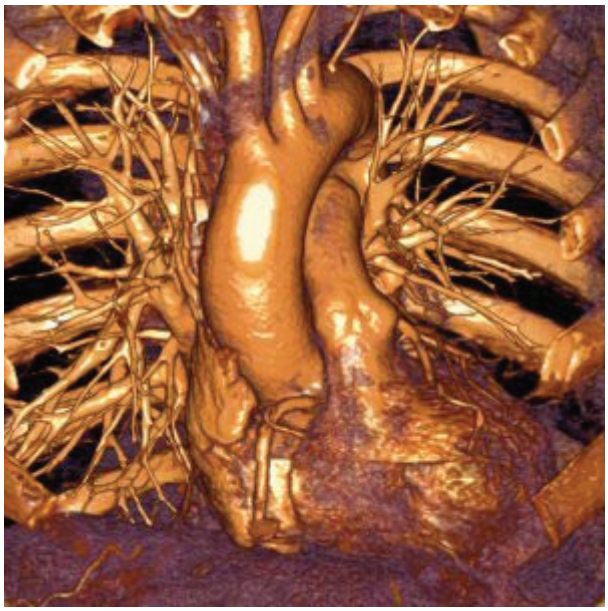
Zadanie komputera w tomografii do łatwych nie należy. Liczba punktów wewnątrz ciała pacjenta, których zdolność pochłaniania promieniowania rentgenowskiego usiłujemy ustalić na drodze rozwiązania wspomnianych równań, bywa ogromna. Kilkadziesiąt tysięcy takich punktów (pikseli) to wartość minimalna, a bywa, że jest ich kilka milionów. Próba rozwiązania tak wielkiej liczby równań o tak ogromnej liczbie niewiadomych nawet dla współczesnych komputerów stanowi poważny problem. Dlatego jest tu pole do popisu dla młodych informatyków!

Komputer przy obrazowaniu medycznym ma obecnie jeszcze jedną rolę. Tradycyjna rentgenografia pozwalała przedstawić narządy wewnętrzne pacjenta w formie projekcji (rzutu na płaszczyznę), gdzie niestety narządy te przesłaniały się wzajemnie, co utrudniało ich analizę i diagnostyczną interpretację (patrz rysunek 13 – zobrazowanie P po prawej stronie).



Rysunek 13. Zobrazowanie tej samej makiety zawierającej wewnątrz kilka elementów w układzie obrazu płaskiego (P), przekrojów (S1 i S2) oraz w formie trójwymiarowej rekonstrukcji

Z kolei tomografia komputerowa dostarcza informacji w postaci przekrojów ciała pacjenta wybranymi płaszczyznami (patrz zobrazowania S1 i S2 w dolnej części rysunku 13) – ale przydatność tych przekrojów jest ograniczona i mocno zależy od tego, czy płaszczyzna przekroju trafi wewnątrz obiektu na któryś z istotnych elementów – czy też je ominie.



Rysunek 14. Trójwymiarowa rekonstrukcja określonych struktur anatomicznych. Jest to podobnie jak na rysunkach 5 i 11 wnętrze klatki piersiowej z sercem i żebrami – ale jakże inaczej przedstawione!

Źródło: http://regmedia.co.uk/2004/09/29/chest_image.jpg, maj 2012

Kompletną informację można uzyskać, gdy dokona się trójwymiarowej rekonstrukcji badanego obiektu (w medycynie – wnętrza ciała pacjenta). I właśnie takie trójwymiarowe rekonstrukcje wybranych fragmentów ciała człowieka potrafią tworzyć komputery – łącząc w tym przypadku elementy analizy obrazów pozyskiwanych na przykład za pomocą tomografii oraz elementy grafiki komputerowej pozwalającej wynik rekonstrukcji odpowiednio prezentować (rys. 14).

Warto dodać, że współczesne systemy komputerowego przetwarzania obrazów medycznych nie tylko pomagają pozyskiwać obrazy narządów wewnętrznych, ale także gromadzą je w szpitalnej bazie danych. Służą do tego specjalne systemy RIS – *Radiology Information System* oraz PACS – *Picture Archiving* and *Communication System*. W szpitalu wyposażonym w taki system oraz w bezprzewodową sieć komputerową lekarz może skorzystać z obrazu wnętrza ciała leczonej osoby w każdej chwili i w każdym miejscu.

Pierwszy system komputerowy przeznaczony do centralnego gromadzenia i zarządzania obrazami medycznymi stworzył w roku 1979 profesor Heinz Lamke na uniwersytecie technicznym w Berlinie. System ten gromadził obrazy z tomografu kom-

puterowego, pozwalał je przetwarzać i zawierał większość elementów wchodzących w skład współczesnych systemów typu PACS łącznie z interfejsem, który łączył ten system z istniejącym wcześniej systemem szpitalnym.

5. Informatyka we wspomaganiu terapii

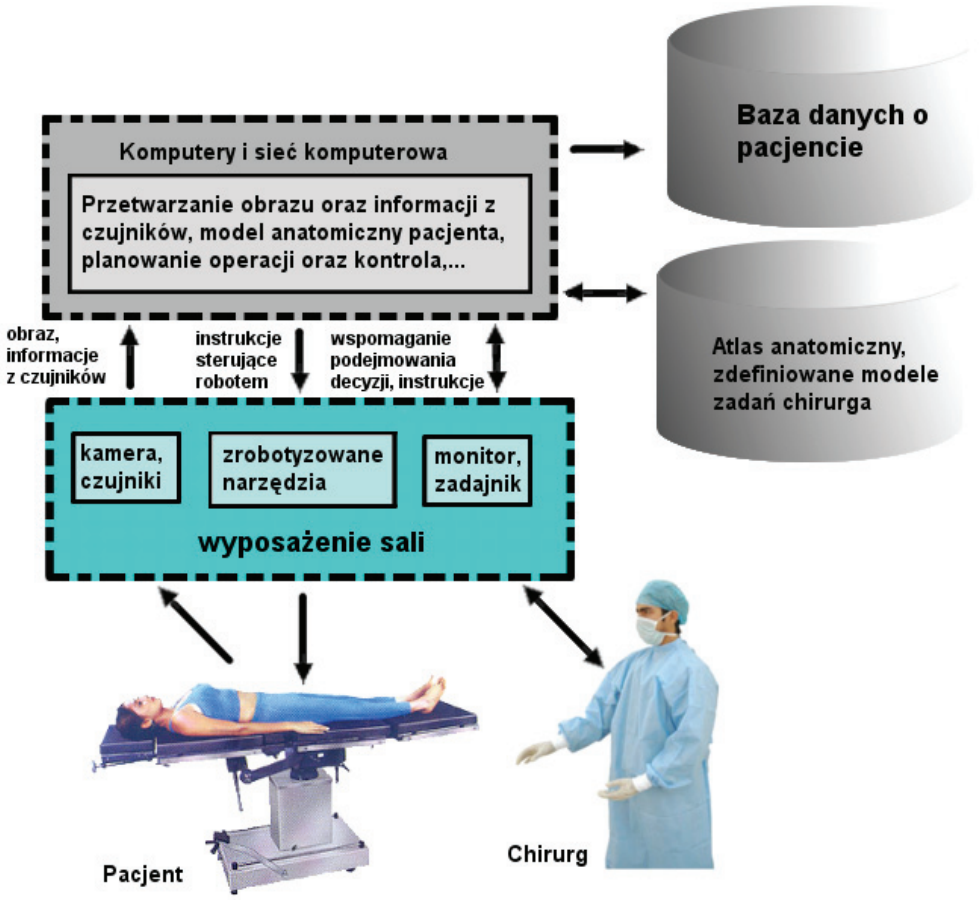
Po postawieniu diagnozy, w czym jak wspomniano wyżej systemy informatyczne mogą być bardzo pomocne, następuje proces leczenia pacjenta. Tutaj także systemy komputerowe i informatyka mogą być bardzo pomocne. Typowe leczenie polega na podawaniu określonych farmaceutyków i obserwowaniu skutków ich działania. Komputer może tu gromadzić dane o zaleceniach lekarzy, kontrolować sumienność i terminowość ich wykonywania oraz monitorować ilościowe (mierzone za pomocą specjalnej aparatury) oraz jakościowe (opisywane przez personel medyczny lub przez samego pacjenta) efekty działania podawanych leków. Jednak w tym zakresie stopień użyteczności informatyki medycznej jest ograniczony i w związku z tym cała ta sfera działania medycznego jest w tym tekście pominięta.

Pierwsze wykorzystanie robota do celów chirurgicznych miało miejsce, gdy w roku 1985 dr Yik San Kwoh przeprowadził biopsję mózgu za pomocą odpowiednio przystosowanego robota przemysłowego PUMA 560. W roku 1988 w Imperial College w Londynie zbudowano specjalnego robota do celów chirurgicznych. Nazywał się PROBOT i był stosowany w Guy's and St Thomas' Hospital w Londynie przez dra Senthila Nathana.

Z odmienną sytuacją mamy do czynienia w momencie, gdy zamiast leczenia farmakologicznego lekarz decyduje się na zabieg operacyjny. Obecnie większość operacji chirurgicznych wykonuje się ręcznie, a komputerowa aparatura na sali operacyjnej głównie służy do dostarczania chirurgom potrzebnych informacji (między innymi omawianych wyżej zobrazowań wnętrza ciała pacjenta) oraz do monitorowania stopnia uśpienia pacjenta (jako konsola anestezjologa).

Jednak ta sytuacja powoli ulega zmianie w związku z rozpowszechnianiem się techniki robotów chirurgicznych.

Ważną zaletą robotów chirurgicznych jest to, że sterujący robotem lekarz może być daleko od sali operacyjnej, na której wykonywany jest zabieg, a jednak to on w istocie przeprowadza operację. Pierwszy taki zdalny zabieg został przeprowadzony w 2001 roku. Chirurg profesor Jacques Marescaux przebywający w Nowym Jorku wykonał operację (cholecystektomię) przy użyciu robota ZEUS u pacjentki przebywającej na sali operacyjnej w szpitalu w Strasburgu we Francji.



Rysunek 15. Rola komputera w zrobotyzowanym stanowisku chirurgicznym. Rysunek zaczerpnięty za zgodą autora z książki Leszka Podsędkowskiego *Roboty medyczne – budowa i zastosowanie* **(WNT, 2010)**

Sam zabieg na ciele pacjenta wykonuje robot, który jest sterowany przez komputer. Chirurg siedzący wygodnie przy specjalnej konsoli sterującej za pomocą trzymanych w rękach manipulatorów wysyła swoje polecenia do komputera, które ten przekłada na sygnały sterujące dla robota. Sygnały te są wysyłane, gdy lekarz wykonuje swymi rękami taki ruch, jaki powinno wykonać narzędzie chirurgiczne

Na ogół ruchy rąk operatora w takim systemie są znacznie obszerniejsze, niż ruchy narzędzi chirurgicznych przymocowanych do ramion robota. Powoduje to, że robot wykonuje narzędziami chirurgicznymi bardzo subtelne i precyzyjne działania wewnątrz ciała operowanego pacjenta, a lekarz nie musi wkładać w to działanie tak wiele wysiłku ani tak wiele uwagi, jakie musiałby angażować, gdyby sam przeprowadzał zabieg. Co więcej, komputer przekazujący polecenia lekarza do robota może eliminować różne niepotrzebne zjawiska – na przykład drżenie rąk chirurga.

Dla nas w tym artykule kluczowe znaczenie ma rola komputera w omawianym systemie. Jest ona uwidoczniona na rysunku 15 i polega na tym, że komputer odbiera, przetwarza, rejestruje i analizuje wszystkie sygnały pojawiające się w systemie – zarówno te pochodzące z wnętrza ciała operowanego pacjenta (obrazy z kamer endoskopowych, czujniki dotyku, siły, momentu obrotowego itp.), sygnały pochodzące od robota (informacje o położeniu i przemieszczeniach poszczególnych części robota i trzymanych przez niego narzędzi chirurgicznych), jak i sygnały pochodzące od lekarza manipulującego dżojstikami i innymi elementami sterującymi. Na podstawie tych danych, a także na podstawie ogólnych danych medycznych i szczegółowych danych o konkretnym pacjencie – komputer opracowuje i wysyła szczegółowe instrukcje sterujące robotem, a także pełni rolę doradczą w stosunku do przeprowadzającego operację lekarza.

Operacja wykonywana z pomocą robota chirurgicznego jest wygodniejsza dla lekarza. Zamiast stać pochylony nad ciałem pacjenta na stole operacyjnym – siedzi wygodnie przy konsoli i widzi pole operacyjne na monitorach pokazujących obraz z wnętrza ciała pacjenta w sposób trójwymiarowy i w dużym powiększeniu. Dzięki temu lekarz może bez nadmiernego zmęczenia wykonać więcej operacji, a jakość tych operacji jest lepsza w stosunku do tych wykonywanych tradycyjnie.

Operacja przeprowadzona z pomocą robota chirurgicznego jest też korzystniejsza dla pacjenta. Narzędzia chirurgiczne przymocowane do ramion robota wnikają do wnętrza ciała pacjenta przez niewielkie (kilkumilimetrowe) otwory w czaszce, ścianach klatki piersiowej lub w powłokach brzusznych. Powoduje to minimalizację zranienia, jakiego doznaje pacjent w wyniku przeprowadzonej operacji, ogranicza utratę krwi i znacząco skraca czas pobytu w szpitalu. Co więcej, poddanie się operacji wykonanej z użyciem robota istotnie przyspiesza

moment, kiedy pacjent skutecznie wyleczony będzie mógł powrócić do swoich obowiązków zawodowych czy do rodziny.

A wszystko dzięki mądremu zastosowaniu informatyki w medycynie.

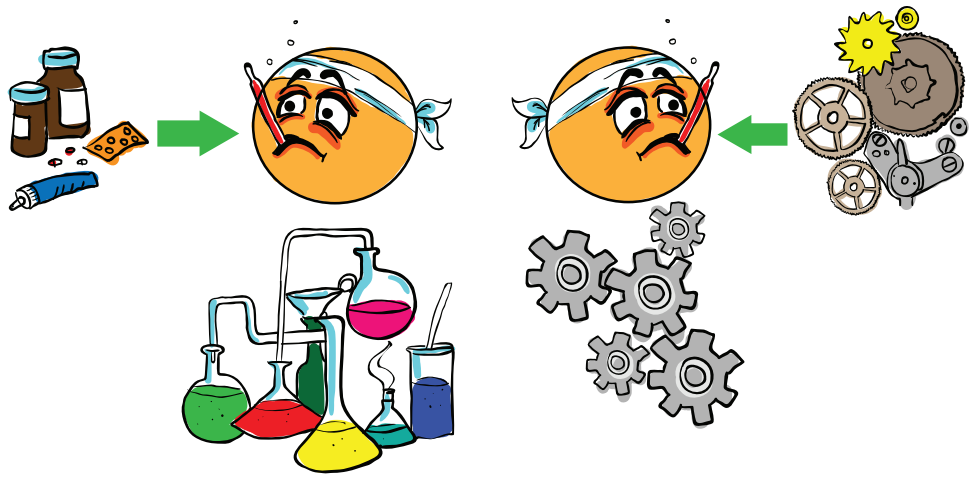
6. Leczenie czynnikami fizycznymi i komputerowe wspomaganie tego leczenia

W poprzednim rozdziale wskazano na rolę komputera w leczeniu metodami znanymi i stosowanymi od wieków: farmakoterapią i chirurgią. Postęp inżynierii biomedycznej doprowadził jednak do tego, że obecnie w asortymencie metod i technik leczniczych pojawiły się także różne czynniki fizyczne, którymi lekarze usiłują naprawiać uszkodzoną przez chorobę „maszynerię" ludzkiego ciała.

Zatrzymajmy przez moment uwagę na ukrytym sensie ostatniego zdania. Przez całe stulecia powszechnie rozważano organizm ludzki jako swoisty reaktor chemiczny, w którym zachodzą różne procesy. Procesy te można obserwować poprzez oznaczanie w tkankach zawartości określonych substancji chemicznych, dlatego przy badaniu pacjenta i stawianiu diagnozy lekarz zleca różne biochemiczne badania krwi i innych płynów ustrojowych. Poznaje w ten sposób aktualny stan tego reaktora chemicznego, jakim jest ludzkie ciało. Od stuleci, wręcz od kilku tysięcy lat ludzie wierzą w to, że na zjawiska zdrowia i choroby można wpływać podając pacjentowi różne leki. Dawniej dobór tych leków był dyktowany obserwacją przyrody (na przykład chore zwierzęta zjadały różne zioła i zdrowiały, więc lekarze je również stosowali u ludzi), ale obecnie, gdy poznaliśmy procesy biochemiczne zachodzące w naszym organizmie – leki można dobierać naukowo. Traktując ciało człowieka w ten sposób można starać się regulować jego pracę poprzez dodawanie do tego reaktora chemicznego różnych substancji. Skutkiem obowiązywania tego paradygmatu jest praktyka leczenia chorób poprzez dostarczanie do organizmu określonych substancji chemicznych (farmakologicznych), których liczba i częstość stosowania ostatnio niepokojąco rośnie. Pacjenci w starszym wieku zażywają całe garście różnych pastylek wydając na ich zakup coraz więcej pieniędzy...

W ostatnich latach prace badawcze wykazały jednak, że obok biochemicznego modelu funkcjonowania organizmu człowieka, którego przydatności nikt nie kwestionuje, ale który dotarł w wielu przypadkach do kresu swego skutecznego działania, możliwe i celowe jest korzystanie w medycynie także z modelu biofizycznego. W tym drugim przypadku ciało ludzkie jest traktowane jako skomplikowana maszyna, w której przebiegają różne procesy – nierzadko wymagające sterowania lub wspomagania czynnikiem fizycznym, a nie chemicznym. Przy tym modelu choroba jest traktowana jako niesprawność jednego lub kilku

wewnętrznych mechanizmów, jako zaburzenie dotykające określonych procesów biofizycznych, charakterystycznych dla stanu dobrego zdrowia. Skoro wnętrze ciała człowieka to maszyna, to czynnikiem leczącym staje się także maszyna wytwarzająca i wprowadzająca do organizmu pacjenta odpowiedni czynnik fizyczny (rys. 16).



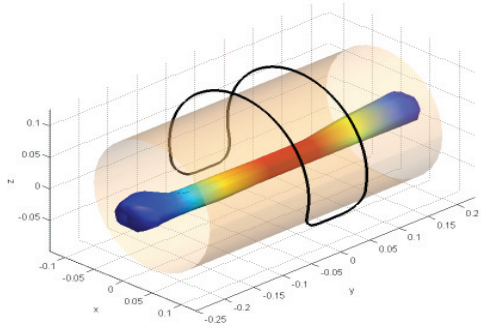
Rysunek 16. Dwa sposoby traktowania organizmu chorego człowieka oraz jego leczenia – biochemiczny i biofizyczny (opis w tekście)

Terapia przy pomocy czynników fizycznych nie jest wyłącznie osiągnięciem najnowszej medycyny, gdyż bywała ona stosowana w przeszłości, chociaż na ogół na zasadzie intuicyjnego poszukiwania czynnika leczniczego wśród oddziaływań dostępnych za pomocą nawet najprostszych środków. Przykładowo można podać, że w medycynie chińskiej od 3000 lat używano akupunktury, starożytni lekarze greccy stosowali gorące i zimne kąpiele taktując ciepło jako czynnik leczący, kapłani egipscy wykorzystywali kamienie magnetyczne jako źródło magnetoterapii, a w starożytnym Rzymie używano elektrycznych ryb zdolnych (np. drętwa) do leczenia tych schorzeń, które dziś też leczymy elektrycznymi impulsami generowanymi przez elektroniczną aparaturę.

O ograniczonej przydatności informatyki w odniesieniu do sfery podejścia biochemicznego i leczenia farmakologicznego była już mowa. Natomiast przy podejściu biofizycznym i leczeniu z zastosowaniem specjalnych maszyn komputery mogą być bardzo użyteczne. Żeby nie rozpraszać uwagi na zbyt wiele

różnych wątków skupimy uwagę na leczeniu przy wykorzystaniu pól elektromagnetycznych, a konkretnie na jednej z technik leczenia metodami elektromagnetycznymi, a mianowicie magnetoterapii. Jest to metoda leczenia pulsującym polem magnetycznym niskiej częstotliwości, które za pomocą specjalnych aplikatorów wprowadza się do wybranych regionów ciała pacjenta.

Przy stosowaniu tej metody leczenia ważne jest takie sterowanie aparaturą aplikującą pole magnetyczne do wnętrza ciała pacjenta, żeby uzyskać maksymalną koncentrację tego pola na chorym narządzie, a minimalnie rozpraszać je na inne, niewymagające tego części ciała. Dla realizacji tego zadania bardzo przydatna jest symulacja komputerowa. Na rysunku 17 pokazano działanie programu modelującego wnikanie pola magnetycznego do wnętrza kończyny w przypadku wspomagania polem magnetycznym procesu zrastania się kości przy złamaniu w połowie jej długości.



Rysunek 17. Komputerowe modelowanie oddziaływania pola magnetycznego i chorego narządu (złamanej kończyny)

Źródło: rysunek pochodzi z artykułu autora [1].

Dzięki takiej symulacji komputerowej można optymalnie dobrać kształt aplikatora pola magnetycznego, jego rozmiar oraz położenie, a także poprawnie dobrać sposób jego sterowania. Komputer jest w tym przypadku koniecznym składnikiem procesu leczenia.

7. Telemedycyna

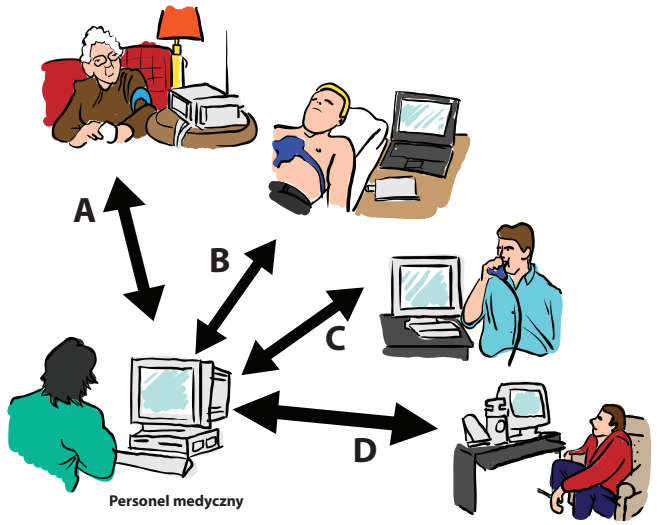
Najbardziej zaawansowanym obszarem zastosowań informatyki w medycynie jest tak zwana telemedycyna. Istota telemedycyny polega na tym, że pewne formy usług medycznych świadczone są nie na zasadzie bezpośredniego kontaktu lekarza z pacjentem, ale są – jak to się czasem brzydko mówi – zapośredniczone przez narzędzia teleinformatyki. Lekarz ma do czynienia z informacją

o pacjencie pozyskiwaną z wykorzystaniem środków technicznych (głównie Internetu), a pacjent jest badany, monitorowany i ewentualnie także konsultowany i instruowany w sprawach związanych z profilaktyką i terapią z użyciem tych samych środków technicznych działających w drugą stronę.

Pierwsze systemy, które dziś byśmy zaliczyli do telemedycyny, powstały w latach 60. XX wieku – początkowo na potrzeby pionierskich w tamtych czasach lotów kosmicznych. Na przykład podczas pierwszego amerykańskiego lotu orbitalnego (John Glen), zdalnie rejestrowano na Ziemi tętno, ciśnienie krwi, a także zapis EKG astronauty. Zapewne podobne badania prowadzili podczas swoich wypraw kosmicznych Rosjanie, ale utajniali wyniki, więc niewiele o nich wiadomo. Również w latach 60. powstała satelitarna sieć telekomunikacyjna o przeznaczeniu telemedycznym łącząca amerykańskie bazy wojskowe, rozrzucone na wszystkich kontynentach, ze specjalistycznymi ośrodkami medycznymi w USA. Taka sama sieć wspomaga obecnie polski kontyngent wojskowy w Afganistanie. Do cywilnych zastosowań technik telemedycznych wiele wniosła Australia, mająca w tym zakresie szczególnie duże potrzeby ze względu na rozproszenie stosunkowo nielicznej ludności na bardzo dużym terenie.

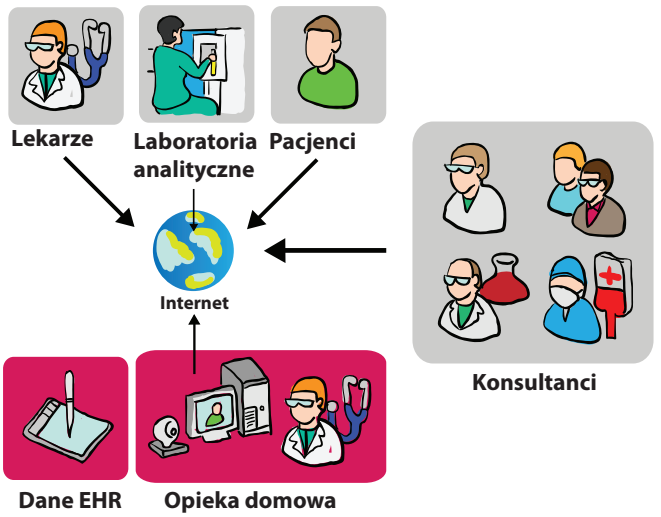
Telemedycyna jest przydatna w kontekście możliwości objęcia opieką medyczną pacjentów do których trudno dotrzeć z tradycyjnymi formami medycznych usług, na przykład mieszkańców małych wiosek oddalonych od szpitali i ośrodków zdrowia, marynarzy statków znajdujących się na morzu, uczestników egzotycznych wypraw, żołnierzy pełniących służbę w zagrożonych miejscach, a także specjalnych pacjentów do których osobisty dostęp jest utrudniony, na przykład więźniów z wieloletnimi wyrokami, którzy bywają niebezpieczni dla personelu medycznego, a których jednak także trzeba leczyć, czasem nawet wbrew ich woli. Telemedyczne metody mogą znaleźć również zastosowanie w leczeniu chorób zakaźnych, w przypadku których bezpośredni kontakt lekarza i pielęgniarki z osobą chorą rodzi niebezpieczeństwo dla nich samych oraz dla ich rodzin.

Telemedyczna pomoc użyteczna jest także w odniesieniu do osób po zabiegach operacyjnych i innych rekonwalescentów, którzy już nie muszą przebywać w szpitalu, ale powinni być nadal pod kontrolą lekarską, a także w odniesieniu do ludzi starszych i samotnych, których stan zdrowia można monitorować w sposób zdalny nie narażając ich na wysiłek i dyskomfort związany z koniecznością wizyt u lekarza. Ogromnie ważna jest rola opieki telemedycznej nad pacjentami chorymi na choroby przewlekłe – na przykład na chorobę niedokrwienną serca albo na cukrzycę. Dobrze przemyślane rozwiązania telemedyczne pozwalają im normalnie funkcjonować, ale bez ryzyka, że ich choroba wymknie się spod kontroli i stworzy zagrożenie.



Rysunek 18. W dobrze zorganizowanym systemie opieki telemedycznej niewielka liczba personelu medycznego może otoczyć opieką bardzo wielu pacjentów

Istotna zaleta wynikająca ze stosowania technik telemedycznych polega także na tym, że dzięki użyciu nowoczesnych technik informatycznych, pozwalających wstępnie analizować dane od pacjentów w sposób automatyczny z odsiewaniem informacji mało znaczących i nie wymagających osobistej interwencji lekarza – niewielka liczba pracowników personelu medycznego może otoczyć zdalną opieką bardzo wielu pacjentów (rys. 18).

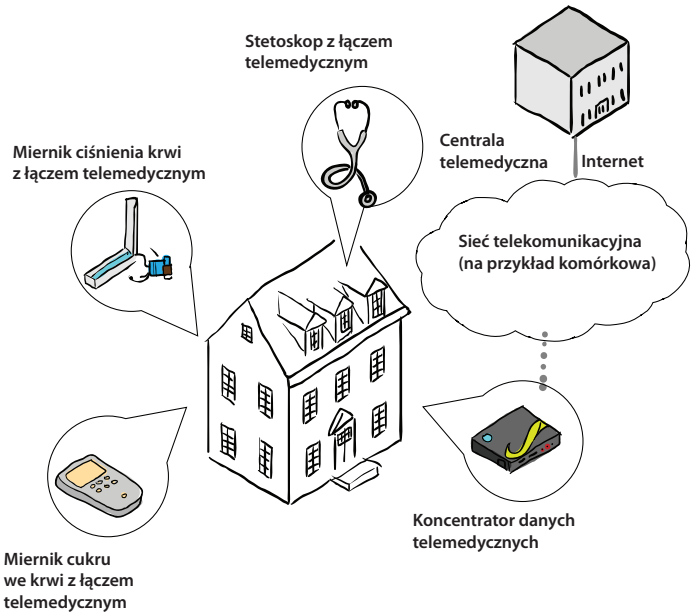


Rysunek 19. Podmioty uczestniczące w systemie telemedycyny

Schemat pokazany na rysunku 18 jest czytelny, ale nadmiernie uproszczony. W rzeczywistości w systemie telemedycznym istnieją także laboratoria analityczne, personel sprawujący opiekę domową oraz liczni konsultanci, z których rad mogą korzystać zarówno pacjenci, jak i opiekujący się nimi lekarze. Schemat powiązań łączący różne podmioty uczestniczące w systemie telemedycyny przedstawiono na rysunku 19. Na rysunku tym nie uwidoczniono ważnego faktu, mianowicie, że pacjentów korzystających z systemu może być bardzo wielu – w odróżnieniu od wszystkich pozostałych podmiotów. Dodatkowego komentarza wymaga pokazana na rysunku pozycja „Dane EHR". Otóż w krajach, w których został już wprowadzony elektroniczny rekord pacjenta (EHR to skrót od *Electronic Health Record*), jednym z ważnych zadań telemedycyny jest umożliwienie upoważnionym do tego jednostkom (szpitalom, lekarzom rodzinnym, ratownikom medycznym) zdalnego dostępu do danych z EHR obsługiwanego w danym momencie pacjenta, a także nanoszenie w tym rekordzie nowych informacji o przeprowadzonych badaniach i zastosowanym leczeniu – niezależnie od tego, gdzie te badania przeprowadzono i gdzie to leczenie zastosowano.

Świadomość zalet, jakie wiążą się z rejestrowaniem w formie zapisów komputerowych danych medycznych każdego pacjenta, pojawiła się zarówno wśród lekarzy, jak i wśród wspomagających ich pracę informatyków. Pierwsze takie elektroniczne kartoteki pojawiły się w większości szpitali pod koniec lat 60. minionego stulecia. Jako pionierską instytucję w tym zakresie wskazuje się powszechnie szpital El Camino (Kalifornia, USA), znany z tego, że jako pierwszy wdrożył szpitalny system komputerowy. Jednak pożytek z takich lokalnych szpitalnych baz danych był niewielki, bo nie obejmował opieki nad pacjentem poza macierzystym szpitalem. Dlatego tak ważne było wprowadzenie w roku 1968 przez Larry'ego Weeda koncepcji zapisów informacji o pacjentach w formie zunifikowanej (nazywanej wtedy ***Problem Oriented Medical Record***). **Pierwszą implementację tej koncepcji wdrożył w roku 1972 Regenstreif Institute. Decydujące znaczenie miało opublikowanie w roku 1991 przez Institute of Medicine (bardzo ceniony w USA ośrodek opiniotwórczy w dziedzinie medycyny) rekomendacji zalecającej wszystkim lekarzom stosowanie elektronicznej rejestracji diagnoz i zabiegów w personalizowanych rekordach pacjentów.**

System telemedyczny zawsze osadzony jest w jakimś konkretnym terenie i oczywiście zawsze zawiera komponentę związaną z pacjentami (odpowiednio wyposażone technicznie mieszkania i domy – rys. 20) – oraz część odbiorczą, za pomocą której personel medyczny odbiera i interpretuje nadchodzące od pacjentów sygnały, udzielając im pomocy stosownie do rzeczywistych potrzeb.



Rysunek 20. Przykładowe elementy telemedyczne znajdujące się w domu pacjenta

Na rysunku 21 przedstawiono bardzo mały system telemedyczny, w którym lokalna centrala przyjmuj i obsługuje sygnały pochodzące z niewielkiej liczby domów zlokalizowanych na pewnym ustalonym obszarze. Takie rozwiązanie może być zastosowane, gdy na przykład chcemy zapewnić opiekę telemedyczną mieszkańcom jakiegoś ośrodka czy osiedla. Być może w przyszłości ten model systemu telemedycznego wykorzystany zostanie w specjalnych osiedlach przeznaczonych dla seniorów – osób starszych i samotnych, które jednak nie godzą się na skoszarowane formy i warunki przebywania w typowych domach starców.

Rysunek 21. Przykładowy mały system telemedyczny dla niewielu pacjentów

8. Podsumowanie

Informatyka może być stosowana wszędzie, więc osoba, która wybierze sobie ten piękny i ciekawy zawód jako swoją przyszłą profesję może dosłownie przebierać w propozycjach, gdzie i co mogłaby robić. Jednak bezspornie zastosowania informatyki w medycynie mają liczne przewagi nad innymi działami aplikacji komputerów – i to przynajmniej z trzech powodów.

Po pierwsze osoby pracujące w informatyce medycznej przyczyniają się do tego, że lekarze wyposażeni w stworzone przez nich nowe algorytmy i programy mogą skuteczniej nieść pomoc osobom cierpiącym oraz zagrożonym kalectwem lub śmiercią. W ten sposób informatyk może swoją pracą przyczynić się do odzyskania zdrowia przez wielu pacjentów, a więc ma prawo twierdzić, że robi coś bardzo potrzebnego i szlachetnego.

Po drugie medycyna stawia przed informatyką wiele niezwykle ciekawych nowych wyzwań, których rozwiązywanie jest wspaniałą przygodą intelektualną, wzbogacającą zarówno profesjonalny warsztat informatyka, jaki i jego ogólną wiedzę – między innymi o tajemnicach zdrowia i choroby, o budowie ludzkiego ciała i funkcjonowaniu poszczególnych narządów i całych ich systemów.

Po trzecie poświęcając się pracy na rzecz informatyki medycznej możemy być pewni, że zapotrzebowanie na naszą wiedzę i pracę nigdy się nie wyczerpie. Inne dziedziny zastosowań informatyki mają swoje pułapy, których osiągnięcie zamyka popyt na kolejne opracowania i kolejne zlecenia. Na przykład komputerowy system bankowy może być doskonalony tylko do pewnego pułapu wynikającego z zapotrzebowania klientów. Gdy już wszystkie typowe i nietypowe potrzeby klientów zostaną zaspokojone – dalsze rozbudowy i modyfikacje oprogramowania znikna. Natomiast w medycynie nie ma takiego progu. Jakkolwiek doskonałe metody leczenia byśmy nie zastosowali – ludzie zawsze będą chcieli więcej i lepiej. Doskonalenie medycyny jest więc procesem, który się nigdy nie skończy, a ulepszająca się medycyna stwarza i będzie stwarzała rosnące zapotrzebowanie na usługi informatyki medycznej.

Warto więc być pionierem nowych zastosowań komputerów w medycynie i dlatego warto studiować tę dziedzinę informatycznej wiedzy. Wierzę w to, że się przy niej spotkamy!

Literatura

1. Cieśla A., Kraszewski W., Tadeusiewicz R., *Magnetic teletherapy: part 2. Software – visualization of magnetic field in magnetotherapy*, „Electrical Review", accepted for printing in 2012
2. Haux R., *Medical informatics: Past, present, future*, „International journal of medical informatics" (79) 2010, 599-610
3. November J., *Biomedical Computing: Digitizing Life in the United States*, Johns Hopkins University Press, Baltimore 2012
4. Podsędkowski L., *Roboty medyczne – budowa i zastosowanie*,WNT, Warszawa 2010
5. Richesson R.L., Andrews J.E., *Clinical research informatics*, Springer, Berlin, 2012
6. Robson B., Baek O.K., *The engines of Hippocrates: From the Dawn of Medicine to Medical and Pharmaceutical Informatics*, John Wiley & Sons, Hoboken 2009
7. Roterman I. (red.), *Elementy informatyki medycznej – 1*, Wydawnictwo Uniwersytetu Jagiellońskiego, Kraków 2011
8. Rudowski R., *Informatyka medyczna*, WN PWN, Warszawa 2003
9. Tadeusiewicz R. (red.), *Inżynieria Biomedyczna – Księga współczesnej wiedzy tajemnej w wersji przystępnej i przyjemnej*, UWND AGH, Kraków 2008
10. Tadeusiewicz R., Śmietański J., *Pozyskiwanie obrazów medycznych oraz ich przetwarzanie, analiza, automatyczne rozpoznawanie i diagnostyczna interpretacja*, Wydawnictwo STN, Kraków 2011
11. Tadeusiewicz R., *Informatyka medyczna*, Skrypt uczelniany UMCS, Lublin 2011 (dostępny w całości na stronie internetowej http://informatyka.umcs.lublin.pl/files/tadeusiewicz.pdf)
12. Tadeusiewicz R., *Jak informatyka pomaga zajrzeć do wnętrza ludzkiego ciała*, Warszawska Wyższa Szkoła Informatyki, Warszawa 2010

Prof. zw. dr hab. inż. Ryszard Tadeusiewicz

informatyk, automatyk, biocybernetyk, prezes Krakowskiego Oddziału PAN, kierownik Katedry Automatyki i Inżynierii Biomedycznej AGH. Absolwent AGH, doktorat uzyskał w roku 1975, habilitację w roku 1980, tytuł naukowy – w roku 1986. W latach 1996-1998 pełnił funkcję prorektora ds. nauki AGH, od roku 1998 rektor AGH, wybrany na powtórną kadencję (1999-2002) oraz po raz trzeci z rzędu na kadencję na lata 2002-2005). W latach 1999-2005 – przewodniczący Konferencji Rektorów Polskich Uczelni Technicznych oraz wiceprzewodniczący Kolegium Rektorów Szkół Wyższych Krakowa. Członek prezydium KRASP w kadencji 1999-2002. Doktor Honoris Causa 12 uczelni krajowych i zagranicznych, członek PAN i PAU. Jest autorem ponad 950 prac naukowych i ponad 80 monografii książkowych.

rtad@agh.edu.pl
www.Tadeusiewicz.pl

Maciej M. Sysło

Wydział Matematyki i Informatyki
Uniwersytet Wrocławski
Uniwersytet Mikołaja Kopernika w Toruniu

Komputer – obiekt i narzędzie edukacji
Poznawcze walory informatyki i technologii informacyjno-komunikacyjnej

W tym rozdziale pochylamy się nad miejscem komputera w edukacji. Podobnie jak wiele innowacji, wynalazków i odkryć, komputer jest zarówno obiektem edukacji – uczy się o nim i o jego oprogramowaniu, jak i jest narzędziem w edukacji – służy jako technologia kształcenia, czyli wspomaga kształcenie w różnych dziedzinach. Celem rozważań jest przekonanie czytelnika, że komputer jest wyjątkową technologią wśród technologii kształcenia – w coraz większym stopniu zlewają się bowiem te dwie role komputera w edukacji, gdyż posługiwanie się nim jako narzędziem wymaga głębszego zapoznania się z jego działaniem i możliwościami, a nierzadko musimy umieć go programować.

Pierwsza część artykułu jest opisem krótkiej historii komputerów w edukacji, w drugiej natomiast przedstawiamy obecny stan edukacji informatycznej i kształcenia informatycznego w szkołach, na końcu kreślimy wizję niedalekiej przyszłości komputerów, a ogólniej – technologii w edukacji. Rozważania są umieszczone w szerszym, społecznym kontekście edukacji, rozumianej jako ogół działań i procesów, których celem jest kształtowanie postaw, wiedzy i umiejętności.

Ten rozdział jest również adresowany do uczniów. Jednym z najważniejszych priorytetów współczesnych systemów edukacji jest personalizacja, czyli dostosowanie kształcenia do indywidualnych zainteresowań, możliwości i potrzeb uczniów. Powinno więc być oczywiste dla każdego, w tym także dla Was – uczniów, że tego celu nie można osiągnąć bez Waszego udziału, bez współpracy wszystkich aktorów w teatrze szkoły, wśród których stanowicie najważniejszą grupę – beneficjentów kształcenia. Szerzej na temat personalizacji kształcenia piszemy w artykule [19].

1. Wprowadzenie

Od chwili pojawienia się komputerów panuje silne przekonanie, nie tylko wśród osób zajmujących się tą dziedziną, ale w oczach niemal całego społeczeństwa, że informatyka to klucz do dobrobytu[1]. Jednym z celów tego rozdziału jest przekonanie Was – uczniów, 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. Chcemy więc Was zainteresować informatyką i jej zastosowaniami już w szkole, jako dziedziną przyszłych studiów i kariery zawodowej. Więcej na ten temat piszemy w [17].

Wokół informatyki narosło 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. Uważa się jednak coraz powszechniej, że każdy człowiek posługujący się komputerem powinien w jakimś zakresie znać głębiej jego działanie, a zwłaszcza sposoby jego wykorzystania w różnych sytuacjach i do rozwiązywania różnych problemów.

Powszechnie znane są nazwiska osób ze świata informatyki i jej zastosowań, które znajdują się na wysokich pozycjach na listach najbogatszych osób na świecie. Chcemy Was przekonać, że reprezentują oni najwyższą **klasę** informatyków, a Ci najbogatsi dodatkowo mają z tego olbrzymią **kasę**. Źródeł sukcesów jednych i drugich można się doszukiwać głównie w ich osiągnięciach na polu informatyki (osiągnięcia wybranych informatyków przedstawiamy w [17]).

2. Innowacje w edukacji w przeszłości i pierwsze obawy

Wynalazki, innowacje, odkrycia, zwłaszcza te, które szybko zdobywały sobie powszechne uznanie, w naturalny sposób pojawiały się w edukacji. Tak było kiedyś z pismem, później z drukiem, a w ostatnim stuleciu – z radiem, filmem, telewizją i wreszcie z komputerami. Każdy nowy wynalazek wywołuje zachwyt i entuzjazm, z czasem jednak rodzą się wątpliwości i obawy, czy aby nie zburzy on zastanego dobrego świata ugruntowanych idei i wartości, wypracowanych i pielęgnowanych przez wieki.

[1] Nawiązujemy tutaj do tytułu książki Andrzeja Tarkowskiego *Informatyka to klucz do dobrobytu* z 1971 roku.

2.1. Innowacje w edukacji

Terminologia. W tym rozdziale odkrycia raczej nie pojawiają się, natomiast **wynalazki** i **innowacje**, to nowe, lepsze lub bardziej efektywne wytwory, procesy lub technologie powszechnie dostępne. **Komputer** reprezentuje tutaj urządzenie (jak laptop, smartfon, tablet) o funkcjach komputera osobistego, a **technologia** oznacza technologię informacyjno-komunikacyjną, na którą składają się środki (czyli urządzenia) i narzędzia (czyli oprogramowanie, w tym także oprogramowanie edukacyjne), jak również inne technologie (jak telekomunikacja), które służą wszechstronnemu posługiwaniu się informacją. **Edukacja** zaś jest rozumiana jako ogół działań i procesów, których celem jest kształtowanie postaw (wychowanie), wiedzy i umiejętności. Chociaż główną uwagę skupiamy na edukacji szkolnej (formalnej), obecnie nierozerwalnie są z nią związane: edukacja nieoficjalna (np. prowadzona na kursach, w klubach, stowarzyszeniach), edukacja nieformalna (kształcenie przez całe życie poza zorganizowanymi formami) i edukacja incydentalna (zachodząca w codziennych sytuacjach, które mogą być źródłem doświadczeń i wiedzy). **Kształcenie** to synonim edukacji.

Pojawienie się wynalazku lub innowacji zwykle wywołuje niepokój potencjalnymi zagrożeniami dla człowieka i holistycznych efektów jego rozwoju i kształcenia. W tym kontekście, tradycyjnie jest przywoływana legenda o Tamuzie, opisana przez Platona w jego *Fajdrosie*, patrz [10]. Pewnego razu Tamuz gościł Teuta, który wynalazł m.in. liczby, rachunki, geometrię, astronomię i pismo. Przekonywał on Tamuza, że wszyscy Egipcjanie powinni poznać i stosować pismo. Na to usłyszał od Tamuza: *Ten wynalazek niepamięć w duszach ludzkich posieje... to tylko środek na przypominanie sobie. Uczniom swoim dasz tylko pozór mądrości, a nie mądrość prawdziwą.*

Określenie epoka Gutenberga wprowadził wybitny kanadyjski teoretyk masowej komunikacji i środków przekazu, Marshall McLuhan (1911-1980). W latach 60. XX wieku, gdy rozpoczynała się ekspansja mediów elektronicznych, zapowiedział on powstanie globalnej wioski. Jest on również autorem poglądu, że medium jest przekazem (ang. *the medium is the message*), czyli środek przekazu ma przynajmniej taki wpływ na odbiorcę, jak sama wiadomość, lub inaczej – środek komunikacji determinuje cel przekazu. W dużym stopniu odnosi się to dzisiaj do komputerów, jako medium komunikacyjnego, nie należy jednak zapominać, że komputer przestał już być „obojętnym" środkiem przekazu, a stał się integralną częścią dziedzin, w których jest wykorzystywany. To kolejny argument na korzyść edukacyjnego znaczenia komputerów. Polecamy lekturę podstawowego dzieła McLuhana [6].

Po upływie wieków wiemy, że Tamuz mylił się, **pismo** przyniosło bowiem ludzkości wiele pożytków i jego rola w czasach elektronicznej informacji nawet wzrosła, do czego przyczyniła się ekspansja Internetu.

Wynalazek **druku** (faktycznie maszyny drukarskiej) w XV wieku przez Johannesa Gutenberga zapoczątkował erę demokratyzacji edukacji. Dzięki temu, że wiedza mogła być „drukowana" i powielana bez ograniczeń, stała się dostępna dla każdego. Do końca XX wieku żyliśmy w **epoce Gutenberga**, dominacji informacji drukowanej, aż nadciągnęła era, w której informacja stała się dostępna w postaci elektronicznej, możliwy jest więc do niej dostęp bez pośrednictwa papieru.

Wcześniej, przed Internetem, pojawiło się radio i natychmiast stało się ono medium wspierającym przekaz edukacyjny. Dzisiaj radio, a ogólniej przekaz dźwiękowy (np. audioksiążki), jest medium, któremu w edukacji stawia się najmniej zarzutów.

Nieco później, ale jeszcze przed erą Internetu, zaczęły rozpowszechniać się film i telewizja. Rozpoczęła się **era obrazu**. W miejsce wytworów Gutenberga, logicznie uporządkowanego przekazu w postaci drukowanej, również tekstu z ilustracjami, pojawił się świat telewizji z naciskiem na obraz, i trwa do dzisiaj, zdominowany przez narrację z dźwiękiem i szybką reakcję osób oglądających. **Dzieci obrazu** nie potrafią dłużej skupić myśli i uformować jej w logiczną całość.

O wspomnianych w tym, jak i w innych rozdziałach aspektach historycznych, związanych z ekspansją komputerów, należy cały czas pamiętać, gdyż ta historia nadal się toczy. Media elektroniczne nie wyparły i nadal konkurują z tradycyjnymi środkami przekazu, które nie zniknęły z naszego życia i ze szkoły, takimi jak: pismo, książka, radio (audioprzekaz), telewizja. Raczej można mówić o pewnej **konwergencji mediów** – wszystkie one zlewają się w jedno medium, dla którego nośnikiem przekazu jest sieć Internet, a odbiornikiem i nadajnikiem to samo urządzenie, komputer lub podobne funkcjonalnie mu urządzenia. Przyglądamy się tutaj, jak rozwijała się rola komputerów w edukacji, gdyż niemal wszystkie elementy, jakie pojawiły się na przestrzeni niedługiego czasu są nadal obecne w kształceniu, często unowocześnione dzięki nowocześniejszej technologii, a niektóre pozostają w edukacji mimo nie najlepszej ich oceny dydaktycznej i metodycznej oraz efektów kształcenia. Historia, o której tutaj piszemy, nie tylko odcisnęła się piętnem na obecnym stanie, ale wiele jej elementów nadal ma wpływ na to, co i jak dzieje się dzisiaj w szkołach. Pamiętajmy więc:

Aby drogę poznać przyszłą
trzebać pomnieć, skąd się przyszło.

Cyprian Kamil Norwid

2.2. Obawy związane z rozwojem technologii komputerowej

Komputer uznaje się za **technologię definiującą** koniec XX wieku [1], gdyż, jak żadna inna technologia, wpływa na rozwój techniki, nauki i edukacji, zmienia zawody i przyczynia się do powstawania nowych, ma wpływ na stosunki i życie społeczne, pobudza również wyobraźnię. Można odnieść wrażenie, że komputery potrafią niemal wszystko. Poważnym zagrożeniem może być jednak przyjęcie założenia, że oto pojawiła się technologia, która jest tak potężna, że należy podporządkować jej edukację. Na przykład od czasu do czasu odżywają pomysły, by nauczyciela zastąpić komputerem. Ten niebezpieczny stan zaprzedania umysłu i duszy technologii określa się mianem technopolu, patrz [10]. **Technopol** jest stanem kultury i stanem umysłu, poszerzeniem technokracji w kierunku opanowania człowieka i społeczeństwa przez technologie. Technokracje zajmują się tworzeniem maszyn i jest dla nich oczywiste, że maszyny zmieniają życie ludzkie. Ale technokracje w swojej filozofii nie zakładają, że sens ludzkiego życia odnajduje się tylko w maszynach i technice. W technopolu natomiast dochodzi do niebezpiecznego redukcjonizmu przyjmującego, że społeczeństwu najlepiej służy oddanie ludzi do dyspozycji maszyn, techniki i technologii, a życie ludzkie w technopolu jest nie więcej warte dla społeczeństwa niż jego maszyny.

Człowiek wiążący się coraz mocniej z komputerem niepokojąco zaczyna wszystko traktować jako dane. Szkoła i edukacja powinny przeciwdziałać kreowaniu **człowieka Turinga** (patrz [1]) – istoty będącej w gruncie rzeczy jedynie procesorem informacji, traktującej swoje otoczenie jako informacje do przetworzenia. Z drugiej jednak strony nie można przeoczyć potencjalnych szans tkwiących w technologii komputerowej i technologii informacyjnej, na co podatne mogą być systemy edukacyjne, z reguły bardzo zachowawcze. Należy więc chronić edukację przed popadnięciem w technopol, jak i przed całkowitym odżegnaniem się od technologii – na tym są skupione rozważania i propozycje zamieszczone w tym rozdziale.

Warto jeszcze zwrócić uwagę, że obecnie w erze informacji każda dziedzina w zawrotnym tempie obrasta w nowe informacje. Na początku tej ery, Adam Schaff, filozof nauki, przewidywał w raporcie Klubu Rzymskiego (patrz [3]) powstanie nowej klasy, **klasy posiadaczy informacji**. Byli i są posiadacze środków produkcji, kapitału i władzy – spodziewano się nadejścia ery posiadaczy informacji. Słychać już było demagogiczne deklaracje: **kto ma informacje, ten ma władzę**, będące parafrazą powiedzenia „kto ma władzę, ten ma religię". Rozrost sieci Internet i rozwój jej usług zdezaktualizował te przewidywania i obawy. Nie można mieć bowiem władzy nad powszechnie panującą „religią" – informacją, którą cechują przede wszystkim demokratyczne, równe prawa dostępu, np. za pośrednictwem tejże sieci Internet.

3. Początki edukacji informatycznej

Edukacja informatyczna obejmuje dwa rodzaje zajęć:

- **wydzielone zajęcia** (przedmioty) **informatyczne** – te zajęcia składają się na **kształcenie informatyczne** uczniów, czyli na kształcenie w zakresie informatyki, piszemy o tym również w [17];
- **wspomaganie komputerami** i innymi technologiami zajęć z pozostałych dziedzin (przedmiotów) nieinformatycznych.

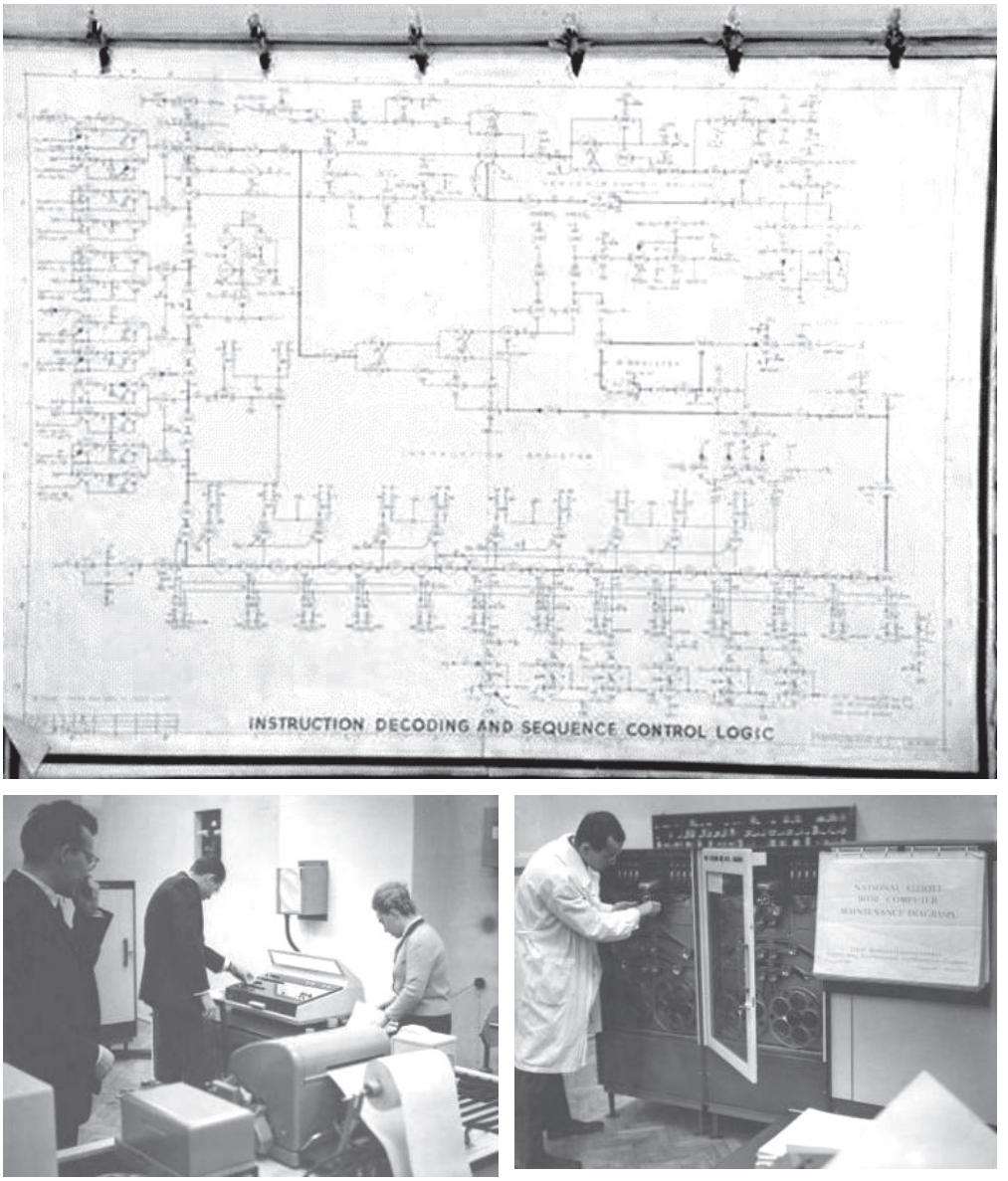
Aż trudno uwierzyć, ale pierwsze regularne zajęcia z komputerami w polskiej szkole miały miejsce w połowie lat 60. XX wieku – był to przedmiot programowanie i obsługa maszyn cyfrowych, prowadzony w III LO we Wrocławiu. Zajęcia odbywały się w klasie – uczniowie pisali programy na tablicy i w zeszytach – następnie programy te były przepisywane na taśmę perforowaną i uruchamiane w obecności uczniów na komputerze Elliott 803 (patrz rys. 1) w Katedrze Metod Numerycznych Uniwersytetu Wrocławskiego. Programy służyły do wykonywania obliczeń matematycznych, w tamtych czasach bowiem komputery, zwane **maszynami matematycznymi**, służyły głównie do wykonywania obliczeń matematycznych i inżynierskich. Kilka osób spośród uczniów tamtych zajęć pracowało przez długie lata na Uniwersytecie Wrocławskim, a dwie z nich pracują do dzisiaj (2012).

Przez niemal dwadzieścia następnych lat wykorzystanie komputerów w edukacji niewiele się zmieniło, jedynie przybywało dużych maszyn w różnych ośrodkach akademickich, w placówkach ZETO (Zakłady Elektronicznej Techniki Obliczeniowej) i w innych instytucjach, które dość chętnie udostępniały je młodzieży ze szkół. Jednocześnie w uczelniach i na specjalistycznych kursach przygotowywano nauczycieli do prowadzenia w szkołach zajęć z wykorzystaniem komputerów. Więcej szczegółów na temat wczesnej historii komputerów w edukacji w Polsce można znaleźć w artykule [12].

W dużych i potężnych, jak na tamte czasy, maszynach upatrywano narzędzia do realizacji popularnego wtedy **nauczania programowanego**[2], którego ideą było „programowanie ucznia" (niekoniecznie za pomocą komputera), polegające na niemal automatycznym prowadzeniu go przez kolejne etapy zdobywania wiedzy. Do tego znakomicie nadawał się komputer. Programowanie ucznia może stwarzać wrażenie indywidualizacji, gdyż każdego ucznia można inaczej, w odpowiednim dla niego tempie, programować. Jednak jest to nadal kierowanie uczniem przez komputer, w sposób, w jaki zaprogramowana została maszyna, a uczeń może poruszać się jedynie w ramach opcji przewidzianych w programie i nie może

[2] Wyobrażano sobie na przykład w Stanach Zjednoczonych, że za wieloma terminalami dużego i potężnego komputera wyposażonego w program uczący będzie można posadzić uczniów i… zwolnić dużą część nauczycieli.

podążać w pełni własną drogą. Wykorzystanie komputerów w nauczaniu programowanym było jednak w pewnym sensie unowocześnieniem tego sposobu kształcenia. Ta chęć programowania ucznia jest nadal bardzo popularna, a nowe aplikacje informatyczne tylko uatrakcyjniają to podejście.



Rysunek 1. Komputer Elliott 803: jego schemat logiczny i programiści w czasie jego użytkowania

Źródło: archiwum autora oraz archiwum Katedry Metod Numerycznych Uniwersytetu Wrocławskiego.

Opisane wzmocnienie nauczania programowanego komputerami znalazło swojego wielkiego oponenta dopiero pod koniec lat 70. XX wieku w osobie Seymoura Paperta, jednego z prekursorów wykorzystania komputerów w edukacji, który, przesiąknięty ideami konstruktywistycznymi, odwrócił relację i pisał w roku 1980 [8]: *Można by sądzić, że komputer jest wykorzystywany do programowania dziecka. W mojej wizji to dziecko programuje komputer.* Papert widział w programowaniu[3] sposób na porozumiewanie się człowieka z komputerem w języku, który rozumieją obie strony. Stworzył w tym celu język Logo. Przedstawił także ideę uczenia się matematyki w Matlandii, *czyli w warunkach, które są dla uczenia się matematyki tym, czym mieszkanie we Francji jest dla uczenia się języka francuskiego*. Papert wyprzedził swoją epokę ideami, które mają szansę być zrealizowane dopiero w warunkach sieci Web 2.0, gdy uczeń może być współtwórcą treści i środowiska kształcenia.

Papert nie uniknął jednak błędu. Tworząc wspaniałą wizję zajęć wspomaganych komputerem, był przekonany, że komputery plus język Logo nieuchronnie wzbogacą edukację. Po dekadzie oczekiwań na rezultaty, w kolejnej swojej książce [9] był rozczarowany, że jego pomysły nie rozlały się powszechnie po szkołach. Później bił na alarm, że szkoły nie stanowią dla uczniów tak obiecywanego, głównie przez polityków, pomostu do społeczeństwa informacyjnego i z wielkim oporem przyjmują jego idee, stosując komputery podobnie do: *prób udoskonalenia transportu w XIX wieku poprzez przymocowanie silników odrzutowych do drewnianych wozów*. Zwracał on również uwagę na inny powód braku sukcesów: *stosowanie komputerowego wsparcia jako nowego sposobu nauczania według starych programów*. Ta opinia Paperta pozostaje nadal aktualna – szkoły, nauczyciele i całe systemy edukacji bardzo powoli zmieniają programy nauczania i warunki, w jakich przebiega kształcenie w szkołach, a także poza zajęciami szkolnymi.

4. Informatyka a technologia informacyjna

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, które w ostatnich latach przechodzą głęboką ewolucję. Można przyjąć, że **informatyka** jest dziedziną, która zajmuje się projektowaniem, realizacją, ocenianiem, zastosowaniami i utrzymaniem systemów przetwarzania informacji z uwzględnieniem przy tym aspektów sprzętowych, programowych, organizacyjnych i ludzkich wraz z implikacjami przemysłowymi, handlowymi, publicznymi, politycznymi i społecznymi. Wspomniane systemy przetwarzania informacji na ogół bazują

[3] Programowanie jest tutaj rozumiane jako umiejętność komunikacji z komputerem.

na rozwiązaniach komputerowych, a w ogólności – mikroprocesorowych (jak telefony komórkowe). Z kolei informacje mogą mieć najróżniejszą postać. Na początku były to tylko liczby, ale z czasem stało się możliwe przetwarzanie tekstów, a później również grafiki, dźwięków i filmów.

Termin informatyka pojawił się w języku polskim jako odpowiednik terminu angielskiego *computer science*, a brzmi podobnie, jak jego odpowiedniki w języku francuskim i niemieckim.

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. *Information and Communication Technology* – ICT), która swoim zakresem znacznie wykracza poza tradycyjnie rozumianą informatykę. Przyjmuje się, że **technologia informacyjno-komunikacyjna (TIK**, w skrócie **technologia)** 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 połączeniem zastosowań informatyki z wieloma innymi technologiami pokrewnymi.

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***[4], 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):

- *Computer Engineering* – budowa i konstrukcja sprzętu komputerowego;
- *Information Systems* – tworzenie systemów informacyjnych;

[4] *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 przekładem tego terminu na język polski posługują się w swoich publikacjach m.in. ks. Józef Kloch i Andrzej Walat.

- *Information Technology* – technologia informacyjna, zastosowania informatyki w różnych dziedzinach;
- *Software Engineering* – produkcja oprogramowania;
- *Computer Science* – studia podstawowe, uniwersyteckie studia informatyczne.

Poza komputyką, tradycyjnie rozumianą jako dziedzina uniwersytecka i politechniczna, istnieje jeszcze wiele innych kierunków kształcenia związanych z informatyką i jej zastosowaniami, jak na przykład informatyka medyczna (na akademiach i w szkołach medycznych), ekonometria (na uczelniach ekonomicznych), bioinformatyka i wiele innych.

Mając obecnie na uwadze przyszłą karierą zawodową uczniów, należy uwzględnić poszerzającą się gamę zawodów określanych mianem ***IT Profession***, czyli zawodów związanych z profesjonalnym wykorzystywaniem zastosowań informatyki i technologii informacyjno-komunikacyjnych. Pracownicy tych zawodów albo są informatykami z wykształcenia, albo najczęściej nie kończyli studiów informatycznych, jednak muszą kompetentnie posługiwać się narzędziami informatycznymi. Są nimi na przykład specjaliści z zakresu bioinformatyki, informatyki medycznej, telekomunikacji, genetyki – 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 informatyki, i ta lista stale się powiększa.

5. Komputery osobiste w edukacji – początki powszechnej edukacji informatycznej

Pojawienie się komputerów osobistych stworzyło nadzieję, że odegrają one przynajmniej podwójną rolę, zwiększą możliwości indywidualizacji kształcenia i poza wydzielonymi zajęciami informatycznymi znajdą również swoje miejsce na zajęciach innych przedmiotów. Po prawie ćwierćwieczu od ich pojawienia się jest jeszcze daleko od zadowalającej realizacji tych dwóch celów. Co więcej, pojawiło się zagrożenie, że realizacja tych zamierzeń zepchnie na plan dalszy solidne przygotowanie informatyczne uczniów (patrz p. 5.3).

5.1. Model rozwoju edukacji pod wpływem technologii

Planując wykorzystanie kolejnej nowej technologii w edukacji, warto pamiętać, że wdrożenie każdej technologii rządzi się pewnymi ogólnymi prawidłowościami, które dość wcześnie zidentyfikowano i szczegółowo opisano (patrz artykuł *Model rozwoju technologii informacyjnej w edukacji* w zbiorze [12]). By je wyjaśnić, nie-

zbędne jest przyjęcie **modelu zmian** zachodzących lub co najmniej oczekiwanych w związku z integrowaniem nowej technologii z kształceniem, a w szczególności z rozwijaniem kompetencji uczniów, doskonaleniem nauczycieli i rozwojem szkoły. Proponowany model składa się z czterech etapów. Najpierw sama technologia jest przedmiotem zainteresowań uczniów i nauczycieli oraz zajęć, następnie jako technologia kształcenia pojawia się w różnych miejscach procesu kształcenia, zastępując stare narzędzia i metody lub tylko integrując się z nimi, by wreszcie na trzecim etapie rzeczywiście zintegrować się z procesem kształcenia. Pełne zaś wykorzystanie technologii następuje w ostatniej fazie transformacji szkoły i systemu edukacji ku „nowej szkole". Tym etapom rozwoju szkoły odpowiada rozwój kompetencji informatycznych zarówno nauczycieli, jak i uczniów: na początku interesują się oni technologią, później podejmują próby jej wykorzystania w różnych dziedzinach, by wreszcie posługiwać się nią w sposób zintegrowany. Odniesienie się do modelu rozwoju technologii w edukacji pozwala dostrzec niedopatrzenie we wczesnej propozycji Paperta – dodanie lub tylko postawienie komputerów obok tego, co robią uczniowie i nauczyciele samo niestety nie wystarcza do wniesienia znaczących efektów, dopiero ich **zintegrowanie** w procesie nauczania i jego organizacji (etap trzeci w modelu) stwarza taką szansę. Pod warunkiem jednak, że jednocześnie ulegnie zmianie relacja między uczącymi się i nauczycielem – nauczyciel zejdzie z piedestału nieomylnego „dostawcy" informacji i wiedzy i stanie się doradcą ucznia w jego własnym, zindywidualizowanym rozwoju.

Ten wzorzec zmian należy traktować elastycznie, odpowiednio do rodzaju technologii oraz przygotowania uczniów i nauczycieli. Obecnie, gdy nowa technologia trafia do szkół, często wcześniej trafia do rąk uczniów poza szkołą, a więc pierwszy etap związany z jej poznaniem można często pominąć. Tak jest na przykład z laptopami, telefonami komórkowymi, smartfonami, tabletami. Ale już tablice interaktywne, czy też systemy odpowiedzi (ang. *voting systems*) są na ogół nowością zarówno dla uczniów, jak i dla nauczycieli, i wymagają uwzględnienia pierwszego etapu, czyli zapoznania się z tymi urządzeniami.

Znajomość tego modelu i uwzględnienie go w rozwoju technologii w edukacji pozwala lepiej zaplanować działania w szkole i często uniknąć rozczarowań oraz zaoszczędzić środków, gdy spodziewane efekty wdrożenia technologii nie są widoczne. Na ogół wynika to z pominięcia lub niedocenienia w pełni któregoś z etapów.

Z umiarkowanym optymizmem i właściwą rezerwą należy podchodzić zarówno do kolejnych technologii, które pojawiają się na rynku i w rękach uczniów, jaki i są oferowane nauczycielom i szkołom. Na początku ekspansji komputerów w edukacji przewidywano, że komputery wspomogą edukację zwiększając korzyści edukacyjne uczniów. Nie do końca się to spełniło. Nowa techno-

logia, zanim nie uzyska **edukacyjnego wsparcia**, czyli nie zostanie przygotowana wraz z całym środowiskiem jej wykorzystania przez uczniów i przez nauczycieli i nie będzie przynosić sprawdzonych korzyści edukacyjnych w postaci zwiększonych osiągnięć uczniów, pozostawać będzie propozycją pozaedukacyjną. Szkoły nie są poletkami doświadczalnymi dla nowych technologii – kolejne rozwiązania technologiczne powinny więc być zweryfikowane zanim masowo trafią do rąk uczniów jako wsparcie ich kształcenia. Z drugiej jednak strony, jak już pisaliśmy, szkoła nie powinna się zamykać przed nowymi technologiami, zwłaszcza tymi, które masowo pojawiają się w rękach uczniów (patrz p. 6).

5.2. Pojawienie się komputerów osobistych w szkole

W połowie roku 1981 odbyła się premiera mikrokomputera IBM PC, którego dominacja na rynku komputerów osobistych trwa do dzisiaj, chociaż wcześniej i później pojawiło się wiele innych rozwiązań, m.in. mikrokomputery ZX Spectrum i Elwro 800 Junior (patrz rys. 2), które w większych ilościach zaczęły trafiać do polskich szkół. Mikrokomputery zapoczątkowały w połowie lat 80. XX wieku rzeczywiście powszechną edukację informatyczną, z szansami na pełne uwzględnienie indywidualizacji kształcenia.

Rysunek 2. Mikrokomputery ZX Spectrum i Elwro 800 Junior

Źródło: archiwum autora.

Na początku ery mikrokomputerowej edukacja informatyczna nadal w dużym stopniu ograniczała się do wydzielonych zajęć informatycznych – początkowo był to przedmiot **elementy informatyki** – z niewielkim tylko wykorzystaniem komputerów na przedmiotach matematycznych i przyrodniczych. Pierwszy program nauczania tego przedmiotu dla liceów został opracowany przez zespół Polskiego Towarzystwa Informatycznego (PTI) w roku 1985, a w roku 1990 został zatwierdzony program elementów informatyki dla ostatnich klas szkoły podstawowej. Na początku lat 90., zespół z Instytutu Informatyki Uniwersytetu Wrocławskiego kierowany przez autora niniejszego tekstu przedstawił modułowy pro-

gram nauczania elementów informatyki, który mógł być dostosowany do różnych warunków nauczania tego przedmiotu w szkołach. Inny zespół kierowany przez autora opublikował pierwszy podręcznik do elementów informatyki, uzupełniony w roku 1997 przewodnikiem dla nauczycieli. Podręcznik był uniwersalny, gdyż mało zależał od konkretnego oprogramowania – miał aż 9 wydań i do dzisiaj korzysta się z niego na niektórych zajęciach. Poza budową komputerów, opisem systemu operacyjnego i programowaniem (w języku Pascal, do dzisiaj uczonym w szkołach), znalazły się w nim rozdziały omawiające komputerowe wspomaganie edycji tekstów i obliczenia prowadzone w arkuszu kalkulacyjnym. Na początku lat 90. minionego wieku, zespół kilkudziesięciu informatyków z kilku uczelni, kierowany również przez autora, opracował i dostarczył do szkół oprogramowanie edukacyjne – **pakiet EI** – o którym z perspektywy czasu można śmiało powiedzieć, że „wyprzedziło epokę". Programy z tego pakietu, po przeniesieniu do środowiska Windows, są obecnie powszechnie dostępne w sieci (wiele z nich można znaleźć na stronie http://mmsyslo.pl/Materialy/Oprogramowanie).

5.3. Upadek kształcenia informatycznego

Wydzielone zajęcia informatyczne w polskich szkołach były bardzo poważnie traktowane w kolejnych reformach systemu oświaty i nigdy pod żadnym naciskiem nie dopuszczono do usunięcia ze szkół przedmiotu informatyka, 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. Teraz Amerykanie przywracają znaczenie kształcenia w zakresie informatyki, gdy okazało się, że sprowadzenie edukacji informatycznej do zajęć tylko z technologii informacyjno-komunikacyjnej, czyli wykorzystania narzędzi informatycznych, spowodowało spadek zainteresowania uczniów karierami informatycznymi aż o ponad 50% (podobnie jest w Wielkiej Brytanii). Malejące zainteresowanie studiami informatycznymi obserwuje się nie tylko w Stanach Zjednoczonych i w Wielkiej Brytanii, ale również w wielu innych krajach, także w Polsce.

Powodów tego stanu rzeczy jest wiele. Mnóstwo osób, w tym nauczyciele i rodzice, nie uważa informatyki za niezależną dziedzinę nauki, a zatem także za szkolny przedmiot. Znaczna część po prostu myli i utożsamia informatykę z technologią informacyjno-komunikacyjną i sprowadza edukację informatyczną do udostępniania uczniom i nauczycielom komputerów i Internetu w szkole i w domu. Nie odróżniają oni stosowania komputerów i sieci Internet od studiowania podstaw informatyki.

Jest też wiele powodów zmniejszonego zainteresowania samych uczniów informatyką, jako dziedziną kształcenia i przyszłą karierą zawodową. Początkowo informatyka była kojarzona z programowaniem komputerów, co wywo-

ływało silny sprzeciw, gdyż uważano, że niewielu uczniów zostanie kiedyś programistami. Na przełomie lat 80. i 90. XX wieku 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 wstąpieniem na uczelnię, najczęściej poza szkołą. Co więcej, dostępne oprogramowanie umożliwia tworzenie nawet bardzo złożonych aplikacji komputerowych bez wcześniejszego zaznajomienia się z: logiką, metodami programowania, matematyką dyskretną, metodami numerycznymi, 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 niewielka 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.

Aby zmienić tę sytuację, zajęcia informatyczne w szkołach powinny przygotowywać uczniów do dalszego kształcenia w kierunkach związanych z informatyką i technologią, zamiast utwierdzać ich w przekonaniu, że ukształtowane w tym zakresie wiedza i umiejętności, w szkole i poza szkołą, są wystarczające. Czasem uczniowie są niezadowoleni i zniechęceni sposobem, w jaki są prowadzone w szkole zajęcia informatyczne, i nie widzą przyszłości w głębszym poznawaniu tej dziedziny, nawet na potrzeby innych dziedzin, którymi są zainteresowani.

Poza zmniejszoną podatnością na zmiany, jeszcze inne czynniki powodują, że przed rozwojem edukacji informatycznej w szkołach piętrzy się wiele trudności, często obiektywnych. Wśród nich:

1. Brak nauczycieli przygotowanych do realizacji wydzielonych zajęć informatycznych. Ukończenie studium podyplomowego nie jest wystarczającym przygotowaniem. Niewielu absolwentów kierunków informatycznych podejmuje pracę w szkołach.
2. Metodyka kształcenia informatycznego nie nadąża za zmianami, nauczyciele incydentalnie posługują się metodologią rozwiązywania problemów z pomocą komputerów i myśleniem komputacyjnym (patrz p. 5.4), do wyjątków należy realizacja algorytmiki na informatyce w gimnazjum.
3. Problemy rozwiązywane na zajęciach informatycznych rzadko odwołują się do rzeczywistych zastosowań, z którymi uczniowie spotykają się na co dzień, nawet w zakresie wykorzystania komputerów i Internetu w innych przedmiotach.

4. Większość nauczycieli przedmiotów informatycznych nie jest przygotowanych do pracy z uczniami uzdolnionymi informatycznie.

W ostatnich latach w Polsce, podobnie jak w USA, podejmuje się wiele inicjatyw, których celem jest poprawa nakreślonej wyżej sytuacji w obszarze kształcenia informatycznego oraz w odniesieniu do innych deficytowych kierunków kształcenia (ścisłych i przyrodniczych). Inicjatywy te można podzielić na dwie grupy, w obu przypadkach są wspierane przez fundusze UE. Z jednej strony Ministerstwo Nauki i Szkolnictwa Wyższego zleca uczelniom prowadzenie studiów zamawianych na kierunkach deficytowych, a z drugiej strony – Ministerstwo Edukacji Narodowej i samorządy prowadzą projekty, których celem – pod hasłem: Człowiek – najlepsza inwestycja – jest wspieranie rozwoju wiedzy i umiejętności w dziedzinach deficytowych.

W tej drugiej grupie inicjatyw był realizowany projekt **Informatyka+** (http://www.informatykaplus.edu.pl/), w którym w latach 2009-2012 wzięło udział ponad 17 tys. uczniów ze szkół ponadgimnazjalnych z pięciu województw. Celem projektu było podwyższenie kompetencji uczniów ze 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 był alternatywną formą kształcenia pozalekcyjnego, miał również wpływ na podniesienie poziomu osiągnięć uczniów w szkołach, patrz [15].

Program Informatyka+ jest przykładem działań określanych mianem ***outreach***, polegających na tym, że uczelnia wyższa wraz ze swoimi pracownikami naukowo-dydaktycznymi prowadzi zajęcia z uczniami, które mają pogłębić ich wiedzę i umiejętności w tematycznych obszarach projektu i przygotować ich do podjęcia kształcenia na kierunkach reprezentowanych przez uczelnię.

Badania rynku zatrudnienia i jego potrzeb potwierdzają, że nadal są i 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.

5.4. Rozwój edukacji informatycznej

W rozwoju edukacji informatycznej w szkołach można wyróżnić kilka etapów, które pojawiły się zarówno w związku z ewolucją technologii komputerowej, jak i rozwojem metod kształcenia oraz metod wykorzystania komputerów w edukacji i poza nią.

W początkowym okresie (lata 80.-90. XX w.), celem wydzielonych zajęć informatycznych było kształtowanie **alfabetyzacji komputerowej** (ang. *computer*

literacy), czyli podstaw posługiwania się komputerem i jego oprogramowaniem, a w późniejszych latach – także korzystania z podstawowych usług sieci komputerowej (Internet). Uwaga uczniów i nauczycieli była skupiona głównie na tym, co oferowały komputery i ich oprogramowanie. Ale środowisko pracy z komputerem zmieniało się co kilka lat, czy w takim razie należało ponownie zapoznawać uczniów z nowymi rozwiązaniami? Takie podejście groziłoby, że w ciągu 12 lat pobytu w szkole uczeń musiałby kilka razy poznawać nowe środowisko pracy z komputerem. Ponadto, niedostateczne uwzględnienie zmian w technologii było źródłem braku zaufania do nabytych umiejętności i obawy, czy są one wystarczającym przygotowaniem na czekające wyzwania w przyszłości. Co więcej, skupienie się na aktualnej technologii było faktycznie niepełnym obrazem możliwości technologii w jej ciągłym rozwoju.

Na przełomie wieków XX i XXI nastąpiło więc poszerzenie alfabetyzacji do **biegłości komputerowej** (ang. *fluency in IT*), której celem stało się również przygotowanie uczniów na zmiany w technologii. Biegłość komputerowa, poza podstawami alfabetyzacji w zakresie aktualnych możliwości komputerów i technologii, przejawia się również umiejętnością poznawania i dostosowania się do zmieniającego się środowiska pracy. Do tego niezbędna jest znajomość podstawowych pojęć i idei informatycznych, takich jak reprezentacja informacji, funkcjonowanie komputera i sieci komputerowej, struktura oprogramowania, elementy algorytmiki, historia i trendy w rozwoju informatyki i technologii – które stanowią bazę dla rozumienia technologii komputerowych w ich rozwoju. Niezbędnym elementem biegłości komputerowej są również wyższego stopnia zdolności intelektualne, na które w kontekście technologii składają się m.in. myślenie abstrakcyjne, analiza sytuacji problemowych, postępowanie przez analogię, podejście problemowe, działania projektowe i zespołowe, a wszystko w odniesieniu do przetwarzania informacji.

Warto zwrócić uwagę, że alfabetyzacja komputerowa odnosi się głównie do zmieniających się elementów technologii, podczas gdy pojęcia i idee informatyczne oraz zdolności intelektualne mają charakter ponadczasowy i w nich ta zmieniająca się sfera technologii znajduje mocne oparcie i wsparcie.

W drugiej połowie pierwszej dekady XXI wieku pojawiły się obawy, że nawet biegłość komputerowa stanowi niewystarczające przygotowanie do stosowania komputerów z ich pełną mocą i możliwościami, do rozwiązywania problemów, jakie stają przed współczesnym człowiekiem, bez względu na dziedzinę, którą się zajmuje. Tak zrodziła się idea **myślenia komputacyjnego** (ang. *computational thinking*) sformułowana po raz pierwszy przez Jeannette M. Wing w 2006 roku [20]. Myślenie komputacyjne można uznać za poszerzenie myślenia algorytmicznego, które na ogół kojarzy się z informatyką, podczas gdy myślenie komputacyjne odnosi się do każdej dziedziny, w której stosowane są komputery.

Myślenie komputacyjne obejmuje szeroki wachlarz intelektualnych narzędzi reprezentujących metody modelowania i rozwiązywania problemów z pomocą komputerów, na przykład takich jak: dekompozycja złożonego problemu, aby móc go rozwiązać efektywnie, przybliżanie rozwiązania, gdy dokładne rozwiązanie jest poza zasięgiem nawet możliwości komputerów, rekurencja, czyli metoda indukcyjnego myślenia, modelowanie złożonych problemów. Podobnie jak maszyny drukarskie przyczyniły się do upowszechnienia kompetencji w zakresie 3R (***r**eading*, *w**r**iting*, *a**r**ithmetic*), tak dzisiaj komputery i informatyka przyczyniają się do upowszechniania myślenia komputacyjnego, związanego z posługiwaniem się komputerem [20].

Dużym wyzwaniem czekającym szkoły od roku 2012, czyli od wejścia reformy systemu edukacji do szkół ponadgimnazjalnych, jest oparcie kształcenia informatycznego wszystkich uczniów (przedmiot informatyka w I klasie) na idei myślenia komputacyjnego, uwzględnionej w nowej podstawie programowej informatyki, czyli na podstawie metod rozwiązywania problemów z różnych dziedzin z pomocą komputerów przy jednoczesnym uświadomieniu sobie stale rosnącej mocy komputerów oraz ich ograniczeń. To podejście do rozwiązywania problemów można scharakteryzować następującymi cechami:

- problem jest formułowany w postaci, która dopuszcza i umożliwia posłużenie się w jego rozwiązaniu komputerem lub innymi urządzeniami służącymi do zautomatyzowanego przetwarzania informacji;
- problem polega na logicznej organizacji danych i ich analizie (danymi mogą być teksty, liczby, ilustracje itp.) i wyciągnięciu z nich wniosków;
- 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 oraz sieci;
- 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 nadużywając zasobów komputera, czasu działania, pamięci, oprogramowania, zasobów informacyjnych.

Kształtowanie myślenia komputacyjnego przyjęto jako podstawowe podejście w podręczniku do informatyki dla wszystkich uczniów szkół ponadgimnazjalnych, patrz [5].

6. Najbliższe perspektywy

Trwa dyskusja o kondycji szkoły i jej przyszłości. Z jednej strony gremia specjalistów debatują nad możliwymi scenariuszami rozwoju szkoły (patrz p. 6.1), a z drugiej – pojawiają się głosy wątpiące, czy szkoła, jako instytucja, przetrwa. Można mieć takie wątpliwości obserwując „kariery z kasą", zwłaszcza w informatyce, takich osób jak Bill Gates czy Steve Jobs, chociaż wiele innych karier akurat świadczy na korzyść solidnego wykształcenia w szkole, a później w uczelni. Jednak sukces kilku indywidualności, którym szkoła specjalnie nie pomogła w karierze wystarczają, by wołać: „Nie pozwól, żeby nauczyciele zmarnowali życie twoim dzieciom"[5]. Przez pryzmat przyszłej kariery kilku geniuszy nie można jednak patrzyć na rozwój i kształcenie wszystkich uczniów. Kilku zostanie mistrzami w swoich specjalnościach, ale większość to przyszli rzemieślnicy, czyli także mistrzowie, ale na miarę swoich możliwości, zainteresowań i potrzeb. Zamiast rewolucji w edukacji można zaproponować scenariusz ewolucji, która może okazać się rewolucyjna dla szkoły, patrz [16].

W tym rozdziale wybiegamy nieco w przyszłość i rysujemy możliwy scenariusz dla szkoły w czasach społeczeństwa sieciowego, pochylamy się nad uczniem – jaki jest a jaki powinien być, by rzeczywiście był świadomym beneficjentem systemu współczesnej edukacji nastawionej na indywidualizację jego kształcenia, a na koniec zatrzymujemy się nad kilkoma metodami kształcenia, związanymi z technologiami, które lada dzień zawitają do naszych szkół, jeśli ich jeszcze tam nie ma.

6.1. Edukacja bez szkoły?

W roku 2001, w ramach programu **Szkoła przyszłości** (ang. *Schooling for Tomorrow*), prowadzonego przez Ośrodek Badań Edukacyjnych i Innowacji, afiliowany przy Organizacji Współpracy Gospodarczej i Rozwoju (ang. **OECD** – *Organization for Econimic Co-operation and Development*), opracowano sześć scenariuszy dotyczących przyszłości szkoły do roku 2020 (patrz *Sześć scenariuszy dla przyszłości szkoły* w [13]). Najbliższy obecnym zmianom w technologii i w edukacji wspieranej technologią jest scenariusz 3a, będący odbiciem mechanizmów społeczeństwa sieciowego. Charakteryzujemy tutaj krótko zmiany według tego scenariusza.

[5] Piotr Cieśliński, *Dlaczego szkoła przegapiła ich talenty*, „Gazeta Wyborcza", 17-18.09.2011.

Niezadowolenie ze szkoły, jako instytucji, może prowadzić do jej porzucenia na korzyść sieci uczenia się, bazujących na efektywnych i coraz tańszych rozwiązaniach wykorzystujących technologie sieciowe. Za takim rozwiązaniem mogą stać liberalne grupy obywateli przewidujące upadek instytucji państwa, jak również grupy społeczne i religijne wspierane przez partie polityczne, media i komercyjne firmy z branży technologicznej. Prowadzić to może do załamania się narodowych systemów edukacyjnych, upadku roli władz publicznych i jednoczesnego rozwoju lokalnych systemów szkolnych (intranet) i w sieciach globalnych (Internet).

Jest to jedna z częściej przedstawianych wizji przyszłości edukacji, bazująca na widocznych tendencjach w rozwoju społeczeństwa ku społeczeństwu sieciowemu, zbudowana na potędze technologii i jej możliwościach do stworzenia systemu kształcenia, bez ograniczeń co do miejsca i czasu kształcenia się. W przeciwieństwie do modelu rynkowego, konkurencja jest zastąpiona współpracą. Poważne zastrzeżenia budzi brak w tym scenariuszu ukrytych funkcji systemu kształcenia, w tym przystosowania do życia w społeczeństwie. Zasadniczym problemem może być los osób wykluczonych przez sieciowy model rozwoju społeczeństwa.

Oto cechy edukacji w modelu sieciowego kształcenia:

- *Kształcenie i organizacja:* Większa waga przykładana jest w tym scenariuszu do kształcenia w różnych kulturach, wartościach za pomocą sieci pozostających w dyspozycji różnych grup społecznych, wyznaniowych, interesu, a także rodzin. Powszechne staje się kształcenie zindywidualizowane, w małych grupach, w domu. Zanika rola szkoły, jako miejsca kształcenia, i nauczyciela, jako profesjonalisty w kształceniu. Technologia jest w większym stopniu wykorzystywana w kształceniu, co ma wpływ na rozwój rynku oprogramowania. Znika różnica między początkową a ciągłą fazą ustawicznego kształcenia. Tradycyjne szkoły mogą pozostać dla tych, którzy zostali wykluczeni przez technologię.
- *Zarządzanie:* Kształcenie dostępne za pośrednictwem sieci zmniejsza rolę zarządzania przez instytucje edukacyjne w obecnym sensie. Pozostawia jednak obowiązek na służbach publicznych zapobiegania cyfrowemu wykluczeniu i rozwarstwianiu.
- *Zasoby i infrastruktura:* Znacząca jest redukcja udziału instytucji publicznych (dzisiejszych szkół) na korzyść rozwoju sieciowej struktury. Kształcenie wspierane przez różne formy udziału: prywatne, dobrowolne, społeczne. Następuje ekspansja, często agresywna, firm z branży technologicznej i multimediów.
- *Nauczyciele:* Zanika rola nauczyciela w tradycyjnym ujęciu, zaciera się lub znika rozróżnienie między: nauczycielem a uczniem, rodzicami a nauczy-

cielem, edukacją a społecznością uczących się. Pojawia się nowa profesja konsultanta, wykorzystywana w nauczaniu sieciowym, zdalnym i w doradztwie.

6.2. Dzisiejszy uczeń w dzisiejszej szkole

Obecnie jeszcze bardziej niż dotychczas w większości współczesnych systemów edukacyjnych przyjmuje się, że naczelnym priorytetem kształcenia jest personalizacja, czyli podmiotem kształcenia jest uczący się, ze swoimi zainteresowaniami, możliwościami i potrzebami edukacyjnymi, zawodowymi i osobistymi oraz sposobami uczenia się i kształtowania wiedzy. Dodatkowych argumentów na korzyść personalizacji dostarczają badania nad mózgiem, których konkluzję można streścić – każdy mózg jest inny – lub dosadniej, w terminach technologii:

Every brain is wired differently (...)
(Każdy mózg jest inaczej okablowany)

John Medina, [7]

Dlatego we wszystkich dokumentach unijnych, a także w większości dokumentów krajowych, określających kierunki rozwoju edukacji w społeczeństwie i priorytety systemów kształcenia, głównym podmiotem kształcenia jest uczący się. Jeśli zaś chodzi o relację między edukacją i technologią, to może to zabrzmieć jak odwrócenie ról, ale, by technologia rzeczywiście okazała się wsparciem dla edukacji, zwłaszcza indywidualnej, sama wymaga wsparcia udzielonego jej przez człowieka, wsparcia ideami i metodami kształcenia, w których dopiero może znaleźć swoje miejsce.

Technologia komputerowa od czasów dużych komputerów podążała również drogą coraz większej personalizacji, chociaż początki komputerów osobistych IBM PC w szkołach dalekie były od indywidualnego korzystania z nich przez pojedynczych uczniów. Najpierw IBM PC były dzielone przez dziesiątki uczniów, aż dopiero ostatnio poczyniono znaczny postęp w zbliżeniu się do idei „jeden uczeń jeden komputer" – piszemy o tym dalej w p. 6.5.1.

Od jakiegoś już czasu personalizacja technologii podąża swoimi ścieżkami, obok personalizacji kształcenia. Dzisiaj już niemal każdy uczeń nosi przy sobie i przynosi do szkoły urządzenia, które często przewyższają swoją mocą i zakresem możliwości komputery znajdujące się na wyposażeniu szkoły. Dzięki tej technologii uczeń cały czas (określa się to skrótem 24/7 – przez 24 godziny na dobę, przez 7 dni w tygodniu) ma dostęp do informacji dostępnych w sieci Internet, może również korzystać z tych urządzeń w komunikacji z innymi uczniami i nauczycielami. Ciśnie się więc pytanie, dlaczego uczniowie nie mogą swobodnie korzystać w szkole z możliwości, jakie dają im ich własne urządzenia. Postaramy się na to odpowiedzieć w dalszej części, patrz p. 6.5.2.

Jednym z wyzwań stojących przed szkołą jest zniwelowanie podziału między warunkami pracy w szkole – często z użyciem przestarzałej technologii – a warunkami, z którymi uczniowie spotykają się poza szkołą. Jak spowodować, by uczeń wiecznie połączony z innymi i podłączony do „repozytorium wszelkiej wiedzy" (tak często określa się Internet) korzystał z tych połączeń w swoim kształceniu się i rozwoju, ale nie tylko w szkole, również poza szkołą, w tym także w domu. Tak zrodziło się wyzwanie **uczyć się będąc połączonym** (ang. *learning while we are connected*), obrane jako temat Światowej Konferencji na temat Komputerów w Edukacji WCCE 2013 w Toruniu, patrz szczegóły na stronie http://wcce2013.umk.pl.

Zwróćmy jeszcze uwagę, że globalność technologii i powodowanych przez nią zmian powoduje, że szkoła, a nawet systemy edukacji straciły „granice", jakimi do niedawna były: mury szkoły, dokumenty (podstawy i programy nauczania) i standardy edukacyjne, ramy formalnych i nieformalnych form kształcenia. Kształcenie incydentalne (a więc przy różnych okazjach) robi zawrotną karierę. Brytyjczycy ocenili, że osoby w wieku szkolnym niemal 70% swojej wiedzy zdobywają poza szkołą! Obowiązek szkolny podrywa każdego dnia na nogi miliony uczniów, którzy coraz częściej zdają się powtarzać, po co nam szkoła?! Zadaniem szkoły, jeśli ma nadal istnieć, pozostaje wyrobić w uczniach przekonanie, że szkoła „nie przeszkadza im w kształceniu się", ale może pomóc w wyrobieniu wyobrażenia, czym może być: *my education* – **moje wykształcenie**[6]**.**

Jak wspomnieliśmy na początku tego rozdziału, komputer i technologia w edukacji występują w podwójnej roli, obiektu i wsparcia (narzędzia) kształcenia. Funkcjonowanie zaś w społeczeństwie informacyjnym wymaga wykształcenia tzw. **kompetencji XXI wieku**, do których zalicza się:

- umiejętność rozwiązywania problemów i podejmowania decyzji;
- twórcze i krytyczne myślenie;
- zdolność komunikowania się i współpracy;
- umiejętność prowadzenia negocjacji;
- intelektualną ciekawość;
- umiejętność krytycznego wyszukiwania, selekcji, porządkowania i oceniania informacji;
- wykorzystywanie wiedzy w nowych sytuacjach, integrowanie technologii z kształceniem i własnym rozwojem.

Zauważmy, że po pierwsze – technologia pojawia się *explicite* dopiero w ostatnim punkcie, a więc w kształtowaniu tych kompetencji pełni rolę narzędzia wsparcia. Po drugie zaś, te kompetencje nie są skupione na przedmiotach

[6] Wsparliśmy się tutaj powiedzeniem Marka Twaina *I have never let my schooling interfere with my education* – Nigdy nie dopuściłem, by chodzenie do szkoły zaszkodziło mojemu (wy)kształceniu.

nauczania, ale rozciągają się ponad dziedzinami kształcenia. Dominujący jeszcze system klasowo-lekcyjny jest jednak często ciasnym gorsetem dla rozwijania tych kompetencji, można się więc spodziewać, że będzie stopniowo poluźniany.

6.3. Mobilna edukacja

Rozwój technologii w ostatnich latach doprowadził do wykreowania na potrzeby szkoły modelu mobilnej technologii, która wspiera mobilną edukację. Mobilny to przenośny. Można powiedzieć, że IBM PC był także komputerem przenośnym, ale tutaj jednak chodzi o technologię, która może być dostępna uczniom i nauczycielom w dowolnym czasie (ang. *anytime*) i w dowolnym miejscu (ang. *anywhere*), jeśli tylko jej potrzebują, dodatkowo jest spersonalizowana do ich potrzeb. Na **technologię mobilną** składają się:

a. **komputery przenośne** (np. laptopy, tablety, smartfony), wyposażone w kartę sieciową do bezprzewodowego dostępu do Internetu i przeznaczone do indywidualnego wykorzystywania,
b. **bezprzewodowy dostęp** do Internetu,
c. **platforma** wypełniona zasobami edukacyjnymi i służąca do organizacji kształcenia, dostępna w każdej chwili z dowolnego miejsca, w którym jest dostęp do Internetu,
d. **zmiany w organizacji dostępu do technologii** w szkole i w domu.

Technologia mobilna w szkole umożliwia zmianę sposobu korzystania z technologii przez uczniów i przez nauczycieli – zamiast poszukiwania dostępu do niej placówce oświatowej, **technologię można znaleźć w szkole wszędzie tam, gdzie jest potrzebna uczniom i nauczycielom**. Poza nią również. Wymaga to wielu zmian w tradycyjnej szkole, zarówno w sposobach uczenia się (uczniowie) i nauczania (nauczyciele), jak i w organizacji pracy szkoły (personel administracyjny i zarządzający), a także w społeczności lokalnej, której trzon stanowią rodzice i rodziny uczniów.

Na bazie mobilnej technologii można określić model mobilnej edukacji, opisujący takie warunki kształcenia, w których edukacyjny rozwój ucznia następuje nie tylko w warunkach systemu klasowo-lekcyjnego, ale korzystając z wszelkich udogodnień, kształcenie może przebiegać w dowolnym czasie i w dowolnym miejscu, jeśli tylko takie są potrzeby, zainteresowanie i wola uczących się.

Model mobilnej edukacji można scharakteryzować następującymi postulatami:

1. Przeniesienie nacisku z nauczania (*teaching*) na **uczenie się** (*learning*).
2. Przejście od modelu *teacher centered* do *learner centered*, czyli **uczeń** staje się głównym **podmiotem edukacji**.
3. Istnieją daleko zaawansowane możliwości personalizacji, czyli tworzenia **indywidualnych środowisk i ścieżek kształcenia**.

4. Uczący się gromadzi swoje własne zasoby w **osobistym archiwum** i może tworzyć na ich podstawie **e-portfolia**, będące materiałem do refleksji nad własnym kształceniem i rozwojem oraz współczesną wersją wizytówki uczącego się, ilustrującą jego rozwój i możliwości, suplementem certyfikatów.
5. Realizowana jest idea *learning anytime* i *anywhere*, czyli uczenia się w dowolnym czasie i w dowolnym miejscu, co wymaga świadomego **zaangażowania ucznia**.
6. Proces kształcenia ma charakter **asynchroniczny** (nie wszyscy uczą się jednocześnie i tego samego) i **rozproszony** (przebiega w różnych miejscach i w różnym czasie).
7. System kształcenia jest oparty na **ideach konstruktywistycznych**, czyli budowania i rozwoju wiedzy przez uczniów w rzeczywistym środowisku ich przebywania i rozwoju.

Wszystkie te postulaty mogą być spełnione w warunkach korzystania z wirtualnego środowiska edukacyjnego, jakim jest **platforma edukacyjna**, która w modelu mobilnej edukacji spełnia rolę, którą w ujęciu tradycyjnym odgrywa szkoła.

Dwa najważniejsze aspekty w modelu edukacji mobilnej to: uczeń w centrum uwagi i personalizacja elektronicznych środowisk rozwoju i kształcenia (na platformie edukacyjnej). Wyznaczają one kierunki działań i określają rolę i miejsce technologii. W szczególności, dostęp uczniów do technologii powinien być rozważany nie w kategoriach dostępu do komputera, jako urządzenia, ale dostępu do elektronicznych środowisk, które towarzyszą edukacji, w których uczniowie się kształcą, a komputer to tylko furtka do tych środowisk i okno na świat. Taką furtką może być również komputer stacjonarny w szkolnej pracowni, w domu[7] lub w innym miejscu. Ten dostęp powinien być w każdym miejscu, w którym może być potrzebny i to nie tylko uczniom, ale także ich rodzicom, nauczycielom, personelowi szkoły oraz organom prowadzącym szkoły.

6.4. e-szkoła

Nie tak dawno pojawiło się określenie e-szkoła, nie tylko u nas w kraju, ale również w innych państwach (jako *e-school*, *eSchool*). W jednym z dokumentów eksperckich zaproponowano bardzo ogólne znaczenie tego terminu:

[7] Pod koniec 2008 roku, w ramach jednego ze szkoleń nauczycieli, prowadzonego w województwie kujawsko-pomorskim przez Regionalne Studium Edukacji Informatycznej (RSEI) na WMiI UMK w Toruniu, przeprowadzono ankietę wśród uczniów z blisko 70 klas w szkołach podstawowych, gimnazjach i liceach. Okazało się, że w zdecydowanej większości 100% uczniów w klasie miało dostęp do komputera w domu. Było to olbrzymim zaskoczeniem dla prowadzących zajęcia. Zaskoczenie *in minus* przyniosła analiza scenariuszy lekcji z wykorzystaniem technologii, opracowanych przez nauczycieli będącymi słuchaczami tego studium – tylko jeden z nauczycieli uwzględnił zadanie do wykonania przez uczniów za pomocą komputera w domu.

e-szkoła – to szkoła, która wykorzystuje technologie w procesie swojego rozwoju ku lepszemu, bardziej skutecznemu wypełnianiu swojej misji edukacyjnej, wychowawczej i społecznej.

W takiej szkole, jak również poza jej murami, technologia jest dostępna zawsze wtedy, kiedy oraz wszędzie tam, gdzie potrzebuje jej uczeń, nauczyciel i personel szkoły. Zapleczem technicznym e-szkoły jest bezprzewodowy dostęp do Internetu na terenie szkoły, mobilne (przenośne) komputery w szkole, jak pracownie laptopów, tablice interaktywne i inne urządzenia, oraz dostęp do komputerów i do Internetu w domach uczniów. Środowiskiem edukacyjnym jest zaś platforma edukacyjna, stanowiąca pierwszy krok na drodze ku **chmurze edukacyjnej**, do której będą mieć otwarty dostęp wszyscy aktorzy w teatrze szkoły. A zatem w e-szkole jest realizowany model edukacji mobilnej. Takie rozwiązanie jest wdrażane m.in. jako Dolnośląska e-szkoła.

Należy pamiętać, że opisane w tym rozdziale idee i projekty związane z wdrażaniem nowych technologii w szkołach, a ogólnie – w systemach edukacji, zgodnie z ideą Nicholasa Negroponte, animatora ogólnoświatowego programu *One Laptop Per Child* (pol. laptop dla każdego dziecka), nie dotyczą technologii, ale są projektami edukacyjnymi, gdyż założone w nich efekty edukacyjne, jak w modelu edukacji mobilnej, są znacznie trwalsze niż technologia, która służy jako wsparcie.

Ważnym elementem projektów typu e-szkoła jest potraktowanie pobytu ucznia w szkole jako jednego z epizodów jego **kształcenia się przez całe życie** (ang. *lifelong learning* – LLL), ich celem jest więc również położenie podwalin pod ustawiczne kształcenie, w szkole i poza nią, a przede wszystkim po wypełnieniu obowiązku szkolnego.

6.5. Nowe rozwiązania edukacyjne i technologiczne

Przedstawiamy tutaj krótko aktualne trendy w rozwoju edukacji, wspierane rozwojem technologii kształcenia.

6.5.1. Strategia 1:1

Z chwilą pojawienia się w połowie lat 80. XX wieku IBM PC – **komputera osobistego** – zakiełkowała idea, by każdego ucznia wyposażyć w szkole w osobisty komputer. Przez długie lata ten komputer w szkołach tylko z nazwy był osobisty i nawet w krajach zamożnych liczba uczniów przypadających na jeden komputer daleka była od 1. Jednak przez wiele lat proporcja liczby uczniów do

liczby komputerów w szkole była miernikiem stopnia komputeryzacji szkół. Tym miernikiem posługiwali się zwłaszcza politycy, uzasadniając swoje decyzje zakupu kolejnego sprzętu dla szkół. Również w dokumentach Unii Europejskiej królował ten miernik komputeryzacji szkól w poszczególnych krajach członkowskich. W szkołach zaś starano się, by komputer PC był rzeczywiście wykorzystywany przez pojedynczych uczniów, zatem zajęcia z informatyki były jednymi z niewielu, na których klasy mogły być dzielone na grupy, gdyż pracownie komputerowe zazwyczaj liczyły po 10-15 komputerów. Przyjmując na przykład, że na jeden komputer w szkole przypada średnio 10 uczniów – odpowiada to skali nasycenia komputerami polskich szkół w roku 2012 – i zajęcia w szkole trwają przez 40 godzin tygodniowo, każdy uczeń ma komputer do swojej wyłącznej dyspozycji w szkole średnio przez 4 godziny tygodniowo. Powiedzmy, że w tym 2 godziny zajmują zajęcia z informatyki, pozostają 2 godziny na inne przedmioty. To niezłe warunki korzystania z komputerów przez uczniów w naszych szkołach.

Przyjęty miernik komputeryzacji szkół w wielu krajach zaczął zbliżać się do 1 i tak pojawiła się **strategia 1:1**, będąca rzeczywistą szansą wyposażenia każdego ucznia w osobisty komputer. Zgodnie z nią, każdy uczeń powinien mieć do swojej dyspozycji komputer przez cały czas przebywania w szkole. Można tutaj wyróżnić dwa warianty tej strategii, gdy uczeń korzysta z tego osobistego komputera tylko w szkole i drugi – gdy może go również zabrać do domu. Ta druga opcja może polegać na tym, że to rodzice kupują każdemu uczniowi komputer, z którym on chodzi do szkoły. Takie rozwiązanie przyjęto w Portugalii, ale po jakimś czasie okazało się, że nie wszyscy uczniowie przynoszą swoje komputery do szkoły.

Strategia 1:1 stała się bardziej realistyczna, gdy na rynku zaczęły pojawiać się komputery przenośne (mobilne), takie jak laptopy, notebooki, netbooki itp. Były nawet specjalne wersje tych komputerów, przeznaczone specjalnie dla szkół, takie jak Classmate PC (oparty na technologii firmy Intel), czy OLPC XO (oparty na technologii firmy AMD).

Inicjowano również specjalne projekty, których celem było wdrożenie strategii 1:1 w szkołach w większym regionie. Jednym z najwcześniejszych był projekt **MLTU** (*The Maine Learning Technology Initiative*) stanu Maine w USA, rozpoczęty we wrześniu 2002 roku. W styczniu 2010 roku wszyscy uczniowie (29 570 osób) i nauczyciele (4468) w *middle schools* (klasy 7 i 8) byli wyposażeni w laptopy i ponad 55% uczniów i nauczycieli w *high schools* miało własne laptopy. Gubernator stanu Maine Angus King, który w czasie swojej kadencji (1995-2003) inicjował projekt MLTU, po latach tak podsumował doświadczenia tego projektu: *Raport podkreśla to, czego nauczyliśmy się tutaj w Maine, że komputer jest niezbędny na początku, ale sam nie jest w stanie spowodować istotnych zmian, których tak oczekujemy. Uzmysłowiliśmy sobie również, że projekt fak-*

tycznie dotyczy nauczycieli i ich przewodniej roli w szkole, dzięki profesjonalnemu przygotowaniu i nowej pedagogice, pojawiają się zadziwiające rezultaty, których trudno oczekiwać, dostarczając tylko laptopy do szkół [11][8].

Innym projektem, ale na skalę światową, jest wspomniany już projekt **OLPC** (*One Laptop Per Child*) – **laptop dla każdego dziecka**. Zainicjował go Nicholas Negroponte jesienią 2005 roku, przedstawiając na Światowym Szczycie Społeczeństwa Informacyjnego, który odbył się w Tunisie, pomysł na laptop XO za 100 dolarów, zaprojektowany dla uczniów z krajów rozwijających się. Cena nie została utrzymana, wzrosła do 180 dolarów, i w następnych latach projektowano kolejne modele komputera XO, w tym także tablet. Do roku 2011 włącznie dostarczono dzieciom, uczniom i do szkół ponad 2,4 miliona laptopów XO – należy to uznać za olbrzymi sukces tego projektu.

Od jakiegoś czasu miernik komputeryzacji edukacji (szkół), bazujący na liczbie uczniów przypadających na jeden komputer, już nie jest stosowany, gdyż szkoły zaczęły mieć wystarczająco dużo sprzętu komputerowego, a z drugiej strony, jak pokazują wyniki *Diagnozy Społecznej 2009* [2], domostwa uczniów są niemal w 100% wyposażone w komputery, w większości z dostępem do Internetu. Poważniejszym wyzwaniem dla szkół stało się więc zarówno wykorzystanie w celach edukacyjnych komputerów, które są w szkołach, które uczniowie mają w domach, jak i urządzeń mających funkcje komputera, które uczniowie noszą przy sobie i z którymi przychodzą do szkoły. Piszemy o tym dalej w p. 6.5.2.

W ostatnich kilku latach rząd inicjował programy, których celem było zmierzenie się ze strategią 1:1, czyli zaspokojenie potrzeb szkół, a zwłaszcza uczniów, na osobisty sprzęt komputerowy.

W pilotażu programu „Cyfrowa szkoła", którym w roku szkolnym 2012/2013 ma być objętych ok. 400 szkół, w wariancie II tego programu uczniowie z klas IV biorących udział w pilotażu będą mogli wypożyczyć laptopy do domu. W odniesieniu do tych uczniów będzie to realizacja strategii 1:1, ale co z innymi uczniami z klas IV, uczniami z innych klas, uczniami przychodzącymi do szkoły w następnych latach? Strategia 1:1 oznacza objęcie nią wszystkich uczniów w szkole, inaczej wprowadza większe rozwarstwienie niż było dotychczas między tymi, którzy mają laptopy, a tymi, którzy nie mają do nich dostępu.

Przy okazji wcześniejszej inicjatywy rządowej „Komputer dla ucznia", ogłoszonej na wiosnę 2008 roku, która ze względu na światowy kryzys finansowy nie doczekała się realizacji, w ekspertyzie sporządzonej na potrzeby tego projektu [14] opisano **złagodzoną strategię 1:1**, zgodnie z którą proponowano, by szkoły zostały wyposażone w mobilne zestawy laptopów w ilości, określonej przez szkoły w ich programach wdrażania technologii do zajęć. Liczba komputerów w zestawie i liczba zestawów miały gwarantować, że na zajęciach, na któ-

[8] Fragment wypowiedzi w tłumaczeniu autora.

rych jest stosowana technologia, każdy uczeń lub zespół uczniów wykonujących projekt ma laptopa do swojej wyłącznej dyspozycji. W powiązaniu z bezprzewodowym dostępem do Internetu w szkołach dawało to faktycznie efekt strategii 1:1 na wszystkich zajęciach, na których jest wykorzystywana technologia. Ta złagodzona strategia 1:1 jest znacznie tańsza (dla szkół, samorządów i dla rządu), a co najważniejsze, sprzęt jest wykorzystywany celowo wtedy, kiedy rzeczywiście jest potrzebny uczniom i nauczycielom (określają to szkoły). To złagodzenie strategii 1:1 nie uwzględnia jednak, że uczeń może chcieć skorzystać z Internetu w dowolnej chwili pobytu w szkole, na przykład do odtwarzania e-podręcznika, ale w tym celu mogą być przydatne urządzenia osobiste uczniów, o czym piszemy w następnym punkcie.

6.5.2. BYOD

BYOD (ang. *Bring Your Own Devices*) to zaproszenie do przyniesienia na zajęcia w szkole swojego urządzenia elektronicznego – weź ze sobą do szkoły swoje urządzenie. Może to być smartfon, tablet, telefon komórkowy, laptop itp. To podejście jest poszerzeniem strategii 1:1. Polega na skorzystaniu z wyposażenia poszczególnych uczniów. Faktycznie, nie trzeba uczniów do tego zapraszać, na ogół każdy z nich nosi takie lub podobne urządzenie przy sobie, zazwyczaj w kieszeni. Zaproszenie do przyniesienia do szkoły to jednak coś więcej – to jednocześnie zezwolenie na korzystanie z tych urządzeń na lekcjach. Tu pojawia się jednak wiele problemów, związanych zwłaszcza z różnorodnością tych urządzeń, ale nie tylko:

- nauczyciel zwykle zna bardzo ograniczoną liczbę różnorodnych urządzeń; na ogół smartfonu lub telefonu komórkowego używa tylko do telefonowania;
- czy wykorzystywane na lekcjach aplikacje będą miały jednakowe funkcjonalności na wszystkich urządzeniach? – jeśli nie, to trudno będzie poprowadzić zajęcia jednolite dla wszystkich uczniów;
- co począć z awariami różnorodnego sprzętu – zajęcia nie powinny być zakłócane indywidualnymi awariami; problemem może być zasilanie różnorodnych urządzeń z niewielu źródeł energii w klasie (w szkole);
- jak zabezpieczyć urządzenia uczniowskie, urządzenia i serwery szkolne przed wykroczeniami i przestępstwami przeciwko prawu autorskiemu i ochronie dóbr intelektualnych?

Kwestie technologiczne związane z BYOD są już przedmiotem zainteresowania poważnych graczy na rynku technologii internetowych (np. Cisco). Jednak koszty takich rozwiązań są dość wysokie i stawiają pod znakiem zapytania ich opłacalność w porównaniu z korzyściami edukacyjnymi. Ale dzisiaj to nie jest tylko problem edukacji, gdyż w każdej firmie większość pracowników ma przy sobie urządzenia, które umożliwiają im stałe połączenie i komunikację w ramach

firmy, i ze światem zewnętrznym. Na rozwiązanie kwestii edukacyjnego wykorzystania urządzeń posiadanych przez uczniów w klasie przyjdzie nam pewnie jeszcze poczekać. Sprawdzianem mogą być e-podręczniki, o których na ogół zakłada się, że powinny dać się odtwarzać na dowolnym komputerze, tablecie czy smartfonie.

Pośrednim etapem między wyposażaniem szkół w sprzęt a ideą BYOD mogłoby być wykorzystanie przez uczniów poza szkołą w celach edukacyjnych tego sprzętu, który noszą przy sobie i tego, który mają w domach. Miejscem, w którym by się spotykali uczniowie z uczniami i uczniowie z nauczycielami byłaby platforma edukacyjna umieszczona w chmurze.

Dwa słowa jeszcze o pochodzeniu akronimu BYOD. Otóż został on utworzony na wzór innego akronimu, który często można znaleźć na zaproszeniach na różne imprezy towarzyskie organizowane w Stanach Zjednoczonych – **BYOB**, gdzie **B** pochodzi od ang. *bottle* (butelka). Czasem, obok akronimu BYOD, jest stosowany w edukacji akronim **BYOT** i można by przypuszczać, że to **T** pochodzi od ang. *technology* (technologia). Faktycznie pod tym **T** kryje się w edukacji *thinking* (myśl, myślenie), można więc przełożyć BYOT na **idąc do szkoły nie zapomnij głowy**.

6.5.3. Odwrócona klasa, odwrócone uczenie się

Odwrócona klasa (ang. *flipped classroom*) lub odwrócone uczenie się to idea, która ma wiele wspólnego z mieszanym uczeniem się (ang. *blended learning*) oraz z bardzo popularną *Khan Academy*. Polega na wykorzystaniu potencjału uczących się w domu i lepszym wykorzystaniu czasu na zajęciach w szkole. Nauczyciel w klasie krótko (5-10 min) wprowadza uczniów do nowego tematu i daje do wykonania proste ćwiczenia. Uczniowie przeglądają w domu wideo z pełnym wytłumaczeniem tematu, mogą je przeglądać wielokrotnie, w całości lub tylko fragmenty i wykonują zadane ćwiczenia. Mogą przy tym kontaktować się (na ogół w specjalnym serwisie) z innymi uczniami oraz z nauczycielem. Po przyjściu do szkoły, w klasie odbywa się wyjaśnianie wątpliwości, rozwiązywanie dalszych zadań, dyskusja z uczniami. Jedną z wersji tego podejścia może być praca metodą projektów, stosowana w całym podręczniku [5].

Niektóre cechy tego podejścia:

- bardziej odpowiada potrzebom uczniów, mogą uczyć się niezależnie od innych uczniów w zróżnicowany sposób i w zróżnicowanych warunkach, ale też żaden uczeń nie ma gdzie się „ukryć" przed nauczycielem;
- umożliwia częstsze kontakty uczniów z nauczycielem, zwiększa ich zakres; kontakty te mogą być *on-line* lub być asynchroniczne; umożliwia także kontakty między uczniami poza klasą;
- lepiej służy indywidualizacji kształcenia, zarówno uczniom, jak i nauczycielom, zwłaszcza w dużej i/lub zróżnicowanej pod względem zainteresowań i możliwości grupie uczniów, którymi zajmuje się nauczyciel;

- uczniowie lepiej poznają materiał zajęć, we własnym tempie, w odpowiednio dostosowanych do siebie warunkach uczenia się.

Odwrócona klasa wymaga odmiennej kultury uczenia się, w której faktycznie **edukacja jest w rękach uczących się**. Jest też dobrym rozwiązaniem w sytuacji, gdy uczniowie pozostają w domu, nie biorą udziału w zajęciach szkolnych, albo rzadko są w szkole z powodów zdrowotnych lub innych.

Implementacja tego podejścia wymaga dobrego przygotowania nagrań – nie każdy nauczyciel jest jednak dobrym aktorem. Innym słabym punktem może być brak chęci uczniów do spędzania w domu dłuższego czasu na oglądaniu i przysłuchiwaniu się wideo.

Organizacyjnie, kształcenie w tym stylu może przebiegać na platformie edukacyjnej, jednak w tym przypadku platforma jest nie tylko repozytorium zasobów uczniów i nauczycieli zarządzanym przez nauczyciela, ale jest spersonalizowanym środowiskiem kształcenia zarządzanym przez uczniów.

Istnieją już platformy edukacyjne, np. TED-Ed o wolnym dostępie, które umożliwiają prowadzenie zajęć w trybie odwróconej klasy, dostarczając narzędzia do tworzenia wideo i korzystania z nich na zajęciach oraz wykorzystywania wideo przez uczniów w sposób spersonalizowany. Platforma ta rozszerza możliwości nauczyciela do działań poza klasą. Do takich celów może być również dostosowana platforma Fronter, wykorzystywana w projekcie Dolnośląska e-szkoła.

6.5.4. e-podręcznik

Elektroniczna książka, w skrócie **e-książka** (*eBook*), to książka zapisana w postaci elektronicznej. Szczególnym przypadkiem e-książki jest **e-podręcznik**. Może być czytana (a ogólniej – odtwarzana) na komputerze lub za pomocą specjalnego urządzenia, zwanego e-czytnikiem (ang. *e-reader*). Niektóre e-książki mogą być również odtwarzane zarówno na zwykłych komputerach PC, jak i na telefonach komórkowych (smartfonach). Idea e-podręczników wiąże się z personalizacją kształcenia, jak również ze strategią 1:1, gdyż do indywidualnego korzystania z e-podręcznika uczeń potrzebuje indywidualnego urządzenia. Wiele uniwersytetów, szkół i dystryktów w USA wyposaża swoich studentów i uczniów w iPady i gwarantuje, że wszystkie podręczniki będą w wersji elektronicznej. Oszacowano, że np. w ciągu trzech lat studiów I stopnia student zaoszczędzi dzięki takiemu rozwiązaniu przynajmniej 50%.

Czynione są starania, aby u nas w kraju podręczniki w wersji elektronicznej stały się alternatywą dla (lub tylko uzupełnieniem) podręczników w tradycyjnej postaci. Ten pomysł napotyka jednak wiele trudności – wiele argumentów za i przeciw jest dyskutowanych w [18]. Doświadczenia innych państw nie są zbyt zachęcające i warto je wziąć pod uwagę. Generalnie uczniowie z dużą rezerwą zamieniają tradycyjne podręczniki na elektroniczne. Na przykład w Korei Połu-

dniowej od roku 2015 wszystkie podręczniki miały być w wersji elektronicznej, ale w pilotażu okazało się jednak, że ponad 80% badanych uczniów woli podręczniki papierowe. Podobne doświadczenia ma wiele szkół i uczelni w USA.

Z założenia e-podręczniki mogą być bardziej atrakcyjne niż tradycyjne. Tylko dlaczego nie spodobały się uczniom w Korei Południowej i w USA i wolą te tradycyjne? Jednym z powodów takiego nastawienia uczniów jest właśnie forma tych podręczników, która powoduje, że e-podręczniki nie mają zamkniętej postaci. Taki podręcznik to drzwi do nieograniczonych zasobów, a za tym uczniowie i studenci nie przepadają, bo chcą być pewni, co od nich wymaga nauczyciel i w jakiej postaci. Dość często właśnie nagromadzenie różnych form przekazu w e-podręczników, podlinkowanie niemal każdego miejsca na elektronicznej powierzchni e-podręcznika powoduje, że uczeń przestaje czuć się pewny, czy wszystko „przerobił", czy nie opuścił jakiegoś odniesienia do ważnego materiału, i gdzie mogą go zawieść ciągi kolejnych odniesień. Po części wynika to z wygody uczniów, ale jeszcze nikt ich nie nauczył „czytania ze zrozumieniem" elektronicznego tekstu – w badaniach PISA (*Programme for International Student Assessment* – Program Międzynarodowej Oceny Umiejętności Uczniów) – polscy uczniowie wypadli z tego zakresu gorzej niż z czytania tradycyjnego tekstu. Symptomatyczne. Gdy zapytałem kiedyś syna, kiedy zagląda do Internetu, a kiedy do papierowej encyklopedii, gdy szuka znaczenia jakiegoś hasła, jego odpowiedź zaskoczyła mnie w pierwszej chwili, ale później znalazłem jej uzasadnienie – zagląda do papierowej encyklopedii, gdy chce coś... szybko znaleźć, a do Internetu – gdy chce skopiować. Teraz go rozumiem – wygooglowanie kilkuset tysięcy stron z odpowiedzią na proste pytanie jest żadną odpowiedzią, a w encyklopedii papierowej trafia się w dziesiątkę.

Wielką pokusą dla twórców podręczników jest łatwość zamiany papierowej wersji podręcznika na elektroniczną, zwłaszcza że każdy tradycyjny podręcznik ma również swój elektroniczny format (np. PDF). Tak powstaje wiele e-podręczników. Jeśli jednak e-podręcznik ma stanowić nowe rozwiązanie i nową edukacyjną jakość, powinien być rezultatem projektu, który nie jest obciążony rozwiązaniami znanymi z tradycyjnych podręczników, powinien powstać od zera.

Alternatywą dla podręcznika może być **komputerowe środowisko aktywności** ucznia, w którym znajdzie on materiał związany z zajęciami (podręcznik) i wiele innych funkcji, związanych z jego edukacyjnymi aktywnościami. Takie funkcje są dostępne na platformach edukacyjnych, takich jak Moodle czy Fronter. Z poziomu platformy uczeń może mieć dostęp do platform zasobowych, oferujących (za darmo lub komercyjnie) zasoby uzupełniające zasoby edukacyjne. W tej koncepcji uczeń zarządza swoim miejscem zajęć edukacyjnych i faktycznie tworzy je kształcąc się, personalizując swoje uczenie się i rozwój. Nie ma potrzeby nazywania takiego środowiska aktywności uczniów **podręcznikiem** – niech

podręcznik pozostanie nazwą tradycyjnego utworu książkowego na papierze. Ewentualne zasoby elektroniczne towarzyszące takiemu podręcznikowi mogą nosić nazwę **elektronicznej obudowy podręcznika**. Zaś nazwa e-podręcznik powinna być związana z sieciowym środowiskiem aktywności uczniów. Pozwoli to uniknąć kłopotów terminologicznych, ale ważniejsze – będzie można nadać tym terminom właściwe znaczenia.

Poza technicznymi cechami e-podręczników i wygodą korzystania z nich, z książką łączy się wiele indywidualnych cech jej czytelnika, a także szeroki kontekst historyczny, społeczny i prawny. Z jednej strony – e-podręczniki bardziej angażują uczniów i mają również charakter bardziej społeczny niż te tradycyjne, mogą być bowiem współdzielone przez wielu uczniów. Z drugiej zaś strony – e-podręcznikom brak jest nostalgicznych cech książek, takich jak: posiadanie osobistego egzemplarza na półce i ich kolekcjonowanie – półka z książkami to jeden z najbardziej cenionych elementów wystroju mieszkania.

Warto wspomnieć, że już w roku 2002 (patrz artykuł *E-podręcznik do nauczania nowoczesnych technologii* w [13]) została opisana koncepcja w pełni spersonalizowanego e-podręcznika, z pomocą którego uczeń mógłby poznawać technologię informacyjno-komunikacyjną oraz informatykę w środowisku technologii i z pomocą technologii. W roku 2004 zademonstrowano podstawowe mechanizmy tej koncepcji, niestety firma, która wykonała wersję demo, wycofała się ze współpracy. W następnych latach nie udało się przekonać i pozyskać do współpracy innych twórców oprogramowania edukacyjnego, chociaż był oferowany pełny kontent, należało jedynie stworzyć interfejs ucznia i nauczyciela. Czy koncepcja wyprzedziła epokę? W pewnym sensie tak, ale w myśleniu. Ta koncepcja e-podręcznika nie była bowiem wynikiem zastanawiania się nad możliwościami istniejącej technologii do zbudowania atrakcyjnego e-podręcznika, ale była efektem rozważań nad postacią środowiska, które byłoby najbardziej odpowiednie dla ucznia i dziedziny, którą poznaje, wspierając się technologią. Już ponad 10 lat temu w koncepcji elektronicznego środowiska dla uczących się znalazły się takie rozwiązania, jak: platforma edukacyjna, chmura edukacyjna, środowisko adaptacyjne, personalizacja i inne, bez tych nazw, bo pojawiły się one znacznie później.

7. Epilog – personalizacja i ewentualne zagrożenia

Na zakończenie kilka słów o ewentualnych zagrożeniach, jakie nieść może personalizacja z wykorzystaniem elektronicznych środowisk uczenia się – warto chuchać na zimne. Bez wątpienia, technologia taka jak platforma edukacyjna znacznie poszerza pole do personalizacji kształcenia, gdyż uczeń może:

- sprawdzić i wybrać najbardziej odpowiednią dla siebie ścieżkę kształcenia w środowisku zaprojektowanym elastycznie, odpowiednio do oczekiwań;

- przyjąć najwłaściwszy dla siebie sposób uczenia się, w wybranym przez siebie tempie, czasie i miejscu;
- mieć spersonalizowane środowisko kształcenia, dostępne dla niego *on-line* w dowolnej chwili i z dowolnego miejsca;
- mieć większy wgląd w swoje osiągnięcia i postępy oraz kontrolę nad nimi;
- budować osobiste archiwa – e-portfolia – umożliwiające dzielenie się swoimi postępami i osiągnięciami w nauce oraz transfer między instytucjami edukacyjnymi na przestrzeni całego życia.

Technologia umożliwia więc już dzisiaj tworzenie spersonalizowanych środowisk kształcenia, wyposażonych w odpowiednie mechanizmy motywujące, stymulujące i ułatwiające kształcenie, a przez to wzbogacające nauczanie i uczenie się. Środowisko to – „rękami" swoich agentów – może dostosować się (adaptować się) do bieżących potrzeb uczącego się, uwzględniając przy tym jego umiejętności i preferowany sposób i styl uczenia się. Ale w tym środowisku, sterowanym bardzo złożonym programem, uczący się nie natrafi na przypadkową informację, która może być dla niego ciekawa, lub na zapomnianą książkę, stojącą obok tej, po którą akurat sięga na półce, bo w tym programie tego nie przewidziano, chociaż te przypadkowe „spotkania" mogłyby mieć dużą wartość edukacyjną.

Personalizacja środowiska e-kształcenia faktycznie może powodować ograniczenie swobody informacyjnej [4], gdyż uczącemu się są podsuwane informacje najbardziej odpowiadające jego profilowi, z czego na ogół on skwapliwie korzysta, nie rozglądając się „na boki", których faktycznie system mu nawet nie oferuje. Jest to więc swoisty rodzaj **wykluczenia informacyjnego**. Dochodzi także do bezkrytycznego przyjmowania podawanych informacji jako tych, które przecież zostały „właściwie dla mnie dobrane". W dalszej konsekwencji, korzystanie z niemal gotowych wzorców postępowania i schematów myślenia odsuwa na plan dalszy kształcenie zdolności do podejmowania prób rozwiązywania sytuacji problemowych. Maleje również chęć podejmowania inicjatywy i realizacji własnych pomysłów, a w rezultacie – ograniczenie kreatywności uczących się. W ten sposób, krytyczne podejście do informacji i kreatywność – zaliczane do kompetencji kluczowych, niezbędnie potrzebnych obywatelom XXI wieku – mogą nie być wspierane przez personalizację środowisk e-kształcenia.

W tym rozdziale nie odnosimy się bezpośrednio do sfery społecznej, w której przebiega edukacja jednostki i całych społeczeństw. Z pozoru jednak tylko, gdyż współczesna technologia to technologia globalna i pisząc o uczniu z tą technologią w rękach, nie sposób postawić granicy między szkołą a nie-szkołą. Cała więc nadzieja w Was – uczniach – nosząc tę technologię przy sobie możecie z niej korzystać w celach edukacyjnych w dowolnym miejscu, gdzie się znajdziecie, i w dowolnym czasie, gdy tylko taką macie potrzebę. Główny motyw tego rozdziału – personalizacja kształcenia – ma szansę urzeczywistnienia tylko z Waszym udziałem.

Literatura

1. Bolter J.D., *Człowiek Turinga. Kultura Zachodu w wieku komputera*, PIW, Warszawa 1990
2. Diagnoza społeczna 2009, *Raport: Warunki i Jakość Życia Polaków*, http://www.diagnoza.com/pliki/raporty/Diagnoza_raport_2009.pdf
3. Friedrichs G., Szaff A. (red.), *Mikroelektronika i społeczeństwo. Na dobre czy na złe*, Raport Klubu Rzymskiego, KiW, Warszawa 1987
4. Gogołek W., *Kreatywność z siecią,* V Konferencja „Rozwój e-edukacji w ekonomicznym szkolnictwie wyższym", AE, Poznań 2008
5. Gurbiel E., Hardt-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka to podstawa. Zakres podstawowy, Podręcznik dla szkół ponadgimnazjalnych*, WSiP, Warszawa 2012; patrz również: http://mmsyslo.pl/Edukacja/Aktualnosci/Informatyka-dla-wszystkich-uczniow/
6. McLuhan M., *Zrozumieć media. Przedłużenie człowieka*, WNT, Warszawa 2004
7. Medina J., *Brain Rules*, Pear Press, Seattle 2008
8. Papert S., *Burze mózgów. Dzieci i komputery*, WN PWN, Warszawa 1996
9. Papert S., *The Children's Machine*, Basic Books, New York 1993
10. Postman N., *Technopol. Triumf techniki nad kulturą*, PIW, Warszawa 1995
11. Greaves T.W., Hayes J., Wilson L., Gielniak M., Peterson E.L, *Revolutionizing Education through Technology*, ISTE, Eugene, Oregon 2012
12. Sysło M.M., *Komputery w edukacji. Wyjątki z historii*, 1994, http://mmsyslo.pl/Historia/Artykuly-i-prezentacje/Artykuly-edukacja
13. Sysło M.M., *Informatyka i Technologia informacyjna w szkole*, Wrocław 2004, http://mmsyslo.pl/Materialy/Ksiazki-i-podreczniki/Ksiazki
14. Sysło M.M., *Program 1:1. Program „Komputer dla ucznia"*, Ekspertyza dla KPRM, Wrocław, Toruń 2008, http://mmsyslo.pl/Edukacja/Dokumenty/
15. Sysło M.M., *Projekt Informatyka +. Dobry przykład wyjścia uczelni do szkół*, 2010, http://mmsyslo.pl/Nauczanie/WWSI
16. Sysło M.M., *Jak moglibyśmy się uczyć*, Wrzesień 2011, http://mmsyslo.pl/Historia/Artykuly-i-prezentacje/Artykuly-edukacja
17. Sysło M.M., *Informatyka – klucz do zrozumienia, kariery, dobrobytu*, http://mmsyslo.pl/Nauczanie/WWSI
18. Sysło M.M., *Bojkot wydawnictw i autorów*, http://mmsyslo.pl/Edukacja/Aktualnosci/e-podreczniki-bojot
19. Sysło M.M. *Indywidualizacja kształcenia: idee, metody, narzędzia*, w: *Człowiek-Media-Edukacja*, (red.) Morbitzer J., WN AP, Kraków 2012 (Materiały 22 Sympozjum „Człowiek-Media-Edukacja", Kraków, 2012); patrz również http://mmsyslo.pl/Edukacja/Publikacje/Artykuly
20. Wing J.M., *Computational thinking*, „Comm. ACM" 49(3) 2006, 33-35

Prof. dr hab. Maciej M. Sysło

w połowie lat 60. XX wieku przyglądał się, jak uczniowie z I LO i III LO we Wrocławiu uruchamiali swoje programy komputerowe na komputerze Elliott 803 w ramach pierwszych w kraju zajęć z informatyki (a dokładniej, programowania i metod numerycznych) w szkole. W tym samym niemal czasie i później „przechodził Odrę", pracując kolejno na modelach 1003, 1024, 1304, 1325 tej maszyny, produkowanej nad Odrą we Wrocławiu. Od połowy lat 80. sprowadzał do Instytutu Informatyki Uniwersytetu Wrocławskiego, którym kierował, kolejne modele maszyn 8-bitowych m.in. ZX Spectrum, Amstrad, Schneider i wreszcie dziecko spoza małżeńskiego łoża rodziny Odra – Elwro 800 Junior – i przygotowywał rzesze nauczycieli do ekspansji komputerów w edukacji. W tym czasie zaczął poważnie zajmować się edukacją informatyczną, dla której przez 20 lat prowadził forum spotkań i dyskusji – konferencje „Informatyka w Szkole".

Za swój największy sukces uznaje utrzymanie, wbrew powszechnym tendencjom w kraju i za granicą, wydzielonych zajęć z informatyki w szkołach gimnazjalnych i ponadgimnazjalnych. Teraz marzy, by uczniowie z technologią w rękach, cały czas „połączeni" i z dostępem do kopalni wszystkiego (Internetu), przekonali się (za Markiem Twainem), że „chodzenie do szkoły, a ogólnie – edukacja – nie szkodzi ich kształceniu". Prowadzi więc zajęcia z uczniami i nauczycielami w wielu projektach w całym kraju, wykłada na konferencjach i spotkaniach z nauczycielami, sporządza dokumenty i ekspertyzy dla instytucji rządowych i samorządowych. Pełni wiele funkcji w instytucjach i organizacjach krajowych i zagranicznych, m.in. jest przedstawicielem Polski w Technical Committee on Education (TC 3) w Federacji IFIP. W uznaniu dotychczasowych działań związanych z wdrażaniem komputerów i technologii do edukacji w naszym kraju, Federacja IFIP przyznała Polsce organizację Jubileuszowej X Światowej Konferencji na temat Komputerów w Edukacji (WCCE, Toruń 2-5 lipca 2013). Współprzewodniczy Międzynarodowemu Komitetowi Programowemu i kieruje Komitetem Organizacyjnym tej konferencji.

syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl
http://mmsyslo.pl/

Andrzej Żyławski

Warszawska Wyższa Szkoła Informatyki

Gdy nie wiesz, do którego portu płyniesz,
żaden wiatr nie jest dobry

Seneka

Kariera w ICT – dobry wybór

Technologie ICT są współcześnie synonimem innowacyjności, nowoczesności oraz głównym motorem wzrostu społeczno-gospodarczego. Stanowią przedmiot wielu badań i rankingów, na podstawie których dokonuje się oceny poziomu jakości życia obywateli oraz szans rozwoju poszczególnych krajów. W artykule przedstawiono i krótko omówiono wybrane badania i rankingi dotyczące problematyki społeczeństwa informacyjnego i technologii ICT. Rozwój technologii ICT implikuje powstanie całej gamy profesji dostępnych przy podejmowaniu decyzji o wyborze ścieżki kariery zawodowej. Autor powołując się w artykule na badania własne oraz innych osób, pokazuje, kiedy i na jakiej podstawie jest dokonywany ten jeden z najważniejszych życiowych wyborów. W kolejnej części artykułu jest zarysowana dostępna w Polsce ścieżka edukacji informatycznej, która może prowadzić do uzyskania kwalifikacji zawodowych informatyka, z podkreśleniem wagi, jaką ma w tym zawodzie kształcenie ustawiczne. Tematyce wejścia informatyka na rynek pracy, wyboru konkretnej ścieżki specjalizacyjnej, ocenie możliwości awansu zawodowego oraz warunków materialnych proponowanych przez pracodawców poświęcono kolejną część tekstu. Na końcu krótko przedstawiono aktualny stan oraz perspektywy rynku pracy ICT – czynniki, które mogą mieć istotne znaczenie przy ocenie atrakcyjności zawodu informatyka. W opinii autora prezentowany w artykule materiał wraz z załączoną literaturą może stanowić dobry zasób referencyjny dla osób rozważających wybór ICT, jako ścieżki rozwoju zawodowego oraz dla tych osób, które o tym wyborze współdecydują, w szczególności dla rodziców oraz pracowników poradni zawodowych, jak również instytucji kształcących.

1. Wstęp

Celem artykułu jest przedstawienie i omówienie uwarunkowań związanych z wyborem, realizacją i rozwojem kariery zawodowej w ICT[1]. Wybór profesji informatyka, podobnie jak to ma miejsce w przypadku innych zawodów, powinien być poprzedzony oceną własnych zainteresowań i predyspozycji. Równie ważne jest poznanie czynników oraz motywów, które mają najbardziej znaczący wpływ na wybór przyszłego zawodu. Kolejny krok to zapoznanie się z wymogami kompetencyjnymi dla zawodu informatyka i ich konfrontacja z własnymi oczekiwaniami i możliwościami. Wybór zawodu zawsze wiąże się z koniecznością wyboru ścieżki edukacyjnej, która gwarantuje zdobycie niezbędnych do wykonywania zawodu umiejętności i kwalifikacji, potwierdzonych stosownymi dyplomami i certyfikatami. Warto również wstępnie ocenić możliwości kształcenia ustawicznego oraz awansu zawodowego w obszarze ICT. W podjęciu decyzji o wyborze profesji informatyka i dalszym rozwoju kariery zawodowej z pewnością przydatne jest zapoznanie się z podstawowymi informacjami na temat rynku pracy ICT oraz perspektywami jego rozwoju.

Problematyka kariery w obszarze ICT jest przedmiotem wielu prac naukowych oraz opracowań o charakterze popularnonaukowym z dziedziny psychologii, socjologii, pedagogiki, zarządzania czy informatyki. W artykule omówione są wyniki wybranych badań, raporty oraz dane statystyczne, które mogą być pomocne zarówno w podjęciu decyzji o wyborze informatyki jako przyszłego zawodu, jak również w podejmowaniu decyzji o kierunkach rozwoju kariery w zawodzie informatyka. Szczególne miejsce wśród prezentowanych materiałów zajmują wyniki badania losów zawodowych absolwentów studiów inżynierskich Warszawskiej Wyższej Szkoły Informatyki (WWSI) za lata 2006-2010. Badania te, chociaż dotyczą absolwentów jednej uczelni, ze względu na jej specjalizację, jednokierunkowość i liczbę absolwentów są reprezentatywne dla środowiska młodych zawodowych informatyków. W chwili obecnej są one jednym z nielicznych tak szczegółowych analiz przeprowadzonych na homogenicznej próbie absolwentów kierunku informatyka. Badania obejmują okres od momentu podjęcia decyzji o wyborze kierunku kształcenia na studiach wyższych do czasu od roku do trzech lat po ukończeniu studiów i podjęciu pracy zawodowej. Ich uzupełnieniem są wyniki badania absolwentów studiów pody-

[1] ICT to skrót od *Information and Communication Technology*, czyli technologii informacyjno-komunikacyjnej. Wcześniej stosowano skrót IT, od *Information Technology*, czyli technologia informacyjna. Skróty polskich odpowiedników tych określeń, TIK i TI, są rzadziej stosowane.

plomowych kierunku informatyka, prowadzonych w latach 2009-2012. Z uwagi na wieloletni staż pracy oraz podstawowe wykształcenie kierunkowe (dla 20% ankietowanych inne niż informatyka), badania absolwentów studiów podyplomowych pozwalają na sformułowanie wniosków dotyczących w szczególności problematyki przekwalifikowania zawodowego oraz budowania długookresowej strategii własnego rozwoju zawodowego w branży ICT.

W podsumowaniu zawarta jest próba odpowiedzi na pytanie, dlaczego warto wybrać i wykonywać zawód informatyka, niezależnie od tego, czy jest to zawód pierwszego wyboru, czy też zawód zdobyty w drodze przekwalifikowania.

2. Technologie ICT w centrum uwagi

Ponad dwadzieścia lat temu Alvin Toffler w swojej książce zatytułowanej *Zmiana władzy, wiedza, bogactwo i przemoc u progu XXI wieku* zamiast dotychczasowych podziałów krajów, opartych na kryteriach geograficznych czy społeczno-politycznych, zaproponował wprowadzenie podziału na kraje szybkie i powolne pod względem tempa rozwoju poziomu technologicznego. Dzisiaj nikt już nie ma wątpliwości, że to właśnie zdolność tworzenia nowych rozwiązań technologicznych oraz szybkość i skuteczność ich wdrożenia są głównymi czynnikami decydującymi o sukcesie danego kraju czy społeczności. Wśród technologii szczególne miejsce zajmują technologie ICT. Według The Global Information Technology Report 2008-2009 – raportu Światowego Forum Ekonomicznego (World Economic Forum): „Technologie informacyjne i komunikacyjne są kluczowym czynnikiem umożliwiającym rozwój i postęp społeczno-gospodarczy, zwiększającym wydajność pracy i wzrost gospodarczy, przyczyniają się do redukcji ubóstwa i poprawy poziomu życia. ICT w coraz większym stopniu rewolucjonizuje procesy produkcyjne, dostęp do rynków i źródeł informacji oraz relacje międzyludzkie. ICT wpływa również na efektywność działań rządów, wspierając transparentnosc, lepszą komunikację i poprawę jakości obsługi obywateli" [15]. Spektakularnym przykładem oddziaływania ICT na niemal wszystkie sfery działalności człowieka są smartfony – przenośne urządzenia telefoniczne integrujące w sobie funkcje telefonu komórkowego i komputera kieszonkowego.

W ciągu ostatnich 10 lat na świecie wyprodukowano i sprzedano ponad miliard smartfonów. Prognozy mówią o sprzedaży kolejnych dwóch miliardów do roku 2015. Oznacza to, że ponad połowa użytkowników telefonii mobilnej na świecie będzie do końca roku 2015 korzystać z nowoczesnych wielofunkcyjnych urządzeń sieciowych, mając dostęp do wszystkich udogodnień z tym związanych, tym samym również do wszystkich zagrożeń, jakie ze sobą niosą. Smartfon to jeden z symboli trzeciej rewolucji przemysłowej, w której produkcja

i konsumpcja, podobnie jak inne sfery działalności społeczno-gospodarczej, ulega procesowi cyfryzacji i globalizacji, w której powszechny jest dostęp do informacji, stanowiących obok innowacji główne źródło rozwoju społeczno-gospodarczego.

Ocena wpływu technologii ICT na innowacyjność, wydajność pracy, rozwój społeczno-gospodarczy, techniki produkcji, tworzenie i dystrybucję zasobów pracy, jakość usług w sferze administracji publicznej, opiekę zdrowotną, życie społeczne i kulturalne, środowisko czy edukację jest współcześnie przedmiotem wielu prac badawczo-naukowych, raportów i opracowań. W codziennym użyciu funkcjonują określenia, które powstały w związku z rozwojem technologii ICT: społeczeństwo informacyjne, e-gospodarka, e-biznes, e-praca, e-administracja, nowa ekonomia, gospodarka cyfrowa, cyfrowa szkoła, telepraca, telemedycyna, teleedukacja, wirtualna ekonomia, wirtualna szkoła i wiele innych. Wpływ technologii ICT na wszystkie dziedziny życia stał się na tyle ważny, że obecnie jedną z powszechnie stosowanych na świecie miar rozwoju społeczno-gospodarczego oraz poziomu cywilizacyjnego są osiągnięcia w zakresie rozwoju i wdrażania technologii informacyjno-komunikacyjnej. Osiągnięcia te prezentowane są w ponadnarodowych rankingach i opisywane w towarzyszących im raportach. W tabeli 1 przedstawiono wybrane rankingi zawierające informacje na temat pozycji poszczególnych krajów, pod względem stanu rozwoju infrastruktury i technologii ICT oraz innych czynników, które uważane są za elementy składające się na pojęcie społeczeństwa informacyjnego.

Wśród czynników, które istotnie wpływają na pozycję danego kraju w rankingach, warto zwrócić uwagę na ocenę jakości kapitału ludzkiego, w tym kwalifikacji ICT oraz na potencjał badawczo-rozwojowy każdego kraju. Wartość obu wymienionych czynników jest ważnym elementem oceny w większości rankingów, których przedmiotem jest pozycjonowanie stanu rozwoju społeczeństwa informacyjnego.

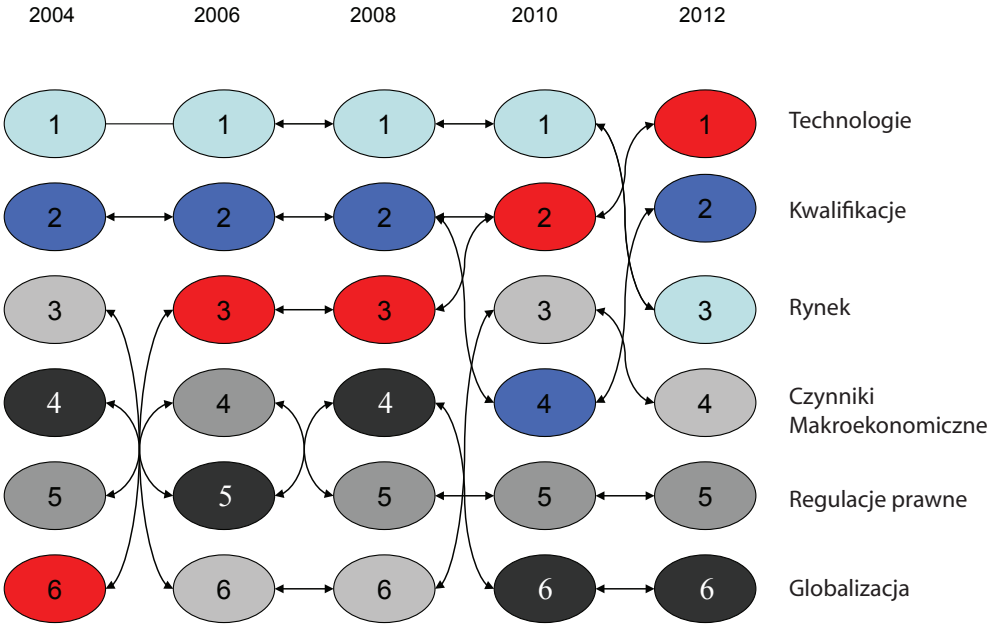
Wysoka pozycja w każdej z powyższych klasyfikacji jest tożsama z wysoką pozycją danego kraju, jeśli chodzi o rozwój społeczno-gospodarczy. Pozytywna korelacja pomiędzy stanem gospodarki i poziomem życia obywateli a stanem rozwoju i zastosowań technologii ICT skutkuje powstawaniem narodowych i ponadnarodowych programów, których celem jest stymulacja działań związanych z cyfryzacją oraz podnoszeniem kompetencji ICT, takich jak europejski program Digital Europe Agenda, czy realizowane w Polsce programy: Państwo 2.0, Interklasa, Cyfrowa Szkoła, Kierunki zamawiane, Informatyka+, IT Szkoła i wiele innych. We wszystkich rankingach i programach jednym z kluczowych elementów jest kwestia jakości kapitału ludzkiego w zakresie e-umiejętności.

Tabela 1.
Wybrane rankingi dotyczące oceny poziomu rozwoju i modernizacji infrastruktury ICT oraz wykorzystania technologii teleinformatycznych

Ranking	Raport	Kryteria oceny	Miejsce Polski w rankingu/liczba ocenianych krajów
ranking rozwoju ICT	*Measuring the Information Society 2011* [13]	dostęp do infrastruktury ICT, zastosowania technologii ICT, umiejętności w zakresie ICT	38/152
ranking gotowości sieciowej	*Living in a Hyperconnected World* [10]	wykorzystanie i stopień zaawansowania infrastruktury technologicznej, stopień przygotowania (gotowości) trzech głównych środowisk wykorzystujących technologie ICT: gospodarstw domowych, przedsiębiorstw i administracji rządowej; stopień faktycznego wykorzystania technologii ICT	49/142
ranking gospodarki cyfrowej	*Beyond e-readiness 2010* [1]	łatwość połączeń i infrastruktura ICT, biznesowe otoczenie dla technologii cyfrowych, względy prawne i polityczne oraz zastosowanie technologii w biznesie i przez konsumentów	39/70
wskaźnik konkurencyjności branży IT	*Investment for the Future, Benchmarking IT Industry Competitiveness 2011* [6]	środowisko biznesowe, infrastruktura IT, kapitał ludzki, środowisko badań i rozwoju, środowisko prawne, wsparcie rozwoju branży IT	30/66

Źródło: Raporty jak w tabeli.

To właśnie na technologie ICT i procesy digitalizacji wskazało 1700 dyrektorów korporacji (CEO) z 64 krajów, w badaniu przeprowadzonym w roku 2012 przez firmę IBM, jako na źródła inspiracji dla całkowicie nowych przemysłów oraz postępu w takich dziedzinach jak: energie alternatywne, biotechnologia, nanotechnologia – postępu, który rewolucjonizuje produkty, procesy i modele biznesowe.



Rysunek 1. Wpływ wybranych czynników na funkcjonowanie organizacji biznesowych w latach 2004-2012

Źródło: *Leading Through Connections*, IBM CEO Study, 2012 [9].

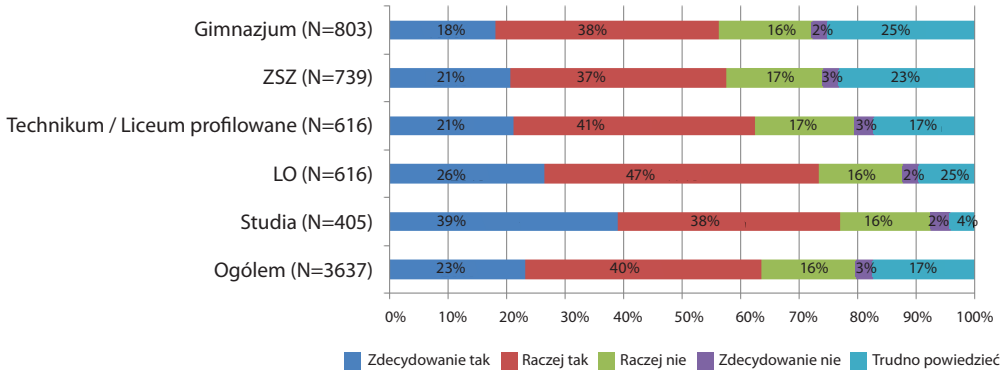
Symptomatyczne informacje dla uświadomienia znaczenia, jakie mają technologie ICT dla młodych ludzi przyniósł raport *Cisco Connected Technology Word* [2], który zawiera wyniki badań przeprowadzonych wśród studentów w wieku 18-24 lat oraz młodych pracowników w wieku 21-29 lat z 14 krajów. Blisko połowa ankietowanych (odpowiednio 49% i 47%) uważa, że Internet jest prawie tak samo ważny w ich życiu, jak woda, żywność powietrze i mieszkanie. Jedna trzecia z respondentów traktuje potrzebę posiadania dostępu do Internetu na równi z podstawowymi dla człowieka potrzebami. Dla 77% studentów i 73% młodych pracowników laptop, komputer stacjonarny i smarfon są podstawowymi źródłami zdobywania informacji i wiadomości. Na zadane pytanie: jeżeli mógłbyś mieć jedną z dwóch rzeczy, dostęp do Internetu albo samochód, co byś wybrał? 64% studentów odpowiedziało, że wybrałoby dostęp do Internetu. Ponad połowa ankietowanych (55% studentów i 62% młodych pracowników) deklaruje, że nie wyobrażają sobie życia bez Internetu, 85% studentów i 81% młodych pracowników wskazuje na laptop, komputer stacjonarny i smartfon jako najważniejsze rozwiązania technologiczne w ich codziennym życiu. Według prognoz International Data Corporation (IDC), dalszy rozwój technologii ICT i ich zastosowań spowoduje, że do 2015 roku 90% stanowisk pracy we wszystkich sektorach wymagać będzie kwalifikacji ICT w różnym zakresie. Powszechna dostępność i wykorzystanie technologii informacyjno-komunikacyjnych będzie

kreować coraz większe zapotrzebowanie na specjalistów informatyków, którzy będą je tworzyć, utrzymywać i rozwijać.

3. Planowanie przyszłości zawodowej

Pierwsze refleksje na temat tego, czym chcielibyśmy się zajmować w przyszłym życiu zawodowym pojawiają się bardzo wcześnie i są najczęściej wynikiem obserwacji życia dorosłych – rodziców i osób nam najbliższych. Potrzeba samookreślenia zawodowego w pierwszym okresie życia realizuje się poprzez udział w zabawach i grach, w których drogą wyliczanek dziecięcych, następuje przydział określonych ról zawodowych na czas zabawy.

Pierwsze bardziej świadome plany dotyczące własnej przyszłości zawodowej pojawiają się nieco później, na etapie edukacji gimnazjalnej, kiedy dokonywany jest wybór ścieżki kształcenia pomiędzy kształceniem ogólnym i zawodowym. Przedział wiekowy 13-17 lat jest dla większości młodych ludzi decydujący, jeśli chodzi o wybór przyszłego rozwoju kariery zawodowej.



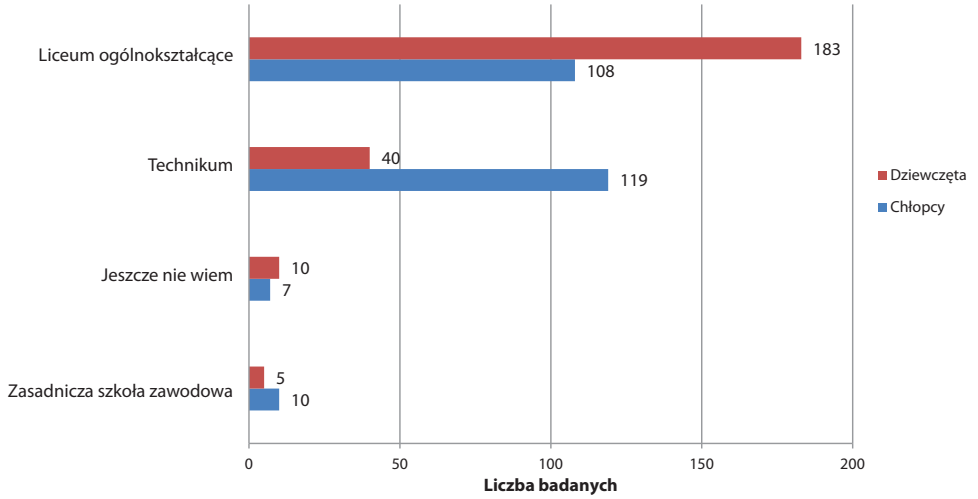
Wykres 1. Planowanie przyszłości zawodowej według typu szkoły

Źródło: *Perspektywy ludzi młodych na rynku pracy* [22].

W grupie studentów już prawie 80% spośród badanych ma mniej lub bardziej sprecyzowane plany, co do swojej przyszłej kariery zawodowej. O ile na niższych etapach edukacji brak przyszłych planów zawodowych u uczniów można wytłumaczyć między innymi słabą znajomością problematyki dotyczącej rynku pracy i uzasadnioną chęcią odłożenia decyzji na późniejszy okres, to wskaźnik ponad 20% niezainteresowanych swoją przyszłością zawodową studentów musi budzić uzasadniony niepokój. Trudno odpowiedzieć, w jakim stopniu wynik ten jest rezultatem bezradności badanych wobec trudnej sytuacji na rynku pracy, szczególnie w zawodach, w których stopień bezrobocia wśród absolwentów uczelni jest bardzo wysoki.

Wyniki badania pokazują, jak duży jest na wszystkich etapach edukacji procentowy udział młodych ludzi, którzy na pytanie o plany dotyczące przyszłości zawodowej odpowiadają, że ich nie mają lub nie mają na ten temat zdania. W przypadku uczniów techników udział ten wynosi 37%, a wśród studentów sięga 23%. W tym kontekście rodzi się pytanie o skuteczność działań z obszaru doradztwa zawodowego, których podstawowym zadaniem jest zainteresowanie uczniów problematyką pracy zawodowej, pomoc we właściwym, zgodnym z predyspozycjami i zainteresowaniami wyborze ścieżki kariery zawodowej, a także pomoc w ewentualnej decyzji, dotyczącej uzupełnienia kwalifikacji lub ich zmiany.

Pierwsze realne wybory zawodów dokonują się w szkole gimnazjalnej, kiedy uczniowie decydują się na kontynuowanie kształcenia w szkole zawodowej lub ogólnokształcącej. Cześć młodzieży podejmuje naukę w szkołach ogólnokształcących, odsuwając tym samym decyzję o wyborze zawodu o trzy lata. W ośrodkach miejskich wyborem większości gimnazjalistów jest szkoła ogólnokształcąca. W Białymstoku 36% absolwentów gimnazjów kontynuuje naukę w szkołach zawodowych – technikach – tym samym decydując o wyborze zawodu już na III etapie edukacji.



Wykres 2. Preferowany typ szkoły ponadgimnazjalnej wg uczniów gimnazjów białostockich

Źródło: *Plany młodzieży gimnazjalnej Białegostoku dotyczące kształcenia i wyboru zawodu, raport z badania* [7].

W skali ogólnopolskiej wyboru szkoły ogólnokształcącej jako kontynuacji nauki po gimnazjum dokonuje blisko 50% uczniów. Druga połowa kształci się w szkołach zawodowych – zasadniczych, technikach oraz w liceach profilowanych. W tabeli 2 przedstawiono odsetek uczniów kształcących się w zawodowych szkołach ponadgimnazjalnych w najliczniej reprezentowanych profesjach. Największy odsetek młodzieży w ramach poszczególnych kierunków kształcenia – blisko 8% kształci się w zawodzie technik informatyk.

Tabela 2.
Zmiany w odsetku uczniów kształcących się w ramach poszczególnych kierunków kształcenia w szkołach ponadgimnazjalnych w latach 2010-2011

Kierunek kształcenia	% uczniów	Miejsce 2011 (2010)
Technik informatyk	7,8	1 (1)
Technik ekonomista	5,9	2 (2)
Muzyk	5,0	3 (3)
Technik hotelarstwa	4,2	4 (4)
Technik administracji	3,7	5 (6)
Technik mechanik	3,6	6 (5)
Technik budownictwa	3,6	7 (8)
Mechanik pojazdów samochodowych	3,6	8 (7)
Technik żywienia i gospodarstwa domowego	2,8	9 (9)
Technik logistyk	2,7	10 (10)
Kucharz małej gastronomii	2,7	11 (11)
Fryzjer	2,3	12 (13)
Technik usług kosmetycznych	2,2	13 (14)
Technik bezpieczeństwa i higieny pracy	2,1	14 (12)
Technik usług fryzjerskich	2,1	15 (15)
Technik pojazdów samochodowych	2,0	16 (17)
Technik handlowiec	2,0	17 (16)
Sprzedawca	1,9	18 (18)
Technik architektury krajobrazu	1,7	19 (21)
Technik elektronik	1,7	20 (19)
Technik organizacji usług gastronomicznych	1,7	21 (20)
Technik obsługi turystycznej	1,5	22 (22)
Kucharz	1,5	23 (23)
Technik rolnik	1,5	24 (24)
Technik mechatronik	1,3	25 (25)
Technik elektryk	1,3	26 (26)
Technik organizacji reklamy	1,0	27 (30)

Źródło: *Kogo kształcą polskie szkoły* [31].

4. Wybór kierunku kształcenia (zawodu) oraz szkoły wyższej po szkole ponadgimnazjalnej

Ukończenie szkoły ponadgimnazjalnej to dla uczniów kolejna okazja do podjęcia decyzji dotyczącej wyboru kierunku dalszego kształcenia lub innej drogi życiowej. Większość uczniów szkół ponadgimnazjalnych w badaniu CBOS (54%) planuje kontynuować naukę na wybranym kierunku studiów (48%) lub w szkole policealnej (6%). Warto podkreślić, że według CBOS aż o 11 punktów w stosunku do roku 2008 spadł odsetek osób planujących kontynuację edukacji w szkole wyższej. Wzrósł natomiast odsetek osób planujących wyjazd za granicę.

Młodzież, która decyduje się na kontynuację nauki na studiach wyższych kieruje się różnymi motywami przy podjęciu decyzji o wyborze uczelni, a *de facto* o wyborze kierunku studiów na danej uczelni. W badaniach przeprowadzonych przez Uniwersytet Jagielloński (wykres 4) na pierwszym miejscu, jeśli chodzi o powód, dla którego uczniowie szkół ponadgimnazjalnych wybierają szkołę wyższą, znalazła się oferta programowa uczelni, rozumiana jako oferta studiów na danym kierunku. Wybór kierunku kształcenia jest na ogół pierwszym wyborem, na drugim miejscu jest decyzja o uczelni, w której planuje się studiować wybrany kierunek.

Wykres 3. Plany życiowe młodzieży szkół ponadgimnazjalnych po ukończeniu szkoły

Źródło: *Młodzież 2010* [15].

Na kolejnych miejscach znalazły się osobiste ambicje, łatwość znalezienia pracy oraz zainteresowania i pasja. Podobnie, kandydaci do WWSI (2012), jako najważniejsze powody wyboru informatyki, jako kierunku kształcenia, podawali interesującą ofertę studiów, perspektywy uzyskania pracy po studiach oraz referencje wystawiane przez znajomych, studentów oraz absolwentów.

Wyniki badań przeprowadzone w ramach projektu *Mechanizmy decyzyjne ludzi młodych przy wyborze kierunków kształcenia* wśród studentów ostatnich lat studiów inżynierskich, licencjackich i magisterskich w szkołach wyższych publicznych i niepublicznych województwa lubelskiego potwierdzają te same obserwacje, jednak z perspektywy o kilka lat późniejszej.

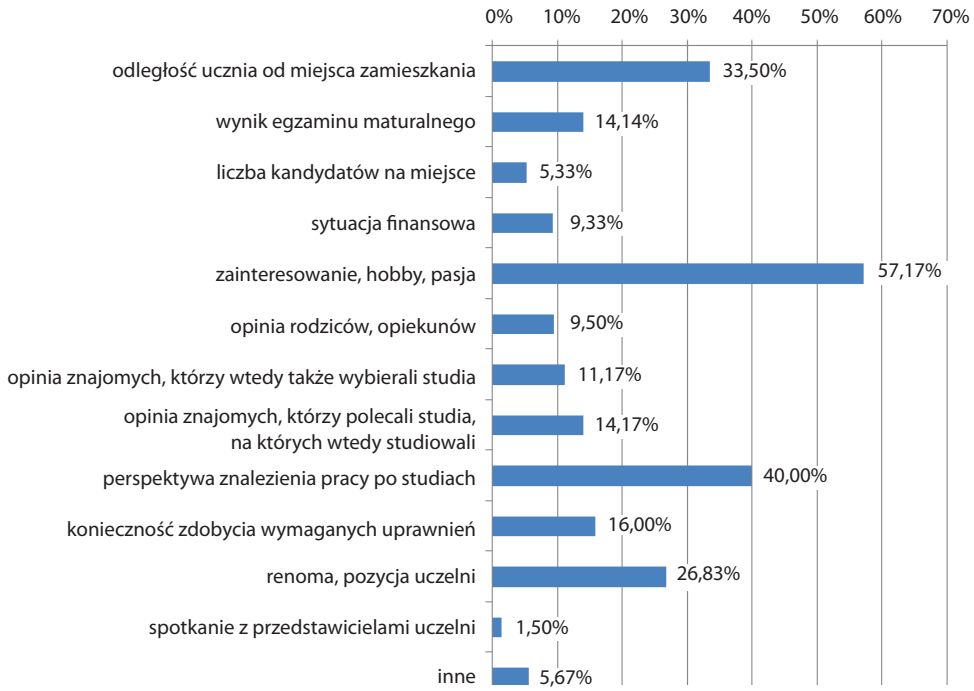
Wykres 4. Motywy wyboru miejsca (kierunku) studiów

Źródło: *Raport z badań preferencji licealistów* [24].

W stosunku do motywów wyboru studiów deklarowanych przez licealistów, nie obserwujemy istotnych zmian. Ankietowani na pierwszym miejscu zdecydowanie wskazują zainteresowania, hobby i pasję jako najważniejsze czynniki

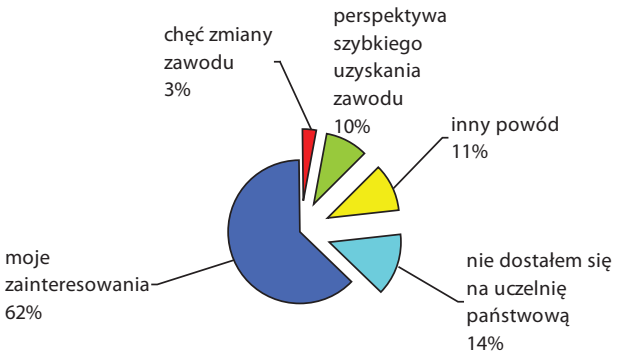
decydujące o wyborze kierunku kształcenia. Na drugim miejscu lokuje się perspektywa znalezienia pracy po studiach. Ważnym powodem wyboru kierunku studiów (uczelni) jest też odległość szkoły wyższej od miejsca zamieszkania.

I wreszcie spojrzenie na problem wyboru kierunku studiów z perspektywy absolwentów. W badaniu losów zawodowych absolwentów WWSI zadano pytanie, co skłoniło ich do podjęcia studiów na kierunku informatyka w WWSI. Motywy wyboru uczelni przedstawiono na wykresie 6. Zdecydowana większość ankietowanych (62%) wybrała WWSI i informatykę – jako kierunek studiów, aby rozwijać swoje zainteresowania. Tylko część osób (14%) podkreśla, że nie dostała się na uczelnie państwowe. Natomiast 10% absolwentów podjęło decyzję, ze względu na perspektywę szybkiego uzyskania zawodu. 3% absolwentów wybrało WWSI z powodu chęci zmiany zawodu, 14% absolwentów podaje inny powód wyboru uczelni, np.: brak egzaminów wstępnych, większy szacunek do studenta niż na uczelni państwowej, tytuł inżyniera, atmosfera uczelni, bliskość do centrum, stypendium za wyniki w nauce, kontynuacja, interesujący program studiów, namowa kolegi, dobra opinia znajomych.



Wykres 5. Odsetek studentów wskazujących poszczególne czynniki, jako istotnie wpływające na wybór kierunku kształcenia

Źródło: *Mechanizmy decyzyjne ludzi młodych przy wyborze kierunków kształcenia* [14].

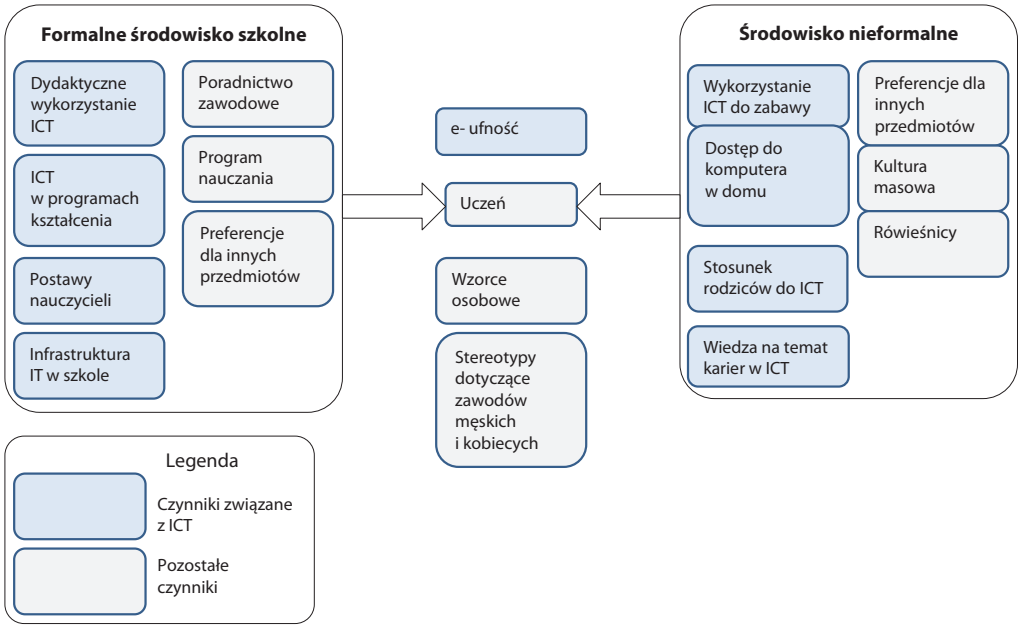


Wykres 6. Motywy wyboru informatyki w WWSI jako kierunku studiów

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

Niezależnie od tego, na jakim etapie życia podejmowane są decyzje, jako najważniejszy powód wyboru kierunku studiowania ankietowani podają osobiste zainteresowania i pasje. W związku z tym, jeśli chcemy zwiększyć zainteresowanie wyborem zawodu informatyka, powinniśmy tworzyć takie formalne (szkolne) środowisko kształcenia i rozwoju oraz oddziaływać w taki sposób na środowisko nieformalne (pozaszkolne), aby wybór ICT, jako kierunku kształcenia, był u większości uczniów rezultatem autentycznych zainteresowań informatyką. W takiej sytuacji wybór zawodu zwiększa szanse na satysfakcję z jego wykonywania.

Spektrum czynników wpływających na wybór ścieżki kariery w ICT obrazują badania przeprowadzone przez European Schoolnet. Głównym celem badań było zdiagnozowanie przyczyn, dla których studia i kariera w ICT nie są postrzegane jako atrakcyjne dla kobiet – uczennic szkół średnich – co przejawia się w znacznie niższym wskaźniku udziału kobiet w studiach ICT w stosunku do mężczyzn oraz konsekwentnie w późniejszej nadreprezentatywności mężczyzn w zawodzie informatyka. Poruszając ważny problem dysproporcji płci w tym zawodzie, autorzy badania konstatują, że na wybór ICT, jako kierunku studiów oraz przyszłego zawodu przez uczniów szkół średnich, zarówno dziewcząt, jak i chłopców, podobnie jak to ma miejsce w przypadku innych zawodów, wpływa wiele czynników, które oddziałują poprzez formalne środowisko szkolne i środowisko nieformalne. Cześć z nich ma bezpośredni związek z technologią informacyjno-komunikacyjną , część w sposób pośredni wpływa na zainteresowanie studiami i karierą w ICT. Na wykresie 7 zostały przedstawione formalne i nieformalne czynniki wpływające na wybór ICT przez uczniów szkół średnich, jako kierunku studiów oraz ścieżki kariery zawodowej.



Wykres 7. Formalne i nieformalne czynniki wpływające na wybór ICT, jako kierunku studiów oraz kariery zawodowej przez uczniów szkół średnich

Źródło: *Why are girls still not attracted to ICT studies and careers*? [5].

Prezentowany na wykresie zestaw czynników jest próbą zidentyfikowania obszarów, które w istotny sposób wpływają na procesy decyzyjne związane z wyborem ICT jako kierunku kształcenia na poziomie szkoły ponadgimnazjalnej. Nie jest to z pewnością zestaw kompletny. Z perspektywy motywów, jakie podają uczniowie, którzy wybierają ICT jako przyszły zawód, dwa czynniki wydają się być szczególnie ważne. Pierwszy to poradnictwo zawodowe, które jest pomocne w określeniu własnych predyspozycji i zainteresowań oraz próbie przypisania ich do konkretnych ścieżek zawodowych. To właśnie poprzez poradnictwo zawodowe można wspomóc decyzje odnośnie wyboru ICT jako zawodu lub wyłącznie jako narzędzia pomocnego w wykonywaniu innych zawodów. Drugi czynnik wpływający na wybór przyszłego zawodu, to wszelkie działania, których celem jest identyfikacja i rozbudzenie własnych zainteresowań uczniów, zarówno w ramach formalnej edukacji, jak i poprzez działania nieformalne.

Decyzje o wyborze kierunku kształcenia po ukończeniu szkoły ponadgimnazjalnej ujawniają się w danych dotyczących rekrutacji na studia wyższe. Przedstawiono je w tabeli 3 najpopularniejszych kierunków studiów w latach 2009-2012.

Tabela 3.
Najpopularniejsze kierunki studiów w latach 2009-2012 na studiach stacjonarnych pierwszego stopnia i jednolitych studiach magisterskich według liczby zgłoszeń kandydatów

Kierunek kształcenia	2011/2012	2010/2011	2009/2010
Budownictwo	29 888	30 944 (2)	24 637 (4)
Zarządzanie	28 608	37 743 (1)	35 388 (1)
Informatyka	27 625	25 435 (5)	24 055 (5)
Pedagogika	25 839	30 414 (3)	33 094 (2)
Prawo	24 581	26 943 (4)	26 581 (3)
Ekonomia	21 523	24 539 (6)	22 025 (6)
Finanse i rachunkowość	19 998	19 977 (7)	15 418 (12)
Inżynieria środowiska	19 330	19 370 (8)	17 723 (10)
Zarządzanie i inżynieria produkcji	16 622	16 806 (12)	13 996 (15)
Mechanika i budowa maszyn	15 868	15 192 (14)	12 181 (20)
Administracja	15 592	19 255 (9)	21 565 (7)
Psychologia	15 562	19021 (10)	20 293 (8)
Gospodarka przestrzenna	14 779	13087 (19)	12 162 (21)
Automatyka i robotyka	14 252	14207 (15)	12 378 (19)
Turystyka i rekreacja	13 587	15339 (13)	18 997 (9)
Geodezja i kartografia	13 527	13857 (16)	11 702 (23)
Biotechnologia	13 044	13836 (17)	13 451 (17)
Filologia angielska	12 942	17529 (11)	16 484 (11)

Źródło: Dane Ministerstwa Nauki i Szkolnictwa Wyższego [3].

Porównanie popularności poszczególnych kierunków studiów w latach 2011/2012, 2010/2011 i 2009/2010 wskazuje, że maleje zainteresowanie kierunkami: pedagogika, zarządzanie, prawo i administracja, ale ciągle jest ono bardzo wysokie. Rosnącym zainteresowaniem cieszy się informatyka, która jako jedyna obok kierunku gospodarka przestrzenna odnotowuje systematyczny wzrost liczby zgłoszeń kandydatów w ciągu ostatnich trzech lat.

5. Edukacja informatyczna

Przez edukację informatyczną dla potrzeb niniejszego artykułu rozumiemy ścieżki edukacyjne, umożliwiające zdobycie wiedzy i umiejętności niezbędnych do wykonywania zawodu informatyka oraz uzyskania formalnych kwalifikacji

zawodowych, zatem rozumienie edukacji informatycznej jest tu zawężone do wymiaru zawodowego. Kompetencje informatyczne są zdobywane przez ucznia już od I etapu edukacyjnego w ramach zajęć komputerowych w klasach I-III szkoły podstawowej, kontynuowanych na II etapie w klasach IV-VI. Na III etapie edukacyjnym w gimnazjum wiedza i umiejętności informatyczne są kształcone w ramach przedmiotu informatyka. IV etap edukacyjny jest początkiem wprowadzania zawodowego wymiaru edukacji informatycznej – w technikach pojawia się specjalizacja w zawodzie technik informatyk, a w szkołach ogólnokształcących uczniom szczególnie zainteresowanym informatyką są oferowane zajęcia informatyczne w zakresie rozszerzonej podstawy programowej. To właśnie z nich najczęściej rekrutują się w przyszłości informatycy-teoretycy oraz uniwersyteccy nauczyciele informatyki i naukowcy prowadzący badania informatyczne i w dziedzinach pokrewnych. Od roku 2012 edukacja informatyczna w zakresie podstawowym obejmuje wszystkich uczniów w szkołach ponadgimnazjalnych w ramach przedmiotu informatyka, który zastąpił przedmiot technologia informacyjna. Celem tego przedmiotu jest informatyczne przygotowanie do życia i funkcjonowania w społeczeństwie wiedzy (informacyjnym).

Tak więc formalne kształcenie zawodowe w specjalnościach informatycznych jest prowadzone w szkołach ponadgimnazjalnych (np. w technikach), na specjalistycznych kursach lub podczas studiów na kierunkach informatycznych w uniwersytetach, na politechnikach i w innych uczelniach, takich jak WWSI.

Edukacja informatyczna w szkołach jest szerzej omówiona w rozdziale *Komputer – obiekt i narzędzie edukacji. Poznawcze walory informatyki i technologii informacyjno-komunikacyjnej.*

Kolejna tabela przedstawia cele edukacyjne zajęć komputerowych w szkole podstawowej oraz przedmiotu informatyka na poziomie gimnazjum, przedmiotu informatyka i zawodu technik informatyk na poziomie szkoły ponadgimnazjalnej oraz standardy kształcenia dla studiów I i II stopnia na kierunku informatyka.

Tabela 4.
Cele edukacji informatycznej oraz oczekiwane kwalifikacje na kolejnych etapach edukacyjnych

I etap edukacyjny **zajęcia komputerowe, klasy I-III szkoły podstawowej**
Cele kształcenia • umiejętność obsługi komputera • posługiwanie się wybranymi programami i grami edukacyjnymi, rozwijanie swoich zainteresowań, korzystanie z opcji w programach • wyszukiwanie i korzystanie z informacji

<table>
<tr><td colspan="2">• tworzenie tekstów i rysunków
• znajomość zagrożeń wynikających z korzystania z komputera, Internetu i multimediów</td></tr>
<tr><td colspan="2">II etap edukacyjny
zajęcia komputerowe, klasy IV-VI szkoły podstawowej</td></tr>
<tr><td colspan="2">Cele kształcenia – wymagania ogólne
• bezpieczne posługiwanie się komputerem i jego oprogramowaniem; świadomość zagrożeń i ograniczeń związanych z korzystaniem z komputera i Internetu
• komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
• wyszukiwanie i wykorzystywanie informacji z różnych źródeł; opracowywanie za pomocą komputera rysunków, motywów, tekstów, animacji, prezentacji multimedialnych i danych liczbowych
• rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera
• wykorzystywanie komputera do poszerzania wiedzy i umiejętności z różnych dziedzin, a także do rozwijania zainteresowań</td></tr>
<tr><td colspan="2">III etap edukacyjny
informatyka, gimnazjum</td></tr>
<tr><td colspan="2">Cele kształcenia – wymagania ogólne
• bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno – komunikacyjnych
• wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł
• opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych
• rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego
• wykorzystanie komputera oraz programów i gier edukacyjnych do poszerzania wiedzy i umiejętności z różnych dziedzin oraz do rozwijania zainteresowań
• ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki</td></tr>
<tr><td>IV etap edukacyjny – zakres podstawowy</td><td>IV etap edukacyjny – technik informatyk</td></tr>
<tr><td>Cele kształcenia – wymagania ogólne
• bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
• wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych</td><td>Cele kształcenia w zawodzie
Absolwent szkoły kształcącej w zawodzie technik informatyk powinien być przygotowany do wykonywania następujących zadań zawodowych:
• montowania oraz eksploatacji komputera i urządzeń peryferyjnych
• projektowania i wykonywania lokalnych sieci komputerowych, administrowania tymi sieciami
• projektowania baz danych i administrowania bazami danych</td></tr>
</table>

Tabela 4. (cd.)

• rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego • wykorzystanie komputera oraz programów i gier edukacyjnych do poszerzania wiedzy i umiejętności z różnych dziedzin oraz do rozwijania zainteresowań • ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki	• tworzenia stron WWW i aplikacji internetowych, administrowania tymi stronami i aplikacjami
IV etap edukacyjny – zakres rozszerzony Cele kształcenia – wymagania ogólne • bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych • wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych • rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego • wykorzystanie komputera oraz programów i gier edukacyjnych do poszerzania wiedzy i umiejętności z różnych dziedzin oraz do rozwijania zainteresowań • ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki	
studia wyższe studia I stopnia – licencjackie	**studia wyższe studia I stopnia – inżynierskie**
Absolwent studiów licencjackich powinien posiadać wiedzę i umiejętności z zakresu ogólnych zagadnień informatyki. Powinien dobrze rozumieć działanie współczesnych systemów komputerowych oraz posiadać wiedzę z zakresu podstaw informatyki, systemów operacyjnych, sieci komputerowych, baz danych	Absolwent studiów inżynierskich, podobnie jak absolwent studiów licencjackich, powinien posiadać wiedzę i umiejętności z zakresu ogólnych zagadnień informatyki oraz dodatkowo wiedzę i umiejętności techniczne z zakresu systemów informatycznych.

Tabela 4. (cd.)

i inżynierii oprogramowania umożliwiającą aktywny udział w realizacji projektów informatycznych. Powinien także posiadać umiejętności programowania komputerów oraz pracy w zespołach programistycznych. Zdobytą wiedzę i umiejętności powinien umieć wykorzystać w pracy zawodowej z zachowaniem zasad prawnych i etycznych.	Powinien dobrze znać zasady budowy współczesnych komputerów i urządzeń z nimi współpracujących, systemów operacyjnych, sieci komputerowych i baz danych. Powinien posiadać umiejętność programowania komputerów i znać zasady inżynierii oprogramowania w stopniu umożliwiającym efektywną pracę w zespołach programistycznych. Powinien mieć także podstawową wiedzę w zakresie sztucznej inteligencji, grafiki komputerowej i komunikacji człowiek-komputer. Swoją wiedzę i umiejętności powinien umieć wykorzystać w pracy zawodowej z zachowaniem zasad prawnych i etycznych.
studia wyższe studia II stopnia – magisterskie	
Absolwent powinien mieć ogólną wiedzę informatyczną przynajmniej w zakresie wszystkich treści podstawowych i kierunkowych właściwych dla studiów licencjackich na kierunku informatyka oraz wykazywać biegłością w wybranej specjalności. Powinien posiadać wiedzę i umiejętności pozwalające na rozwiązywanie problemów informatycznych – również w niestandardowych sytuacjach – a także umieć wydawać opinie na podstawie niekompletnych lub ograniczonych informacji z zachowaniem zasad prawnych i etycznych. Powinien umieć dyskutować na tematy informatyczne zarówno ze specjalistami, jak i niespecjalistami, a także kierować pracą zespołów. Absolwent powinien posiadać umiejętności umożliwiające podjęcie pracy w firmach informatycznych, w administracji państwowej i samorządowej oraz być przygotowanym do pracy w szkolnictwie (po ukończeniu specjalności nauczycielskiej – zgodnie ze standardami kształcenia przygotowującego do wykonywania zawodu nauczyciela). Absolwent powinien mieć wpojone nawyki ustawicznego kształcenia i rozwoju zawodowego oraz być przygotowany do podejmowania wyzwań badawczych i podjęcia studiów trzeciego stopnia (doktoranckich).	

Źródło: Podstawy programowe Ministerstwa Edukacji Narodowej oraz standardy kształcenia Ministerstwa Nauki i Szkolnictwa Wyższego [24, 27, 28].

Zgodnie z rozporządzeniem ministra nauki i szkolnictwa wyższego od roku akademickiego 2012/2013 szkoły wyższe są zobowiązane wdrożyć system Krajowych Ram Kwalifikacyjnych, zgodnie z którymi dla poszczególnych kierunków kształcenia określane powinny być cele kształcenia, jak dotychczas, a także efekty kształcenia, co jest *novum* w stosunku do minionego okresu. W szczególności, obok efektów będących rezultatem realizacji celów określonych w obowiązujących do tej pory standardach, uczelnie precyzują efekty kształcenia, które obej-

mują wiedzę, umiejętności i kompetencje społeczne uzyskane przez studentów. Uczelnie mogą obecnie w ramach kierunku realizować ogólnoakademicki lub praktyczny profil kształcenia. Ten drugi powinien być realizowany w ścisłym powiązaniu ze środowiskiem pracy typowym dla absolwenta danego kierunku lub programu studiów.

Szkoły wyższe na studiach I/II stopnia oraz na studiach podyplomowych, w odpowiedzi na zapotrzebowanie rynku pracy oferują obok uniwersalnego zawodowego wykształcenia informatycznego węższe specjalizacje dziedzinowe. Studia podyplomowe są okazją i miejscem do uzupełnienia oraz aktualizacji wiedzy zdobytej na studiach I i/lub II stopnia. Kompetencje zdobyte w czasie studiów podyplomowych mogą być dla wielu osób również okazją do przekwalifikowania zawodowego. Przykładowe specjalności studiów informatycznych przedstawiono w tabeli.

Tabela 5.
Specjalizacje oferowane studentom na studiach na kierunku informatyka I i II stopnia oraz na studiach podyplomowych

Studia I stopnia	Studia drugiego stopnia	Studia podyplomowe
Inżynieria oprogramowania Inżynieria baz danych Inżynieria multimediów Inżynieria sieci teleinformatycznych Inżynieria Internetu Inżynieria bezpieczeństwa systemów informatycznych Inżynieria systemów mobilnych	Systemy i sieci teleinformatyczne Informatyczne technologie zarządzania Zarządzanie projektami	Inżynieria systemów przetwarzających w chmurze Internetowe aplikacje bazodanowe Zarządzanie projektami informatycznymi Systemy i sieci teleinformatyczne Bezpieczeństwo systemów i sieci komputerowych Technologie Internetowe w zastosowaniach Business Intelligence

Źródło: Warszawska Wyższa Szkoła Informatyki, www.wwsi.edu.pl.

Poza przedstawionymi powyżej ścieżkami edukacyjnymi, formalne kwalifikacje do wykonywania zawodu informatyka można również uzyskać w trakcie certyfikowanych przez firmy szkoleń, które najczęściej mają charakter szkoleń związanych z wybranymi produktami informatycznymi. Do najbardziej popularnych certyfikatów informatycznych należą: Microsoft Certified IT Professional credential (MCITP), Microsoft Certified Technology Specialist (MCTS), Security + (CompTIA), Microsoft Certified Professional Developer (MCPD), Cisco Certified Internetwork Expert (CCNA), A+ (CompTIA), Project Management

Professional (PMI), Microsoft Certified Systems Engineer/ Microsoft Certified Systems Administrator (MCSE/MCSA), Certified Information Systems Security Professional (CISSP), Linux +.

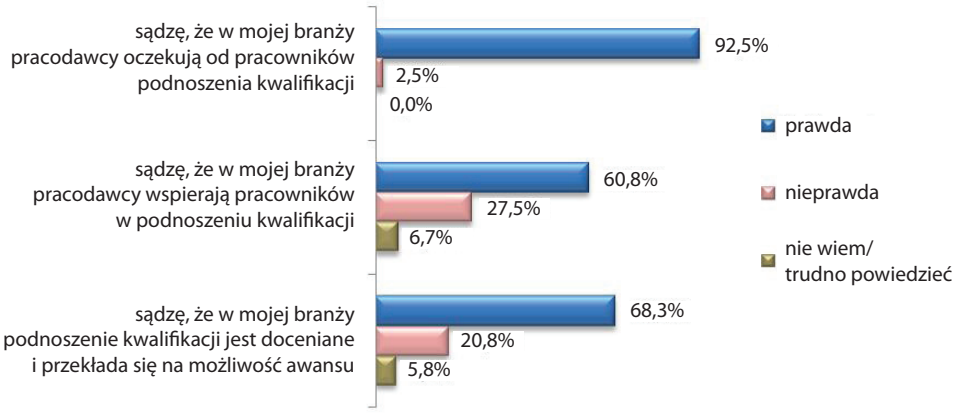
Dynamiczny rozwój technologii informatycznych sprawia, że zawodowa edukacja informatyczna ma zdecydowanie charakter *lifelong learning* – kształcenia się przez całe życie. W badaniach absolwentów studiów podyplomowych WWSI, których uczestnicy legitymowali się w ponad 70% stażem zawodowym powyżej 5 lat (w tym 43% stażem powyżej 10 lat) zdecydowana większość ankietowanych jako przyczynę podjęcia specjalistycznych studiów podyplomowych wskazała potrzebę rozwoju zawodowego oraz chęć poszerzenia i uzupełnienia wiedzy (odpowiednio 85,6% oraz 70,1%).

Wykres 8. Motywy podjęcia kształcenia na specjalistycznych studiach podyplomowych

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

Wyniki badań pokazują ponadto, że pomimo upływu czasu od momentu rozpoczęcia pracy zawodowej nadal w decyzjach dotyczących podjęcia ewentualnego kształcenia, podobnie jak to miało miejsce przy wyborze zawodu, istotną rolę odgrywają ambicje i zainteresowania informatyków. Blisko połowa ankietowanych wymienia je jako ważny motyw rozpoczęcia kształcenia na studiach podyplomowych. Kształceniu ustawicznemu informatyków niewątpliwie sprzyja,

a nawet wymusza je stosunek do tej kwestii pracodawców. W opinii ankietowanych studentów i absolwentów studiów podyplomowych pracodawcy oczekują od pracowników podnoszenia kwalifikacji.



Wykres 9. Stosunek pracodawców do kwestii podnoszenia kwalifikacji przez pracowników ICT

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

Podnoszenie kwalifikacji przez pracowników ICT jest w opinii ankietowanych i ich pracodawców koniecznością i przekłada się na poprawę efektywności działania firm oraz na możliwości awansu zawodowego pracowników.

W związku z nieustannym rozwojem technologii ICT, informatyka jako dziedzina stale poszerza się o nowe obszary, które wymagają nowych specjalistycznych kwalifikacji. Kształcenie ustawiczne jest w tej sytuacji jak w żadnym innym zawodzie *sine qua non* w pracy każdego informatyka.

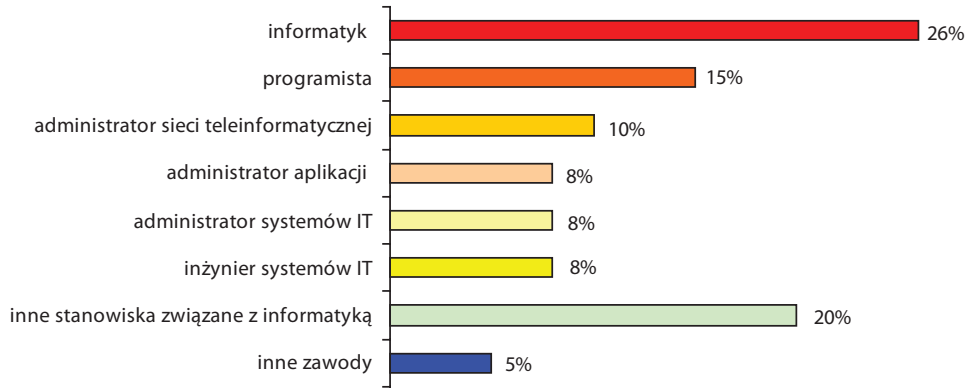
6. Zawód Informatyk – specjalizacje, stanowiska pracy, wynagrodzenia

Ogólnie zawody (specjalności) informatyczne można podzielić na trzy grupy zawodów: z obszaru projektowania systemów ICT, z obszaru tworzenia systemów ICT oraz te, które związane z konserwacją i utrzymaniem systemów informatycznych w ruchu. Do pierwszej grupy zaliczają się pracownicy naukowi instytutów badawczych i szkół wyższych, konsultanci ICT, kierownicy ds. zarządzania informacją, analitycy i architekci systemowi, managerowie ds. e-biznesu, e-handlu, kierownicy ds. *business intelligence*. W drugiej grupie można wymienić specjalistów z zakresu inżynierii oprogramowania i aplikacji, projektantów serwisów internetowych, integratorów systemów. Do trzeciej grupy

zaliczają się specjaliści np.: ds. bezpieczeństwa systemów, administratorzy sieci, administratorzy baz danych.

W badaniu losów zawodowych absolwentów WWSI, jako najczęściej wykonywany zawód ankietowani absolwenci wymieniają zawody z trzeciego obszaru (utrzymanie wewnętrznej infrastruktury ICT):

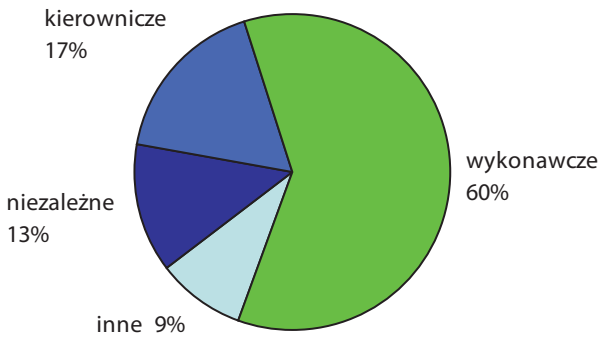
- informatyk (26% odpowiedzi);
- programista (15% odpowiedzi);
- administrator sieci teleinformatycznej (10% odpowiedzi);
- administrator aplikacji (8% odpowiedzi);
- administrator systemów IT (8% odpowiedzi);
- inżynier systemów IT (8% odpowiedzi).



Wykres 10. Jaki zawód wykonują absolwenci kierunku informatyka?

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

W tym samym badaniu 60% respondentów odpowiedziało, że są zatrudnieni na stanowiskach wykonawczych, 17% badanych absolwentów zajmuje stanowiska kierownicze różnego szczebla (np.: dyrektor biura informatyki i strategii IT, dyrektor departamentu telekomunikacji, dyrektor ds. realizacji, kierownik działu serwisu komputerowego, kierownik ds. systemów i baz danych, kierownik projektu, kierownik oddziału, kierownik sekcji, kierownik techniczny, lider zespołu IT), 13% ankietowanych absolwentów określa swoje stanowiska jako niezależne (np. właściciel, *product manager*, *IT manager*, *operations manager*), a 9% ankietowanych zalicza swoje stanowiska do kategorii inne (np. specjalista, starszy specjalista, starszy inżynier infrastruktury, senior, junior szkoleń, ekspert IT, architekt IT).



Wykres 11. Stanowiska, na których są zatrudnieni absolwenci

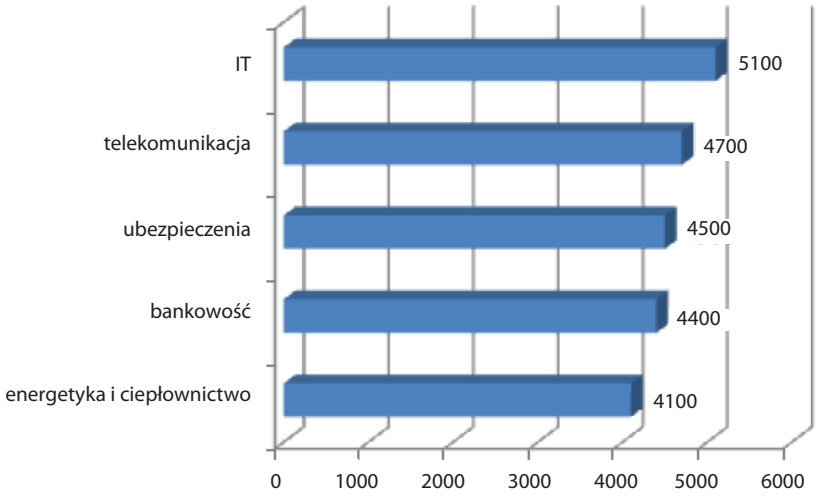
Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

Nazwy stanowisk, na których są zatrudniani zawodowi informatycy pokrywają się z obszarami specjalizacji lub są pochodną funkcji i obowiązków informatyka w firmach. Według danych z badania losów zawodowych absolwentów Warszawskiej Wyższej Szkoły Informatyki respondenci WWSI są zatrudnieni na następujących stanowiskach:

- administrator bezpieczeństwa informacji inżynier wsparcia technicznego ds. produktów IT
- administrator aplikacji kierownik ds. systemów i baz danych
- administrator baz danych kierownik ds. systemów i baz danych
- administrator (utrzymanie systemu) kierownik działu serwisu komputerowego
- administrator i projektant aplikacji sieciowych kierownik oddziału
- administrator sieci LAN kierownik projektu
- administrator sieci teleinformatycznej kierownik sekcji
- administrator systemów IT kierownik techniczny
- analityk systemowy konsultant informatyczny
- analityk systemowy i biznesowy konsultant systemu SAP
- analityk-programista koordynator ds. wsparcia aplikacji
- doradztwo informatycznekoordynator grupy użytkowników IT
- dyrektor biura informatyki i strategii IT lider zespołu IT
- dyrektor departamentu telekomunikacji nauczyciel informatyki
- dyrektor ds. realizacji operator systemów informatycznych
- grafik komputerowy programista
- grafik, webmaster projektant hurtowni danych
- informatyk serwisant

- informatyk – grafik specjalista ds. e-commerce
- informatyk – administrator aplikacji specjalista ds. informatyzacji
- informatyk – administrator baz danych specjalista ds. IT
- informatyk – analityk specjalista ds. komutacji
- informatyk – IT leader CEspecjalista ds. wdrożeń oprogramowania
- informatyk wdrożeniowiec specjalista ds. wsparcia infrastruktury IT
- informatyk web designer specjalista LAN/SAN Backup
- informatyk/elektronik specjalista zastosowań informatyki
- informatyk – administrator sieci starszy specjalista ds. zasobów sieciowych
- internet developer stażysta service academy
- inżynier lokalizacji oprogramowania tester oprogramowania
- inżynier serwisutrener/programista/architekt
- inżynier sieciowy webdeveloper
- inżynier systemowy webmaster
- inżynier systemów Audio Wideo
- żołnierz-informatyk

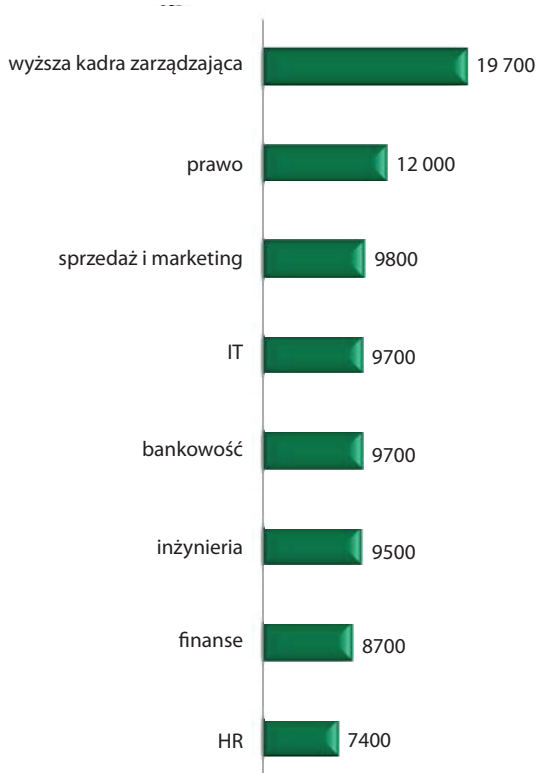
Ważnym argumentem przy wyborze zawodu informatyka jest wysokość średniego wynagrodzenia w branży ICT. Według badań wynagrodzeń przeprowadzonych przez firmę Sedlak & Sedlak, jest ono najwyższe wśród badanych branż i blisko dwukrotnie wyższe niż w branżach o najniższej średniej wynagrodzeń (wykres 12).



Wykres 12. Wynagrodzenia informatyków na tle innych branż

Źródło: *Ogólnopolskie badanie wynagrodzeń 2010* [21].

Również na wysokim czwartym miejscu lokują się wynagrodzenia doświadczonych specjalistów ICT, ustępując wynagrodzeniom wyższej kadry zarządzającej, prawnikom oraz specjalistom ds. sprzedaży i marketingu (wykres 13).



Wykres 13. Średnie wynagrodzenie doświadczonych specjalistów i menadżerów w Polsce wg dyscyplin

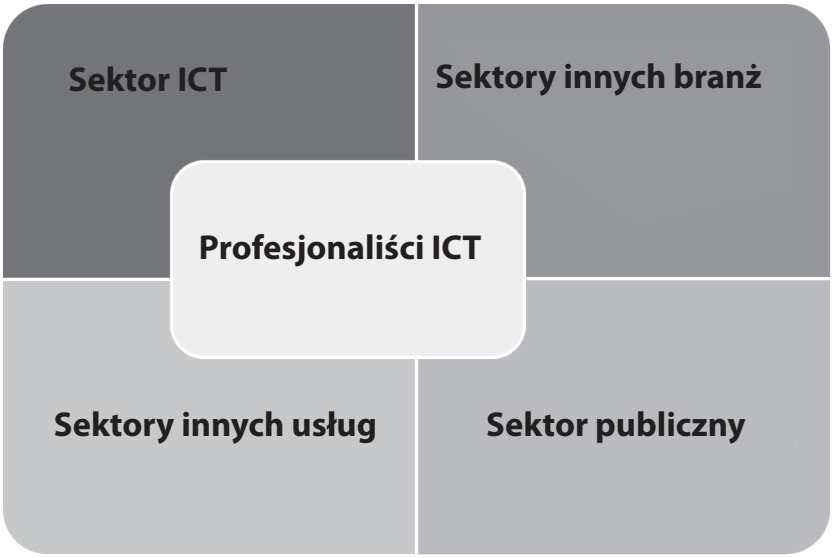
Źródło: *Wynagrodzenia specjalistów i managerów w Polsce* [35].

7. Rynek pracy ICT, stan aktualny i perspektywy

W branży ICT w krajach OECD zatrudnionych jest blisko 16 milionów pracowników (dane z 2008 roku). To blisko 3 do 4% zatrudnionych ogółem w gospodarkach krajów rozwiniętych. W miarę jak proces cyfryzacji poszerza się na nowe obszary życia społeczno-gospodarczego, udział ten rośnie rocznie średnio o 1,2%. Dodatkowo, w zawodach korzystających intensywnie z technologii ICT pracuje kolejne 20% wszystkich zatrudnionych w krajach OECD [3]. W Unii Europejskiej w branży ICT jest zatrudnionych 5,8 miliona osób, liczba zatrudnionych w euro-

pejskiej branży teleinformatycznej podwoiła się od roku 2000 [20]. Także w naszym kraju obserwujemy stały wzrost zatrudnienia w sektorze nowych technologii. W Polsce liczba zatrudnionych bezpośrednio w branży ICT w roku 2010 wynosiła 162 tysiące osób i była o 8% wyższa niż w roku 2007 [29].

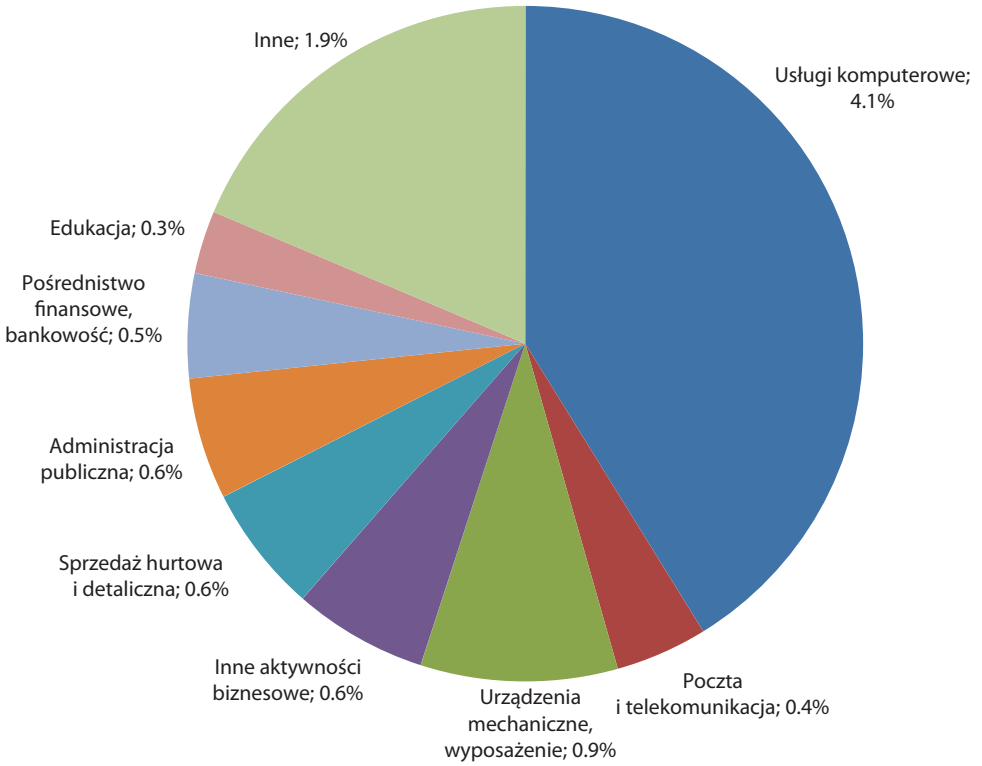
Informatycy są zatrudniani w różnych sektorach gospodarki, tworząc trzon pracowników zatrudnionych w branży ICT. Z punktu widzenia kompetencji związanych z tą technologią, jakimi posługują się pracownicy w większości branż, zasadniczy jest podział na specjalistów ICT, czyli tych, którzy rozwijają, obsługują i konserwują systemy ICT. Wiedza i kompetencje ICT są podstawowym wyznacznikiem zakresu ich czynności zawodowych. Drugą grupą są użytkownicy systemów technologii informacyjno-komunikacyjnej. Informatyka jest dla nich często zaawansowanym, ale jedynie narzędziem pracy. Podstawowy zakres ich czynności zawodowych jest związany z innymi niż informatyka obszarami wiedzy oraz z inną ścieżką kariery zawodowej.



Rysunek 2. Informatycy w gospodarce

Źródło: *Monitoring e-skills demand and supply in Europe* [32].

Procentowy udział zatrudnienia informatyków w różnych branżach gospodarki w krajach Unii Europejskiej przedstawiono na wykresie 14.

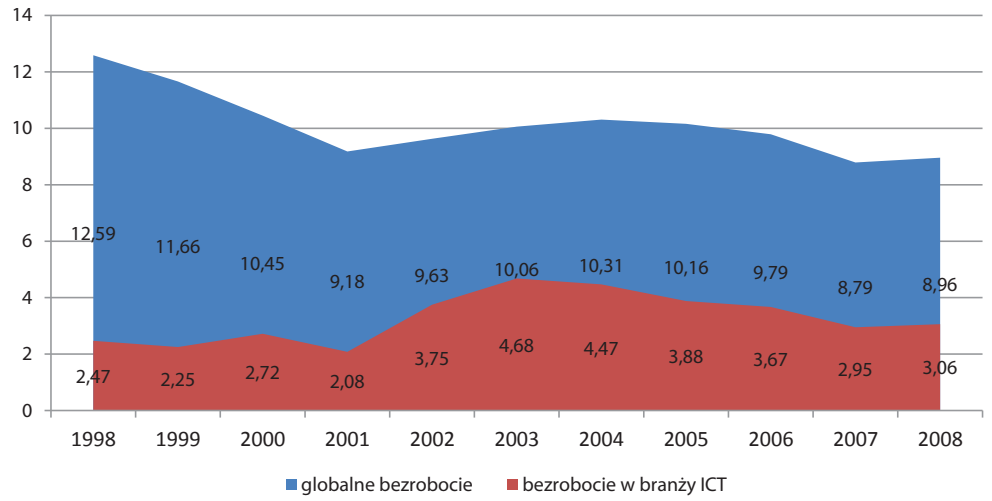


Wykres 14. Odsetek informatyków zatrudnionych w gospodarce według branż w krajach Unii Europejskiej

Źródło: *Monitoring e-skills demand and supply in Europe* [31].

Według wiceprzewodniczącej Komisji Europejskiej odpowiedzialnej za Agendę Cyfrową w Europie, Neelie Kroes, zwiększający się niedobór specjalistów w obszarze ITC w Europie przyczyni się do braku niemal 700 000 specjalistów do 2015 roku. Warto podkreślić, że wartość polskiego rynku informatycznego według badania Computerworld TOP200 wyniosła w 2011 roku 31,3 mld zł. W porównaniu z sytuacją z roku 2010 przychody wzrosły o 2,3 mld zł, czyli o 8,3% [11].

Dwa kolejne wskaźniki – wskaźnik liczby bezrobotnych oraz wskaźnik wakatów – pozwalające ocenić branżę ICT pod kątem ewentualnego wyboru własnej kariery zawodowej. Wskaźnik bezrobocia wśród specjalistów ICT w 15 krajach Unii Europejskiej wahał się w latach 1998-2008 pomiędzy 2 a 4% osiągając 3% w 2008 roku. W analogicznym okresie, ogólny średni wskaźnik bezrobocia w tych samych krajach zawierał się w przedziale 9-13%, przyjmując w roku 2008 wartość blisko 10%.



Wykres 15. Odsetek bezrobotnych informatyków oraz odsetek bezrobotnych ogółem w krajach Unii Europejskiej w latach 1998-2008

Źródło: *Monitoring e-skills demand and supply in Europe* [32].

Zatem zagrożenie utratą pracy jest trzykrotnie mniejsze w branży ICT niż średnia w całej gospodarce. „Według polskich danych statystycznych w końcu czerwca 2011 r. wśród zarejestrowanych bezrobotnych absolwentów było: l,7 tys. pedagogów, 1 tys. ekonomistów, ok. 0,4 tys. politologów i po ok. 0,3 tys. socjologów i specjalistów od marketingu i handlu. Wśród studentów to właśnie te kierunki są najbardziej popularne, choć nie od dziś wiadomo, że brakuje ofert pracy dla osób je kończących. A zatem wymienione zawody są zawodami nadwyżkowymi, tzn. liczba ofert pracy zgłaszana do urzędów pracy jest niepo-równanie niższa od liczby zarejestrowanych bezrobotnych w tych zawodach. Dla przykładu w I półroczu 2011 r. zarejestrowało się 8 tys. pedagogów, w tym 2,7 tys. absolwentów, natomiast w urzędach pracy było dla nich zaledwie 220 ofert" [18]. Z podanej przez ministra pracy i polityki społecznej wypowiedzi wynika, że informatycy to zawód, który w praktyce nie pojawia się w statysty-kach bezrobotnych. Również w innej kategorii objętej badaniami rynku pracy – kategorii wakatów – branża ICT prezentuje się optymistycznie. Jako że tech-nologie informacyjne i Internet są uważane współcześnie za główne motory innowacji, wzrostu gospodarczego i zmian społecznych można spodziewać się, że będą w przyszłości generować nowe zawody i nowe miejsca pracy. Technologie *green IT*, aplikacje *smart* czy obliczenia w chmurze (*cloud computing*) będą w najbliższej przyszłości przyczyniać się do powstawania nowych miejsc pracy. Rośnie także zapotrzebowanie na umiejętności związane z rozwojem aplikacji internetowych, takich jak Ajax, PHP, Javascript, Flash.

Jak pokazują wyniki siódmej edycji globalnego badania *Niedobór talentów* [13] (poniższe zestawienie), 37% pracodawców w Polsce nie może znaleźć pracowników. Najtrudniej znaleźć na rynku pracy inżynierów, a na szóstym miejscu w zestawieniu znaleźli się specjaliści ICT.

1. Inżynierowie
2. Wykwalifikowani pracownicy fizyczni
3. Technicy
4. Kierowcy
5. Przedstawiciele handlowi
6. Pracownicy działów IT
7. Szefowie kuchni/kucharze
8. Menedżerowie projektów
9. Operatorzy maszyn
10. Pracownicy finansów

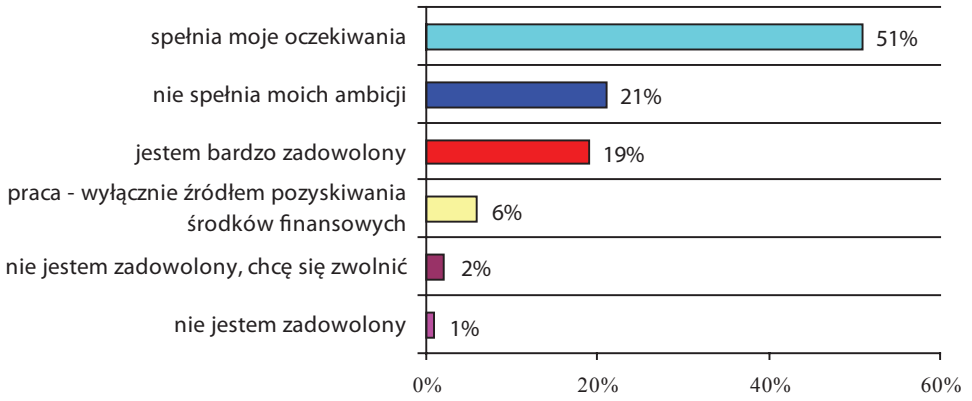
8. Podsumowanie

Powodem, dla którego powinniśmy zdecydować się na karierę zawodową w danej profesji powinny być ponad wszelką wątpliwość zainteresowania i pasje, bo tylko wówczas możemy podsumować swoje życie zawodowe tak jak Thomas Alva Edison, który przyznał:

Nie przepracowałem ani jednego dnia w swoim życiu.
Wszystko, co robiłem, to była przyjemność.

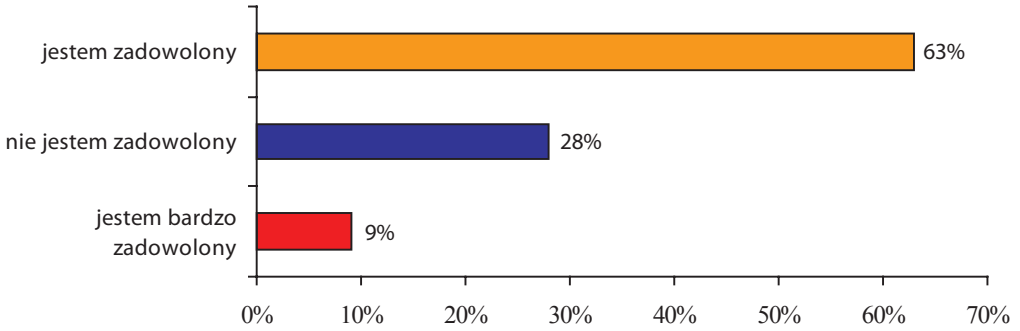
W badaniach losów absolwentów WWSI zdecydowana większość ankietowanych (70%) jest zadowolona z obecnej pracy, 51% absolwentów uważa, że praca zawodowa spełnia ich oczekiwania, natomiast 19% jest z obecnie wykonywanej pracy bardzo zadowolona. Tylko 2% absolwentów nie jest zadowolonych z obecnej pracy, a 6% – traktuje pracę wyłącznie jako źródło pozyskiwania środków finansowych. Wyniki badań pokazują więc, że dla zdecydowanej większości informatyków praca jest źródłem satysfakcji zawodowej.

Przy wyborze zawodu, poza kierowaniem się znajomością własnych predyspozycji i zainteresowań, warto poznać pragmatyczne powody wyboru kierunku kariery zawodowej. Dla zawodu informatyka do najważniejszych z nich trzeba zaliczyć: dobrą sytuację na rynku pracy, który charakteryzuje się nie tylko rosnącym wskaźnikiem zatrudnienia, ale co więcej, sytuacją permanentnego, niezaspokojonego zapotrzebowania na specjalistów ICT, zarówno w Polsce, jak i w innych krajach oraz wyższe na tle innych zawodów średnie wynagrodzenia. Jednym z ważnych powodów zadowolenia z wykonywanej pracy jest właśnie status materialny potwierdzony satysfakcją z wysokości otrzymywanego wynagrodzenia (wykres 17). Prawie 90% absolwentów kierunku informatyka ma wynagrodzenie powyżej średniej krajowej. Większość ankietowanych absolwentów (63%) jest zadowolona z wysokości swoich zarobków. Jednak 28% respondentów uważa, że aktualna płaca nie spełnia ich oczekiwań.



Wykres: 16. Czy absolwenci kierunku informatyka są zadowoleni z wykonywanej pracy?

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

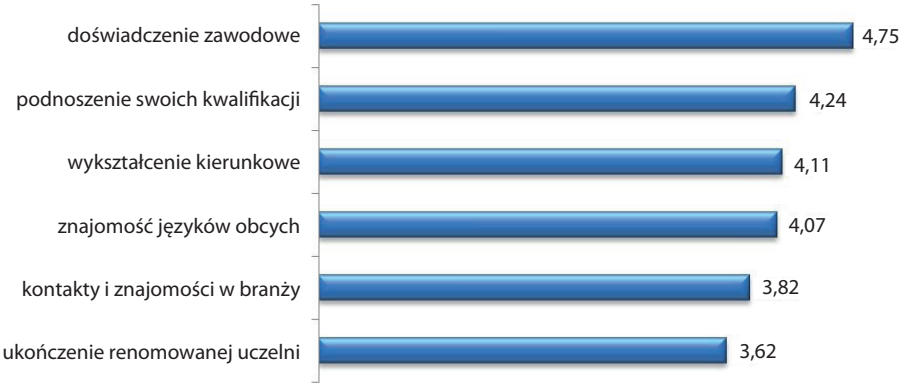


Wykres 17: Czy absolwenci WWSI są zadowoleni z wysokości zarobków?

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [34].

Ważną wskazówką na temat tego, co może przyczynić się do poprawy wskaźnika satysfakcji z płacy oraz z wykonywanej pracy, są informacje o wagach (w skali 1 do 5) przyznanych przez pracowników z pięcioletnim i dłuższym stażem pracy poszczególnym czynnikom, decydującym o osiągnięciu sukcesu w zawodzie informatyka. Odpowiedzi udzielone przez ankietowanych (wykres 18) wskazują jednoznacznie na potrzebę ciągłego rozwoju poprzez zdobywanie doświadczeń zawodowych oraz podnoszenie kwalifikacji, jako główne źródła osiągnięcia sukcesu w zawodzie informatyka. Nowoczesność i innowacyjność zawodu, który

jest synonimem postępu i głównym motorem wzrostu gospodarczego i rozwoju społecznego, stawia wysokie wymagania przed informatykami, często określanymi mianem robotników społeczeństwa wiedzy XXI wieku.



Wykres 18. Czynniki decydujące o osiągnięciu sukcesu w zawodzie informatyka

Źródło: *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki* [33].

Kariera w ICT to dobry wybór dla osób z pasją, które są otwarte na zmiany i kształcenie przez całe życie.

Rysunek 3. Debata „Kariera w IT" z udziałem członków Kolegium Rektorskiego Warszawskiej Wyższej Szkoły Informatyki: Pawła Czajkowskiego, Dyrektora Generalnego Hewlett-Packard Polska, Jacka Murawskiego, Dyrektora Generalnego Microsoft Polska oraz Tomasza Klekowskiego, Dyrektora Generalnego Intel Polska. http://wwsi.edu.pl/pg.php/videoplay/kariery_w_it_z_udzialem_przeds__/214/#video_main

Literatura

1. *Beyond e-readiness, Digital economy rankings 2010*, A report from the Economist Intelligence Unit, written in co-operation with The IBM Institute for Business Value, London 2010, http://www-935.ibm.com/services/us/gbs/bus/pdf/eiu_digital-economy-rankings-2010_final_web.pdf
2. *Cisco Connected Technology World Report*, 2011, http://www.cisco.com/en/US/solutions/ns341/ns525/ns537/ns705/ns1120/CCWTR-Chapter1-Report.pdf
3. Dane Ministerstwa Nauki i Szkolnictwa Wyższego, Informacje o wynikach rekrutacji: http://www.nauka.gov.pl/fileadmin/user_upload/ministerstwo/Aktualnosci/20120118/20120118_WYNIKI_rekrutacji_2011-2012.pdf, http://www.nauka.gov.pl/fileadmin/user_upload/szkolnictwo/Dane_statystyczne_o_szkolnictwie_wyzszym/20110104_WYNIKI_rekrutacji_2010.pdf, http://www.nauka.gov.pl/fileadmin/user_upload/szkolnictwo/Dane_statystyczne_o_szkolnictwie_wyzszym/20100111_WYNIKI_rekrutacji_2009_GS.pdf
4. Digital Agenda Scoreboard 2012, http://ec.europa.eu/information_society/digital-agenda/scoreboard/docs/2012/scoreboard_ict_sector_and_rdi.pdf
5. Gras-Velazquez A., Joyce A., Debry M., *Why are girls still not attracted to ICT studies and careers*? White paper, Brussels 2009, http://blog.eun.org/insightblog/upload/Women_and_ICT_FINAL.pdf
6. *Investment for the Future, Benchmarking IT Industry Competitiveness 2011*, Business Software Alliance, Washington 2011
7. Jegorow D., Niećko I., *Plany młodzieży gimnazjalnej Białegostoku dotyczące dalszego kształcenia i wyboru zawodu,* Raport z badania, Instytut Badań Rynku, Konsumpcji i Koniunktur, Warszawa 2009.
8. Komentarz do podstawy programowej, zajęcia komputerowe I i II etap edukacyjny, informatyka III i IV etap edukacyjny, Podstawa programowa z komentarzami. T.6, Edukacja matematyczna i techniczna w szkole podstawowej, gimnazjum i liceum: matematyka, zajęcia techniczne, zajęcia komputerowe, informatyka, Ministerstwo Edukacji Narodowej, Warszawa 2009
9. *Leading Through Connections*, IBM CEO Study 2012, http://www.brandchannel.com/images/papers/536_IBMGlobalCEOs.PDF
10. *Living in a Hyperconnected World*, The Global Information Technology 2012 Report, Soumitra Dutta, INSEAD, Beñat Bilbao-Osorio, World Economic Forum Editors 2012, http://www3.weforum.org/docs/Global_IT_Report_2012.pdf11
11. Maciejewski A., *Wychodzimy z dołka*, „Computerworld", 25.06.2012, http://www.computerworld.pl/news/383725/Wychodzimy.z.dolka.html
12. *Measuring the Impacts of Information and Communication Technology for Development*, UNCTAD Current Studies on Science, Technology and Innovation, no. 3, United Nations, New York, Geneva 2011, http://unctad.org/en/Docs/dtlstict2011d1_en.pdf
13. *Measuring the Information Society*, International Telecommunication Union, Geneva 2011. http://www.itu.int/net/pressoffice/backgrounders/general/pdf/5.pdf
14. *Mechanizmy decyzyjne młodych ludzi przy wyborze zawodu*, Wojewódzki Urząd Pracy, Lublin 2009
15. *Młodzież 2010*, seria Opinie i Diagnozy Nr 19, CBOS, Warszawa 2011
16. *Mobility in a Networked World*, The Global Information Technology Report 2008–2009, https://members.weforum.org/pdf/gitr/2009/gitr09fullreport.pdf
17. *Niedobór talentów*, Wyniki badania, Grupa Manpower, Warszawa 2012, https://candidate.manpower.com/wps/wcm/connect/8b5ddd004bc170fbb7abbf1abeefe959/Niedobor_talentow_2012_pl.pdf?MOD=AJPERES
18. *Nowoczesne kompetencje IT dla rynku pracy, Studia podyplomowe dla przedsiębiorców i pracowników przedsiębiorstw*, wybrane wyniki badania ewaluacyjnego, WWSI, Warszawa 2012

19. Odpowiedź ministra pracy i polityki społecznej na interpelację nr 66 w sprawie wzrostu bezrobocia wśród osób z wyższym wykształceniem, Władysław Kosiniak-Kamysz – minister pracy i polityki społecznej, 19-12-2011, http://www.sejm.gov.pl/sejm7.nsf/InterpelacjaTresc.xsp?key=442D9758
20. *OECD Information Technology Outlook 2010*, http://www.kigeit.org.pl/FTP/mk/ogolne/IToutlook2010.pdf
21. *Ogólnopolskie badanie wynagrodzeń, Wynagrodzenia na stanowiskach IT*, Sedlak & Sedlak, Warszawa 2010, http://www.wynagrodzenia_na_stanowiskach_it_w_2010_roku_-_podsumowanie.pdf
22. *Perspektywy ludzi młodych na rynku pracy*, Raport końcowy, Wojewódzki Urząd Pracy, Lublin 2011
23. *Plany edukacyjno zawodowe uczniów III klas poznańskich techników*, Centrum Doradztwa Zawodowego dla Młodzieży, Poznań 2011, http://www.cdzdm.pl/sites/default/files/Raport%202011.pdf
24. *Program nauczania dla zawodu technik informatyk nr 351203, Krajowy Ośrodek wspierania Edukacji Zawodowej i Ustawicznej*, Warszawa 2012, http://www.koweziu.edu.pl/programy_nauczania/pliki/351203_technik_informatyk_program_P_2012-05-07.pdf
25. *Raport z badań preferencji licealistów*, Uniwersytet Jagielloński, Kraków 2011
26. *Raport z badania planów edukacyjnych i zawodowych mazowie*ckiej *młodzieży dla Mazowieckiej Izby Rzemiosła i Przedsiębiorczości*, Pretendent, Warszawa 2010
27. Rozporządzenie Ministra Edukacji Narodowej dnia 23 grudnia 2008 r. w sprawie nowej podstawy programowej wychowania przedszkolnego oraz kształcenia ogólnego w poszczególnych typach szkół. Dz.U. z 2009 r. Nr 4, poz. 17
28. Rozporządzenie Ministra Nauki i Szkolnictwa Wyższego w sprawie standardów kształcenia dla kierunku Informatyka, 2007, Dz.U. z dnia 13 września 2007 r.
29. *Społeczeństwo informacyjne w Polsce. Wyniki badań statystycznych z lat 2007-2011*, GUS Warszawa 2012, http://www.stat.gov.pl/cps/rde/xbcr/gus/nts_spolecz_inform_w_polsce_2007-2011.pdf
30. Sysło M.M., Jochemczyk W., *Edukacja informatyczna w nowej podstawie programowej*, OEIiZK, Warszawa 2010, http://www.bc.ore.edu.pl/Content/141/Edukacja+informatyczna+w+nowej+podstawie+programowej+-+Maciej+M.+Sys%C5%82o.pdf
31. Szczucka A., Jelonek M., *Kogo kształcą polskie szkoły*, Raport z badań uczniów szkół ponadgimnazjalnych i analizy kierunków kształcenia, PARP, Warszawa 2011, https://portal.uw.edu.pl/documents/5800711/2a9cf067-44cd-45b5-ab3e-b243328208fb
32. *The evolution of the supply and demand of e-skills in Europe*, Synthesis Report, Monitoring e-skills demand and supply in Europe, Empirica, Bonn 2009, http://www.eskills-monitor.eu/documents/Synthesis%20ReportMeSkills_final.pdf
33. Toffler A., *Zmiana władzy, wiedza, bogactwo i przemoc u progu XXI wieku*, Zysk i S-ka, Poznań 2003
34. Wojciechowska H., Żyławski A., *Losy zawodowe absolwentów studiów I stopnia (inżynierskich) Warszawskiej Wyższej Szkoły Informatyki (lata 2007, 2008, 2009, 2010)*, Raport z badania, WWSI, Warszawa 2011, http://wwsi.edu.pl/upload/large/badanie_losow_absolwentow.pdf
35. *Wynagrodzenia specjalistów i managerów w Polsce*, Raport z badania wynagrodzeń, Antal International, Warszawa 2012, http://www.bpcc.org.pl/att/a1828927-9b6b-4a62-b36d-1a5806aa3b91_wynagrodzenia-specjalistow-i-menedzerow-w-polsce.pdf

Andrzej Żyławski

rektor Warszawskiej Wyższej Szkoły Informatyki, w swoim dorobku zawodowym posiada między innymi wieloletnie kierowanie instytucjami edukacyjnymi kształcącymi profesjonalnych informatyków. W latach 1991-2000 kierował Pomaturalnym Studium Informatycznym Mila College, a od roku 2000 kieruje Warszawską Wyższą Szkołą Informatyki. W tym czasie obie szkoły ukończyło blisko 7000 osób legitymujących się dyplomami technika, inżyniera oraz magistra informatyki. Od samego początku kariery zawodowej wspólnie z Haliną Wojciechowską prowadzi cykliczne badania losów zawodowych absolwentów studiów informatycznych, które stanowią podstawowe źródło doskonalenia skuteczności pracy szkół oraz inspirację dla ich rozwoju. Wybrane wyniki i doświadczenia zebrane w trakcie tych badań zostały wykorzystane w tym rozdziale.

azylawski@wwsi.edu.pl

Maciej M. Sysło

Wydział Matematyki i Informatyki
Uniwersytet Wrocławski
Uniwersytet Mikołaja Kopernika w Toruniu

Historia rachowania – ludzie, idee, maszyny
Historia mechanicznych kalkulatorów

Dla wielu osób informatyka nie ma jeszcze swojej historii. Współczesny komputer elektroniczny jest jednak ukoronowaniem wspólnych wysiłków cywilizacji i pokoleń, rozwijających w ciągu wieków wiele różnych dziedzin nauki i techniki, które kształtowały również sposoby rachowania i konstrukcje urządzeń wspomagających złożone i masowe obliczenia. Od zarania bowiem ludzkości człowiek starał się ułatwiać sobie prowadzenie rachunków i obliczeń, posługując się przy tym różnymi urządzeniami. Tak rodziły się abaki (liczydła), kalkulatory i wreszcie komputery.

Komputer osobisty z początków lat 80. XX wieku można uznać za zwieńczenie wysiłków zarówno tych, których efektem były przeróżne konstrukcje kalkulatorów mechanicznych, przeznaczonych na ogół do osobistego użytku, jak i tych, które skupiały się na budowie komputera o ogólnym i powszechnym przeznaczeniu.

Jeśli nawet uznaje się, że informatyka ma swoją historię, to na ogół niewielką uwagę przywiązuje się do urządzeń mechanicznych. A przecież twórcami pierwszych takich maszyn były nieprzeciętne umysły XVII wieku: John Napier, Blaise Pascal i Gottfried Leibniz. Mechanizmy użyte przez Pascala i Lebniza były stosowane w kalkulatorach mechanicznych do ostatnich dni tych urządzeń. Co więcej, ich rozwój i produkcja doprowadziły do sytuacji w latach 50.-70., w której każdy człowiek potrzebujący takiego urządzenia mógł sobie je sprawić, podobnie jak dzisiaj każdy może mieć komputer osobisty. Na początku lat 70. te piękne mechaniczne cacka powędrowały jednak do lamusa, wyparte przez kalkulatory elektroniczne.

Niniejszy rozdział jest poświęcony najważniejszym osiągnięciom w dziedzinie kalkulatorów mechanicznych i kilku ważnym ideom z okresu ich świetności, które można dzisiaj odnaleźć, w nieco przetworzonej postaci, wśród najważniejszych mechanizmów napędzających rozwój współczesnej technologii i informatyki.

1. Wprowadzenie

Wiele momentów przełomowych w historii doprowadziło do powstania współczesnego komputera, patrz p. 9. Zalicza się do nich: projekty maszyn Charlesa Babbage'a (początek XIX wieku), system tabulacyjny Hermana Holleritha (koniec XIX wieku), prace Claude E. Shannona, dotyczące wykorzystania algebry Boole'a do analizy i syntezy układów przełączających i binarnych (pierwsza połowa XX wieku), komputery Konrada Zuse (połowa XX wieku), fundamentalne dla teorii obliczalności prace Alana Turinga (lata 30. XX wieku), pierwsze komputery elektroniczne – ABC, Colossus, ENIAC, Harvard MARK, EDVAC, IBM 701 (lata II wojny światowej i lata powojenne), wynalazki tranzystora i układu scalonego, rozwój Internetu (druga połowa XX wieku).

Znacznie wcześniej, bo od początku XVII wieku, a na dobrą sprawę od zarania ludzkości, narastało zainteresowanie automatyzacją obliczeń i urządzeniami, które byłyby w stanie usprawnić rachowanie. Pojawiały się pomysły i wynalazki, które miały na celu zbudowanie urządzenia obliczeniowego do indywidualnego użytku. Chociaż pierwsze pomysły były elitarne – Wilhelm G. Schickard zbudował maszynę dla Johannesa Keplera (1623), Blaise Pascal zaprojektował Pascalinę dla swojego ojca poborcy podatkowego (1642), a Gottfried W. Leibniz zbudował „ławę liczącą", trochę w rywalizacji z Pascalem, ale głównie z myślą o realizacji maszyny filozoficznej (1694) – to dalszy rozwój urządzeń do indywidualnych obliczeń i ich produkcja doprowadziły w latach 50-70. XX wieku do sytuacji, w której każdy człowiek potrzebujący takiego urządzenia mógł sobie je sprawić, podobnie jak dzisiaj każdy może mieć komputer osobisty.

I pewnego dnia, gdzieś na początku lat 70., te piękne mechaniczne cacka powędrowały do lamusa. Chociaż od wynalezienia tranzystora (koniec lat 40.) i układu scalonego (początek lat 70.) można się było tego spodziewać, wielu użytkowników mechanicznych kalkulatorów żegnało się z nimi z żalem, mogły bowiem one działać i spełniać swoje zadania jeszcze przez wiele lat, a niektóre z nich wręcz w nieskończoność. Zastąpiły je kalkulatory elektroniczne.

Te dwa nurty w historii informatyki przecięły się na początku lat 80. XX wieku, a efektem tego było pojawienie się komputera osobistego, takiego jak IBM PC czy Apple. Wykorzystano w nich więcej wynalazków z tego głównego nurtu, a z tego znacznie starszego pozostała tylko idea urządzenia osobistego.

Czy z tego drugiego, wydaje się, że mniej znaczącego ciągu rozwoju urządzeń do liczenia, wypływa jakaś lekcja historii? Co pozostało w informatyce po urządzeniach, które przeszły w niepamięć, po ideach i wynalazkach, które zostały

w nich zrealizowane? Czy doskonałością swoich rozwiązań zaprzątają one dzisiaj uwagę jedynie kolekcjonerów?[1]

Nie wszystko jednak umarło i idee oraz wynalazki z okresu przedelektronicznego można jednak znaleźć we współczesnej informatyce, czasem w nieco przetworzonej postaci – wymieńmy ważniejsze z nich: logarytm – miał szansę wynaleźć go Euklides, niemal 1500 lat wcześniej niż zrobił to Napier, kompresja informacji ukryta w alfabecie Morse'a, układ klawiszy na klawiaturze i fonty w edytorach pochodzące od pierwszych maszyn do pisania. Piszemy o tym w p. 7.

A przyszłość informatyki? Kalkulatory mechaniczne i elektryczne oraz suwaki logarytmiczne zostały użyte przy projektowaniu kalkulatorów elektronicznych, te zaś wyparły niemal natychmiast z użycia urządzenia, które posłużyły do ich stworzenia. A jaka nowa technologia zostanie stworzona na dzisiejszych komputerach osobistych i superkomputerach, która wyprze je w przyszłości? Może będzie to nie tylko technologia, a wręcz inny rodzaj inteligencji, konkurującej z inteligencją nierozerwalnie związaną z człowiekiem? Szkoda, że nie znamy odpowiedzi na zatroskanie wielkiego fizyka:

Skąd bierze się różnica
między przeszłością i przyszłością?
Dlaczego pamiętamy przeszłość,
a nie pamiętamy przyszłości?
Stefan W. Hawking, *Krótka historia czasu*

2. Liczby, abaki, algorytmy

2.1. Narodziny liczb

Narodziny liczb trwały bardzo długo, przypuszcza się, że nawet przez 30 tys. lat, patrz [3] i [4]. Historia tego wynalazku jest w dużej części anonimowa, ponieważ liczby i sposoby rachowania są produktem zbiorowej praktyki, w której uczestniczyły, w różnych okresach swojej historii i niezależnie, wszystkie znaczące cywilizacje przeszłości: sumeryjska i babilońska, egipska, grecka, rzymska i żydowska, cywilizacje Chińczyków i Japończyków, Majów, Hindusów i Arabów. Zwieńczeniem intelektualnych zmagań ludzkości jest upowszechnienie się we wszystkich współczesnych cywilizacjach **dziesiętnego systemu pozycyjnego** do zapisywania liczb z jednoczesnym przyjęciem roli **zera jako cyfry i liczby**.

[1] Urządzenia na ilustracjach należą do kolekcji autora. Więcej zdjęć i opisów tych i innych urządzeń (będzie) można znaleźć w witrynie autora http://mmsyslo.pl/.

Znaczącym pomysłem na drodze do obecnie stosowanego systemu liczbowego było wprowadzenie **bazy** i posługiwanie się nią w zapisie liczb. Baza służyła do grupowania jednostek i coraz większych grup tak, by móc zapisywać coraz większe liczby. Zapewne w związku z liczbą palców u obu rąk, często za bazę przyjmowano 10. Wtedy wystarczyło nazwać liczby mniejsze od 10 oraz kolejne potęgi liczby 10, by móc wypowiedzieć lub zapisać każdą liczbę. Popularne były także bazy: 5 (liczba palców u jednej ręki), 20 (liczba wszystkich palców u człowieka), 12 (do dzisiaj mamy *tuzin* i *gros*, czyli dwanaście tuzinów), a także zagadkowa baza 60, stosowana przez Sumerów i Babilończyków w XVIII wieku p.n.e., której ślady pozostały do dzisiaj w naszych rachunkach związanych z czasem i kątami.

Za kolebkę stosowanych obecnie cyfr, zwanych **arabskimi**, w tym zera, i pozycyjnego systemu zapisywania liczb, uważa się powszechnie Indie, gdzie te odkrycia pojawiły się w V – I wieku p.n.e., a osiągnęły swoją dojrzałą postać w V – VI wieku n.e. Pięć wieków (VIII – XIII) to okres świetności nauki w świecie muzułmańskim. Arabowie interesowali się osiągnięciami świata starożytnego, poznali odkrycia Hindusów (*de numero Indorum* – indyjską sztukę rachowania) i sami rozwinęli wiele pomysłów matematycznych. Jednym z czołowych matematyków arabskich był **Muhammad ibn Musa al-Chorezmi** (ok. 787 – ok. 847 n.e.), który przyczynił się rozpowszechnienia dziesiętnego systemu pozycyjnego i metod rachunkowych pochodzenia indyjskiego. Same cyfry pochodzą od Arabów, którzy zamieszkiwali w Afryce północnej i w Hiszpanii. Wprowadził je w Europie papież Sylwester II (**Gerbert d'Aurillac**, ok. 945 – ok. 1003 n.e.), ale upłynęło ponad 200 lat zanim w Europie zaczęto w pełni korzystać z dziesiętnego systemu zapisu liczb i zera oraz z metod rachunkowych pochodzących z Indii. Przysłużył się temu wielce **Fibonacci** (Leonardo z Pizy, ok. 1170 – ok. 1250 n.e.) swoim dziełem *Liber Abaci*.

Niektóre cywilizacje wprowadzały własne systemy oznaczania cyfr i liczb oraz rachowania na nich, często związane z językiem, jakim się posługiwały, patrz rys. 1. Większość tych systemów jest stosowanych do dzisiaj, obok systemu dziesiętnego. Własny system wynaleźli Rzymianie (w V wieku p.n.e) – **cyfry rzymskie** służyły głównie do zapisywania i przechowywania liczb, ale trudniej było z ich pomocą wykonywać rachunki. Stosowane są do dzisiaj, np. do oznaczania dat. Wartość dziesiętna liczby zapisanej w tym systemie jest sumą i/lub różnicą wartości znaków tej liczby – jest to przykład **addytywnego systemu liczbowego**. Popularne były również **alfabety liczbowe**, czyli oznaczenia cyfr i liczb za pomocą liter (znaków) alfabetu. Taką notacją do dzisiaj posługują się Żydzi, np. przy numerowaniu stron i wersetów Starego Testamentu oraz dzieł pisanych po hebrajsku. Chińczycy (w XXX wieku p.n.e.) także wprowadzili numerację pisaną, którą stosują do dzisiaj oprócz numeracji arabskiej. Od Chińczyków notację tę zapożyczyli Japończycy.

Rysunek 1. Przykłady liczb zapisanych w innych systemach niż dziesiętny (tablica z Gubbio we Włoszech, zegary z Pragi – górny tradycyjny, a dolny z tarczą hebrajską)

2.2. Pierwsze 'maszyny' do rachowania i algorytmy

Ludzie od dawna w swoich zajęciach praktycznych rachowali, czyli odliczali, różne rzeczy, takie jak zdobycze i łupy z walk czy zwierzęta, które hodowali i wymieniali na inne towary. Na początku stosowano w tym celu **palce u rąk** i ich człony, a dla zwiększenia zakresu liczb – również palce u nóg, a także inne części ciała. W Chinach podobno potrafiono wykonywać obliczenia na palcach i częściach obu rąk aż do dziesięciu miliardów!

Z czasem pojawiła się naturalna konieczność trwałego zapisywania liczb i wyników obliczeń. Najstarsze znane metody polegały na **stosowaniu nacięć** na kościach, kawałkach drewna lub wyżłobień na kamieniach. Co ciekawe, nacięcia na drewnie dla oznaczania liczb stosowano w Europie jeszcze w XIX wieku.

Popularnym sposobem zapisywania liczb i wykonywania na nich obliczeń w większości cywilizacji w przeszłości było posługiwanie się **sznureczkami z węzełkami**. Stosowano je na Bliskim Wschodzie w V wieku p.n.e. i nawet nieco wcześniej na Dalekim Wschodzie, gdzie ten sposób rachowania do dzisiaj nie całkiem zanikł. Najdoskonalsze sznureczki z węzełkami, zwane ***quipu***, stosowano w cywilizacji Inków w pierwszej połowie drugiego tysiąclecia naszej ery. Oznaczano z ich pomocą liczby przedstawiane na bazie systemu dziesiętnego (ale w sposób addytywny) i używano ich głównie w administracji jako archiwa budżetowe. Udoskonalonym *quipu*, zwanym *chimpu*, posługują się jeszcze dzisiaj Indianie boliwijscy i peruwiańscy.

2.2.1. Abaki

Największą rolę w rozwoju pierwszych narzędzi służących do obliczeń odegrały **kamyki**. Kamyk, kamyczek to po łacinie *calculus* – co brzmi jak rdzeń

wielu słów w wielu językach, związanych z obliczeniami, np. kalkulować. Początkowo kamyki układano w stosy. Z czasem zaczęto je układać na 'planszach', które służyły do wykonywania obliczeń – były to pierwsze **abaki**. Mogła to być odpowiednio przygotowana powierzchnia piasku lub kamienia (np. marmuru), na której zaznaczano pionowe lub poziome rowki i w nich układano kamyki – poszczególne rzędy odpowiadały zwykle (np. w starorzymskich abakach) określonej potędze liczby 10. Abaki nie wymagały wprowadzenia zera – odpowiadało mu puste miejsce w rzędzie. Z czasem abaki stały się przenośne dzięki zmniejszeniu ich wielkości. Później wprowadzono pręty, na które nawlekano żetony zrobione z różnych materiałów, co umożliwiało utrzymywanie ich w odpowiednich miejscach względem siebie, stan obliczeń w abakach określa bowiem rozmieszczenie elementów ruchomych (kamieni, żetonów) na piasku lub na prętach.

Rysunek 2. Różne liczydła: japoński soroban (stary i nowy), chiński suan-pan, rosyjskie schoty

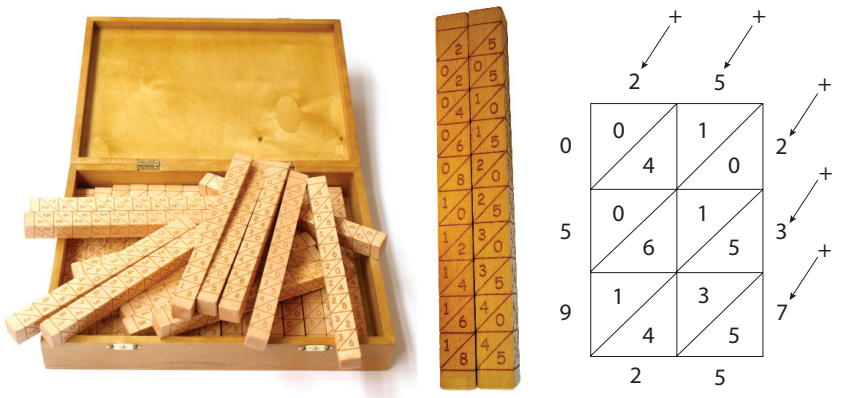
Kolejne rzędy w starszych abakach, w których układano kamyki, i pręty w późniejszych modelach liczydeł odpowiadały pozycjom w systemie rachowania. Abaki rozwinęły się niezależnie w różnych częściach świata, istnieje wiele ich odmian i na ogół są znane pod swoimi lokalnymi nazwami, patrz rys. 2. W Chinach jest to ***suan-pan***, w Rosji – ***schoty*** (wynalezione w XVII wieku), a w Japonii – ***soroban***. Udoskonalano je głównie z myślą o usprawnieniu i przyspieszeniu wykonywania działań, np. przez redukcję liczby żetonów w rzędach lub na prętach. Współczesne abaki są nazywane **liczydłami**. W wielu krajach liczydła

były powszechnie stosowane jeszcze w drugiej połowie XX wieku i do dzisiaj np. w Japonii i w Rosji nie zostały całkowicie wyparte przez elektroniczne kalkulatory. W Japonii, uznawanej za kraj powszechnej elektronizacji, soroban jest nadal wykorzystywany w małych sklepikach rodzinnych i w niektórych urzędach (np. na poczcie) i uczą się stosować go w obliczeniach dzieci w szkołach.

Abakusy i liczydła mają wady, które częściowo zostały usunięte w kalkulatorze, a ostatecznie dopiero w komputerach. Służą one tylko do zaznaczenia bieżących wyników obliczeń, ale nie ma w nich miejsca ani na zapamiętywanie wyników pośrednich i końcowych, ani na zapamiętywanie kolejno wykonywanych działań.

2.2.2. Pałeczki Napiera

Mnożenie przez siebie liczb wielocyfrowych od zarania dziejów sprawiało człowiekowi wiele trudności. Próbowano radzić sobie na wiele sposobów, również za pomocą różnorodnych diagramów i przyborów. Podstawą większości wczesnych metod obliczania iloczynu liczb wielocyfrowych było w oczywisty sposób zastąpienie mnożenia przez wielokrotne dodawanie. Składniki tego dodawania wyznaczano na różne sposoby, np. jako iloczyny jednego z czynników przez odpowiednie potęgi liczby 2 (otrzymywane w wyniku wielokrotnego podwajania czynnika, czyli dodawania do siebie) lub jako iloczyny jednego z czynników przez pojedyncze cyfry. Ten pierwszy sposób jest czasem nazywany **metodą rosyjskich chłopów** (patrz [10]), ten drugi zaś odnajdujemy w bardzo starych **tabliczkach mnożenia**, zwanych obecnie ***gelosia***. Na przełomie XVI i XVII wieku, **John Napier** (1550-1617), lord, matematyk szkocki, zagorzały protestant, wynalazca różnych narzędzi i instrumentów, usprawnił posługiwanie się tabliczkami mnożenia w wersji pisanej, wprowadzając w miejsce tabliczki pałeczki, zwane **pałeczkami Napiera** (patrz rys. 3), a w wersji dla zamożnych – kośćmi Napiera, gdyż były zrobione z kości słoniowej.



Rysunek 3. Pałeczki Napiera – wyrób współczesny. Ustawienie pałeczek do wykonania iloczynu 25 x 237 i schemat dodawania w tym przypadku

Pałeczki Napiera były przeznaczone do wykonywania mnożenia. Algorytm mnożenia dwóch liczb jest przedstawiony w tabeli 1. Niewiele się on różni od pisemnego mnożenia dwóch liczb. Jedynym uproszczeniem jest rozbicie iloczynu pierwszego czynnika przez cyfrę z drugiego czynnika (patrz rys. 3), ale czy rzeczywiście jest to uproszczenie?

Tabela 1.
Algorytm mnożenie za pomocą pałeczek Napiera

	Oblicz: 25 x 237 = 5925
1. Ustaw obok siebie pałeczki oznaczone u góry cyframi pierwszego czynnika, w kolejności zgodnej z kolejnością cyfr w tej liczbie.	Patrz rys. 3
2. Wypisz pod sobą z przesunięciem sumy (na ukos) elementów w wierszach, odpowiadających kolejnym cyfrom od końca drugiego czynnika.	175 075 050
3. Dodaj z przeniesieniem liczby umieszczone w wierszach.	Razem: 5925

Pałeczki Napiera wykorzystał Wilhelm Schickard w swojej maszynie, uważanej za pierwszy kalkulator mechaniczny zbudowany przez człowieka, patrz p. 3.1.

2.2.3. Algorytmy

Na chwilę przerwiemy omawianie urządzeń do liczenia, by wspomnieć o pojęciu **algorytm.** Oznacza ono zbiór poleceń wykonywanych krok po kroku, aby otrzymać pożądany wynik lub osiągnąć zamierzony cel, to także opis posłużenia się urządzeniem do liczenia. Babilończycy używali algorytmów nie tylko w obliczeniach matematycznych, ale również w innych dziedzinach życia, takich jak stosowanie prawa, nauka gramatyki języka, przewidywanie przyszłości, porady medyczne i przygotowywanie potraw.

Wśród osiągnięć greckich matematyków, żyjących w ostatnich pięciu wiekach przed naszą erą, znajdują się zarówno dowody doniosłych twierdzeń matematycznych, takich jak twierdzenie Pitagorasa, jak i przepisy służące do wykonania obliczeń. Najbardziej znanymi algorytmami pochodzącymi ze starożytności są algorytm Euklidesa i sito Eratostenesa. **Algorytm Euklidesa** służy do wyznaczania największego wspólnego dzielnika dwóch liczb (patrz p. 7.1). Euklides zamieścił go w swoim fundamentalnym dziele *Elementy*, które aż do XIX wieku było wykorzystywane jako podręcznik szkolny. Sam algorytm Euklidesa przez długie lata występował w informatyce jako synonim pojęcia algorytm – zasługuje na to miano swoimi własnościami informatycznymi i rozległością zasto-

sowań. Z kolei **sito Eratostenesa** jest metodą, umożliwiającą generowanie kolejnych liczb pierwszych. Ten algorytm ma jednak mniejsze znaczenie praktyczne niż algorytm Euklidesa, gdyż obecnie największym wyzwaniem dla informatyków jest pogoń za największymi liczbami pierwszymi.

Algorytmami posługiwano się znacznie wcześniej niż wymyślono dla nich nazwę. Słowo **algorytm** pochodzi od brzmienia fragmentu nazwiska – Muhammad (Mohammed) ibn Musa **al-Chorezmi** – wspomnianego wcześniej matematyka arabskiego. W Średniowieczu terminem algorytm określano rutynową procedurę obliczeń arytmetycznych, wykonywaną na piśmie za pomocą cyfr arabskich i systemu dziesiętnego. Ten sposób rachowania z trudem zdobywał sobie popularność. Kilka wieków trwał spór między 'abacystami', którzy starali się zachować abaki (czyli wykonywanie rachunków za pomocą żetonów na planszach), a 'algorystami', którzy dążyli do wprowadzenia rachunków na piśmie za pomocą cyfr arabskich.

Z nastaniem ery komputerów pojęcie algorytmu uległo uściśleniu – **algorytmem** jest ścisły przepis, który gwarantuje otrzymanie w skończonej liczbie kroków odpowiedzi na postawione pytanie lub rozwiązanie problemu. W odniesieniu do komputera można powiedzieć, że każdy program komputerowy jest zapisem jakiegoś algorytmu w języku, który jest zrozumiały dla komputera. Zatem komputer to maszyna służąca do wykonywania algorytmów, zapisanych w zrozumiałym dla niego języku. Nie inaczej było z wcześniejszymi urządzeniami do obliczeń – służyły do wykonywania obliczeń według algorytmów, które składały się z ciągu operacji, na ogół manualnych na tych urządzeniach – ilustrujemy to opisami algorytmów zamieszczonymi w kolejnych tabelach.

2.2.4. Dwa algorytmy automatycznych obliczeń

Przedstawiamy w tym punkcie dwa algorytmy, które są stosowane w mechanicznych kalkulatorach. Pierwszy z nich służy do automatycznego mnożenia dwóch liczb, a drugi znajduje zastosowanie przy odejmowaniu. W obu algorytmach podstawowym działaniem jest **dodawanie**. Kalkulatory, w których te operacje są wykonywane zgodnie z poniższymi algorytmami, nazywa się niekiedy **sumatorami**. Często były one budowane z myślą o realizacji tych algorytmów.

Mnożenie przez skomasowane dodawanie

Pierwszy algorytm polega na zastąpieniu mnożenia przez skomasowane dodawanie. Jeśli mamy obliczyć iloczyn $a \times b$, to zamiast dodania liczby a do siebie b razy, dodawane są większe sumy częściowe, dzięki czemu liczba dodawań jest znacznie mniejsza.

Tabela 2.
Algorytm mnożenia przez skomasowane dodawanie

	Oblicz: 25 x 237 = 5925
Jako początkowy wynik mnożenia ustaw 0.	
1. Ustaw (wybierz) pierwszy czynnik.	Patrz rys. 13.
2. Do wyniku dodaj pierwszy czynnik tyle razy, ile wynosi ostatnia od końca cyfra drugiego czynnika.	Początkową wartością wyniku jest równe 25 x 7 = 175,
3. Dla kolejnych od prawej pozycji w drugim czynniku, do wyniku dodaj pierwszy czynnik, wcześniej przesunięty o kolejne miejsce w lewo, pomnożony tyle razy ile wynosi cyfra stojąca na tej pozycji w drugiej liczbie.	25 x 7 = 175 250 x 3 = 750 2500 x 2 = 5000 Razem: 25 x 237 = 5925

W algorytmie w tabeli 2 w kroku 1., „ustaw" oznacza ustawienie cyfr pierwszego czynnika, korzystając z mechanizmu kalkulatora (np. jak w „kręciołkach", patrz p. 3.4 i rys. 13) lub przygotowanie się do wielokrotnego posłużenia się tym czynnikiem w obliczeniach (jak w sumatorach biurowych, patrz p. 3.5).

Uzasadnieniem poprawności powyższego algorytmu może być następujący ciąg przekształceń przykładowego iloczynu 25 x 237 do postaci, która odpowiada wykonywanym działaniom w algorytmie:

25 x 237 = 25 x (7 + 3x10 + 2x102) = 25 x (7 + 30 + 200) = 25 x 7 + 25 x 30 + 25 x 200 =
= 25 x 7 + 250 x 3 + 2500 x 2 = 5925

Zauważmy, że dzięki takiej interpretacji iloczynu liczb, w przykładzie 25 x 237 zamiast 237 razy dodawać do siebie liczbę 25, wykonanych zostało 7 + 3 + 2 = 12 dodawań, gdyż liczba dodawań jest równa sumie cyfr w drugim czynniku.

Odejmowanie przez dodawanie

Drugi algorytm dotyczy zastąpienia odejmowania liczby przez dodanie odpowiednio zmienionego odjemnika, czyli liczby, którą odejmujemy. Ten algorytm jest niezbędny do wykonywania odejmowania, a więc t dzielenia za pomocą kalkulatorów, które są przeznaczone tylko do wykonywania dodawania.

Uzasadnienie poprawności tego algorytmu również podamy na przykładzie. Wyróżniona na czerwono cyfra **1** znajduje się na pozycji, której... nie ma w kalkulatorze. A dokładniej, jeśli w kalkulatorze liczba może mieć pięć cyfr, jak w przykładzie w tabeli 3, to obliczenie różnicy 237 – 48 zostaje zastąpione przez obliczenie wartości wyrażenia 237 – 48 + **1**00000, a to z kolei polega na następującym pogrupowaniu działań:

$237 - 48 + 100000 = 237 - 48 + 99999 + 1 = 237 + (99999 - 48) + 1 = 237 + 99951 + 1 =$
$= 100188 + 1 = 100189$

Tabela 3.
Algorytm odejmowania przez dodawanie

	Oblicz: 237 – 48 = 189
1. Przypuśćmy, że chcemy obliczyć różnicę *a* – *b*.	237 – 48
2. Uzupełnij początkowymi zerami liczbę *a* do maksymalnej liczby cyfr, jaka może wystąpić w obliczeniach (w kalkulatorze). Do *a* dodaj liczbę, której cyfry są uzupełnieniami do 9 cyfr liczby *b*.	00237 99999 – 48 = 99951 Razem: 100188
3. Do otrzymanego wyniku dodaj 1.	100188 1 Razem: 100189

A więc otrzymujemy poprawny wynik, gdyż tej jedynki 1 w wyniku nie ma w kalkulatorze. Utworzenie uzupełnienia liczby *b* do liczby złożonej z samych dziewiątek faktycznie nie jest wykonywane w kalkulatorach, w których odejmowanie jest wykonywane zgodnie z powyższym algorytmem. W tych kalkulatorach cyfry odjemnej *a* i cyfry odjemnika *b* są odmiennie oznaczone (te drugie są na przykład mniejsze lub/i czerwone) i liczbę *a* wybieramy, używając tych pierwszych cyfr, a liczbę *b* – tych drugich, patrz rys. 5, 10.

Wiemy już, jak automatyzować dodawanie, mnożenie i odejmowanie. Z czterech podstawowych działań pozostało jeszcze dzielenie. Nie wynaleziono dla dzielenia żadnego specjalnego algorytmu i jest ono na ogół wykonywane jako wielokrotne odejmowanie, podobnie jak mnożenie jest wielokrotnym dodawaniem. W niektórych prostych kalkulatorach jest osobna skala dla odejmowania, patrz rys. 8.

3. Kalkulatory mechaniczne

Urządzenia obliczeniowe mające na celu ułatwienie i przyspieszenie obliczeń zwykło się nazywać **kalkulatorami**, chociaż oryginalne nazwy, zwłaszcza pierwszych takich urządzeń były przeróżne. To miano odnosi się zarówno do urządzeń mechanicznych, elektrycznych (tj. mechanicznych o napędzie elektrycznym), jak i elektronicznych. Kalkulator odróżnia się od komputera brakiem możliwości jego programowania, chociaż nie jest to do końca prawdą – niektóre bardziej zaawansowane kalkulatory elektroniczne można programować i można

wykonywać z ich pomocą programy. Z drugiej strony, na początku ery komputerów niektóre komputery miały w nazwie *calculator*.

W tym rozdziale zajmujemy się kalkulatorami mechanicznymi. **Kalkulatorom elektrycznym** nie poświęcamy odrębnego miejsca, gdyż były to konstrukcje, w których silnik elektryczny służył wykonywaniu czynności wcześniej realizowanych mechanicznie. Kalkulatorami elektronicznymi nie zajmujemy się tutaj w ogóle.

Podstawowym działaniem w większości kalkulatorów mechanicznych jest dodawanie, stąd popularna ich nazwa **sumatory**, stosowana najczęściej do kalkulatorów, które służą głównie do wykonywania dodawania. Jednak, jak pokazaliśmy w p. 2.2.4, dysponując w kalkulatorze tylko dodawaniem, można go użyć również do wykonywania odejmowania, mnożenia i dzielenia. W wielu kalkulatorach bazujących na dodawaniu, wykonywanie innych działań znacznie uproszczono, stosując przy tym różne rozwiązania techniczne.

Wśród kalkulatorów mechanicznych, ze względu na podstawowe mechanizmy ich funkcjonowania i sposoby wykonywania działań, można wyróżnić kilka grup:

- **sumatory proste**, które służą do dodawania lub dodawania i odejmowania, a pozostałe działania muszą być sprowadzone do tych dwóch działań;
- kalkulatory, w których są wykorzystywane **bębny z zębami** (ang. *stepped drum*), użyte po raz pierwszy przez Leibniza – ten mechanizm ułatwia dodawanie (z przenoszeniem), a więc i mnożenie całych liczb, patrz p. 3.3;
- kalkulatory, w których są wykorzystywane **koła z ruchomymi zębami** (ang. *pinwheel*) – ten mechanizm również ułatwia dodawanie (z przenoszeniem), a więc i mnożenie całych liczb – patrz p. 3.4;
- **sumatory klawiaturowe** – pełnoklawiszowe lub tylko z 10 klawiszami z cyframi, patrz p. 3.5.

Kalkulatory mechaniczne, zwłaszcza w wersji biurowej, były często wyposażone w dodatkowe urządzenie do drukowania, patrz rys. 7.

3.1. Pierwsze kalkulatory

Maszyna Schickarda

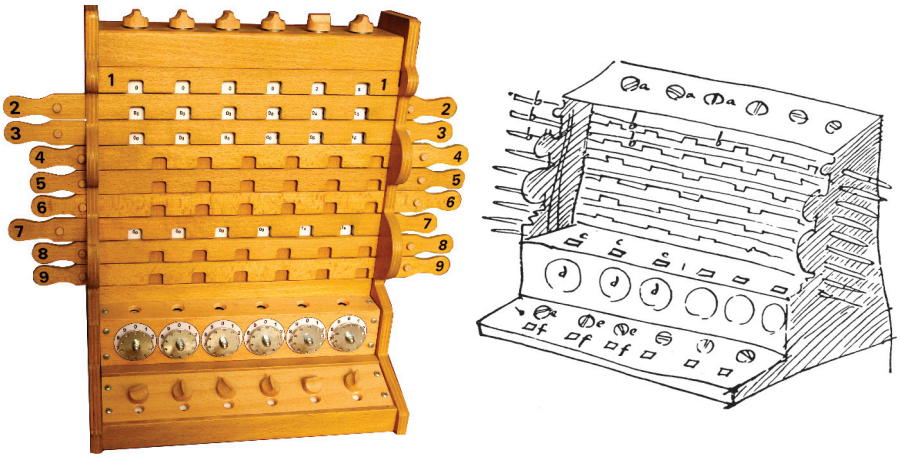
Za twórcę pierwszej w historii maszyny liczącej jest uznawany **Wilhelm Schickard** (1592-1635), profesor języków biblijnych, matematyk, geodeta niemiecki. W roku 1623 ukończył on budowę maszyny, zwanej **zegarem liczącym**, w której wykorzystał udoskonalone pałeczki Napiera w postaci walców do automatyzacji mnożenia (patrz rys. 4). Maszyna ta miała pomóc Johannesowi Keplerowi w jego astronomicznych (dosłownie i w przenośni) rachunkach. Niestety, zbudowana z drewna, spłonęła w niewyjaśnionych okolicznościach.

Przez ponad trzysta lat za wynalazcę pierwszej maszyny do liczenia uchodził Pascal. O maszynie Schickarda, po latach zapomnienia, dowiedziano się bowiem dopiero pod koniec lat 50. XX wieku po odczytaniu jego listów do Keplera, znajdujących się w zbiorach carycy Katarzyny II, oraz po odnalezieniu w tym samym czasie w Niemczech szkiców przygotowanych przez Schickarda dla osoby, która miała wykonać jego maszynę. Na podstawie tych dokumentów zbudowano repliki maszyny Schickarda (patrz rys. 4).

Można mieć wątpliwości, czy kalkulator Schickarda byłby pomocny Keplerowi – służył mianowicie do obliczeń na liczbach co najwyżej sześciocyfrowych i automatyzował jedynie przenoszenie przy dodawaniu. Algorytm mnożenia dwóch liczb za pomocą tej maszyny jest przedstawiony w tabeli 4.

Tabela 4.
Algorytm mnożenie za pomocą maszyny Schickarda

<table>
<tr><th></th><th>Oblicz: 25 x 237 = 5925</th></tr>
<tr><td>Stan początkowy (zerowy) maszyny:
• pokrętła u góry i u dołu oraz wszystkie tarcze mechanizmu zegarowego są ustawione na 0;
• deszczułki poziome oznaczone cyframi są dosunięte do prawej i zasłaniają wszystkie okienka z liczbami.
Wynik mnożenia jest obliczany za pomocą mechanizmu zegarowego.</td><td rowspan="4">Ustawienie maszyny do wykonania przykładowego mnożenia, patrz rys. 4.</td></tr>
<tr><td>1. Pokrętłami u góry ustaw jedną z liczb (od prawej strony).</td></tr>
<tr><td>2. Przesuń w lewo te poziome deszczułki, które są oznaczone cyframi, występującymi w drugiej liczbie.</td></tr>
<tr><td>3. Pokrętłami u dołu ustaw drugą z liczb (od prawej strony).</td></tr>
<tr><td>4. Stosując mechanizm zegarowy, dodawaj cyfry z odsłoniętych okienek rzędami, które odpowiadają kolejnym cyfrom drugiej liczby. Cyfry odsłonięte w okienkach jednego rzędu, skumulowane na ukos, dodawaj mechanizmem zegarowym od miejsca wskazanego przez cyfrę drugiej liczby.</td><td>175
075
050
Razem: 5925</td></tr>
</table>



Rysunek 4. Maszyna Schickarda (replika), szkic maszyny z rękopisów Schickarda

Pascalina

Blaise Pascal (1623-1662), matematyk, fizyk i filozof francuski, mając 18 lat zainteresował się zbudowaniem maszyny liczącej, służącej głównie do dodawania, z myślą o dopomożeniu swojemu ojcu, który był poborcą podatkowym. Pierwszy egzemplarz takiej maszyny ukończył w roku 1645 i przez następne 10 lat wyprodukowano około 50 egzemplarzy **Pascaliny** – maszyny według pomysłu Pascala. Kilka egzemplarzy istnieje w muzeach do dzisiaj (patrz rys. 5); część z nich była przeznaczona do obliczeń w różnych systemach monetarnych, a część – dla różnych miar odległości i powierzchni (z przeznaczeniem dla geodetów). Można więc uznać, że ten pierwszy sumator został zaprojektowany z myślą o zastosowaniach praktycznych. Pascalina swoją budową przypomina licznik, w którym można zwiększać każdą pozycję obracając odpowiednie koło, koła zawierają zęby i są sprzężone ze sobą tak, że jest możliwe przenoszenie cyfr. Taki **mechanizm** można nazwać **licznikowym**. W niektórych tego typu kalkulatorach można kręcić koła również w drugą stronę, co umożliwia wykonywanie odejmowania.

Niektóre późniejsze kalkulatory budowane na idei Pascaliny, jak Addometer (patrz rys. 5), zawierały także mechanizm zerowania (wyciągana dźwignia z prawej strony). Wykonanie dodawania za pomocą takiego kalkulatora polega na wprowadzeniu kolejnych od prawej cyfr dodawanej liczby, wybierając większe cyfry na okręgu i kręcąc za pomocą odpowiedniego sztyftu (znajduje się pod dźwignią do zerowania), długopisem lub ołówkiem zgodnie z ruchem wskazówek zegara. Addometer umożliwia również odejmowanie – należy cyfry odejmowanej liczby wybierać spośród małych cyfr na okręgu i kręcić przeciwnie do ruchu wskazówek zegara.

Rysunek 5. Pascalina ze zbiorów muzeum Salonu Aparatów Matematycznych i Astronomicznych (Zwinger) w Dreźnie; Addometer, późniejszy kalkulator zbudowany na idei Pascaliny – widok zewnętrzny i widok mechanizmu; Kalkulator See z widocznym mechanizmem dodawania i przenoszenia

3.2. Sumatory proste

Sumatory proste, z różnymi mechanizmami wykonywania dodawania i odejmowania oraz przenoszenia wyników, były produkowane niemal do ostatnich dni przed pojawieniem się kalkulatorów elektronicznych. Były wśród nich wersje kieszonkowe, podręczne i biurowe, z dodatkową opcją wykonywania odejmowania, z pamięcią (!), z możliwością drukowania itd. Wyprodukowano miliony takich kalkulatorów. Na rysunkach 6-10 ilustrujemy kilka typowych mechanizmów

Na rysunku 6 pokazano kalkulatory będące typowymi sumatorami, działające na zasadzie mechanizmu licznikowego.

Rysunek 6. Kalkulatory: Webb, Stephenson, BriCal

Rysunek 7 przedstawia kalkulatory, w których koła z cyframi są obracane w wyniku przesuwania odpowiednich sztabek. Te sumatory mogą działać w dwóch trybach: z zatrzymaną ostatnio dodaną liczbą i bez jej zatrzymania. Pokazany na rysunku Rapid computer, to według autora pierwsze urządzenie, które ma w nazwie słowo computer[2], jest to jednak nazwa firmy. Te kalkulatory umożliwiały również wykonywanie odejmowania, według algorytmu opisanego w tabeli 3.Jedną z najpopularniejszych marek kalkulatorów kieszonkowych był **Addiator**. W wielu krajach produkowano kalkulatory zbudowane na podobnej zasadzie, również w Polsce (**kalkulator Kopernik**). Te proste w budowie kalkulatory służą do dodawania i odejmowania liczb. Daną liczbę dodajemy przesuwając kolejne od prawej kolumny o tyle oczek, ile wynosi kolejna cyfra w dodawanej liczbie. Jeśli dodawana cyfra nie powoduje przeniesienia, to kolumnę przesuwamy do dołu, a jeśli spowoduje przeniesienie (w takim przypadku dodawana cyfra znajduje się na czerwonym polu), to kolumnę przesuwamy do góry z lekkim nawrotem po dojechaniu do górnej krawędzi kolumny. Odejmowanie wykonuje się podobnie, albo posługując się tymi samymi kolumnami, ale cyframi zapisanymi na ogół w innym kolorze i po drugiej stronie kolumny, albo korzystając z dodatkowego zestawu kolumn poniżej tych, które służą do dodawania.

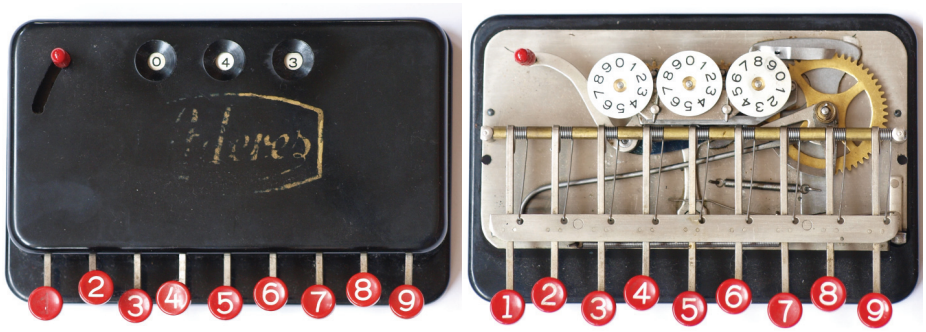
[2] Według słownika języka angielskiego (*Webster's New World Dictionary*, Simon and Schuster, 1969): **computer** – **1.** a person who computes **2.** a devices used for computing.

Rysunek 7. Kalkulatory: Rapid computer, Seidel&Neumann z drukarką

Rysunek 8. Addiatory: w systemie dziesiętnym, dla starej waluty brytyjskiej, na rynek arabski

Tego typu kalkulatory były produkowane dla różnych walut (np. brytyjskiej, patrz rys. 8), dla różnych systemów pozycyjnych (np. dla szesnastkowego), jak również w innych językach (np. w arabskim) i do obliczeń na różnych wielkościach (np. na czasie).

Na rysunku 9 przedstawiono jeszcze kilka innych sumatorów, działających na innych zasadach.



Rysunek 9. Jeszcze inny sumator – Adares

Rysunek 10 przedstawia kilka kalkulatorów zaprojektowanych specjalnie do użytku w szkołach.

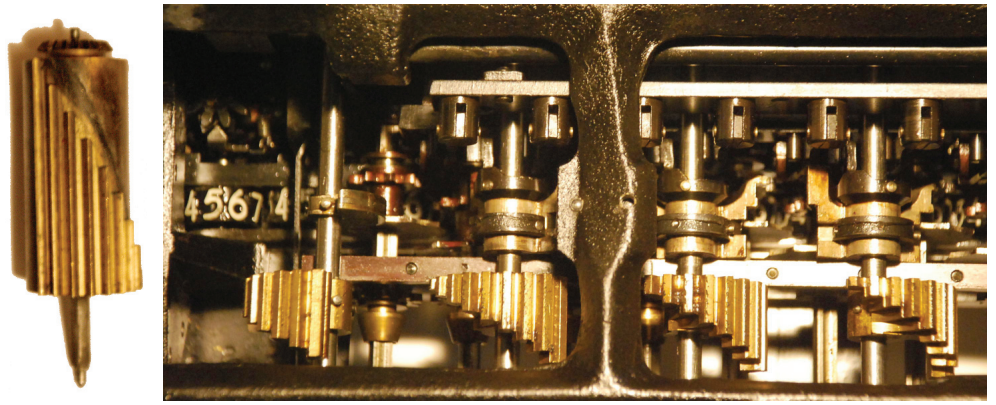
Rysunek 10. Szkolne kalkulatory: piórnik z sorobanem (Japonia), piórnik do przechowywania ołówków (USA), sumator

3.3. Maszyny wykorzystujące bęben schodkowy Leibniza

Gottfried Wilhelm Leibniz (1646-1716) był prawnikiem z wykształcenia, filozofem, matematykiem i niemieckim mężem stanu. Rozległością swoich zainteresowań jest przyrównywany do Leonarda da Vinci. Głównym osiągnięciem matematycznym Leibniza było stworzenie rachunku różniczkowego i całkowego, niezależnie od Isaaca Newtona. Wśród największych zasług Leibniza w informatyce uznaje się opisanie binarnego systemu obliczeń oraz **bęben schodkowy** (ang. *stepped drum*), patrz rys. 11, który zastosował w swojej maszynie, ukończonej w roku 1694 po ponad 20 latach wysiłków. Była to pierwsza w pełni mechaniczna maszyna do mnożenia. Chociaż w tym czasie istniała już Pascalina i Leibniz miał możność zapoznania się z nią w Paryżu, projekt swojej **żywej ławy do liczenia** opisał przed pierwszą wizytą w Paryżu. Koncepcyjnie, maszyny Pascala i Leibniza mają niewiele wspólnego pod względem zastosowanych mechanizmów.

Przez blisko 300 następnych lat wynalazek Leibniza był jednym z podstawowych rozwiązań konstrukcyjnych w maszynach i kalkulatorach mechanicznych.

Prawdziwą erę kalkulatorów rozpoczął francuski wynalazca i producent **Charles Xavier Thomas de Colmar** (1785-1870). W roku 1820 opatentował on pierwszą wersję swojego kalkulatora, który nazwał **Arithmometre** – stąd pochodzi później używana nazwa **arytmometr**. Zastosował w nim bęben schodkowy Leibniza udoskonalony wcześniej przez Ottona Hahna. Dzięki perfekcyjnemu wykonaniu i niezawodności konstrukcji, co było możliwe dopiero w erze rewolucji przemysłowej w pierwszej połowie XIX wieku, a także zastosowaniom, które użytkownikom przynosiły wymierne korzyści ekonomiczne, arytmometr Thomasa odniósł także sukces rynkowy – szacuje się, że wyprodukowano ich ponad 5000. Arytmometry te znalazły zastosowanie w handlu, pracy biurowej, a także w nauce.



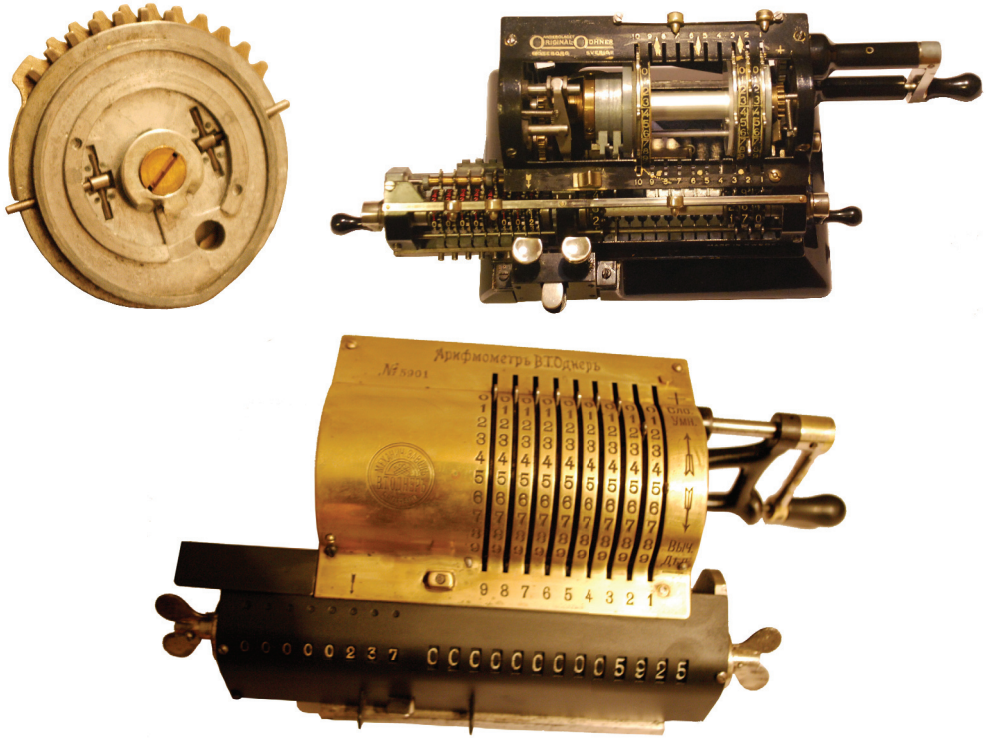
Rysunek 11. Bęben schodkowy Leibniza (element wyjęty z Arithmometre) i jego wykorzystanie w maszynie Rheinmetall na każdej pozycji liczb

Po wielu latach, w czasie II wojny światowej, powstała jeszcze jedna konstrukcja kalkulatora, w której wykorzystano bęben schodkowy Leibniza. Chociaż wydawało się, że to rozwiązanie znacznie powiększa wielkość i wagę maszyny (patrz maszyna Rheinmetall na rys. 11), powstał kalkulator kieszonkowy! Skonstruował go w roku 1943 **Curt Herzstark** (1902-1988) będąc więźniem obozu koncentracyjnego w Buchenwaldzie. Szacuje się, że do pojawienia się kalkulatorów elektronicznych w roku 1972 wyprodukowano blisko 150 000 sztuk tego kalkulatora w dwóch modelach, noszących nazwę **Curta**, patrz rys. 12 i na końcu rozdziału – autor z tym kalkulatorem. Genialnym pomysłem było użycie w tych kalkulatorach jednego bębna Leibniza dla wszystkich pozycji w liczbach, zarówno do dodawania, jak i do odejmowania. Kalkulator ten uchodzi za szczytowe osiągnięcie w dziedzinie mechanicznych urządzeń do obliczeń.

Działania za pomocą kalkulatora Curta wykonuje się bardzo podobnie, jak za pomocą „kręciołków", o których jest mowa w następnym punkcie. Na przykład, aby obliczyć iloczyn dwóch liczb $a \times b$, jedną z nich, np. a, należy ustawić suwakami na boku kalkulatora i następnie wykonywać skomasowane dodawania (algorytm z tabeli 2) kręcąc korbką tyle razy, ile wynosi wartość cyfry w drugiej liczbie, poczynając od najmniej znaczącej cyfry. Przed wykonaniem działań dla kolejnych cyfr drugiej liczby, należy obrócić o jedną pozycję górną część kalkulatora, wcześniej ją podnosząc. Odejmowanie wykonuje się podobnie, wcześniej podnosząc korbkę. Do zerowania służy pierścień w górnej części kalkulatora, którym, po podniesieniu górnej części kalkulatora, należy wykonać pełny obrót.

Rysunek 12. Kalkulator Curta model II

3.4. Maszyny wykorzystujące koła z ruchomymi zębami Odhnera – „kręciołki"



Rysunek 13. Koło z ruchomymi zębami. Demonstracyjny (pocięty) model kalkulatora Original Odhner. Kalkulator Odhnera wyprodukowany w St. Petersburgu (ustawienia po wykonaniu mnożenia 25 x 237)

Bębny schodkowe Leibniza zajmowały w kalkulatorach dość dużo miejsca (wyjątek stanowią kieszonkowe kalkulatory Curta), zwiększały także ciężar tych maszyn. Pod koniec XIX wieku pojawiła się całkiem nowa konstrukcja kalkulatorów, w których wykorzystano **koło zębate**, a dokładniej **koło z ruchomymi zębami** (ang. *pinwheel*), patrz rys. 13. Wynalazcami takich kalkulatorów byli **Frank S. Baldwin** w USA i Szwed **Willgodt T. Odhner** w Rosji. Baldwin opatentował swój wynalazek w roku 1872, a pierwszy model kalkulatora według tego patentu pojawił się na rynku w 1875 roku. Z kolei Odhner zbudował pierwszy model swojego kalkulatora w roku 1874, a produkcję kalkulatorów rozpoczął w 1886 roku. Kalkulatory Baldwina nie były zbyt popularne, natomiast kalkulatory Odhnera cieszyły się dużą popularnością aż do lat 70. XX wieku. Jeszcze w XIX wieku Odhner sprzedał prawa do swojej konstrukcji w Niemczech i tak w roku 1892 w Braunschweigu rozpoczęła się produkcja najpopularniejszych kalkulatorów Odhnera pod nazwą **Brunsviga**. Kalkulatory produkowane przez

Odhnera i jego następców, jeszcze w St. Petersburgu, a później w Szwecji, nosiły nazwę **Original Odhner**. Warto wspomnieć, że kalkulatorami Odhnera zainteresował się Feliks Dzierżyński, szef KGB w czasach po rewolucji październikowej w Rosji. Z jego inicjatywy rozpoczęto produkcję kalkulatorów Odhnera w fabryce jego imienia, które nosiły nazwę **Feliks Dzierżyński Arytmometr**, później uproszczoną do **Felix**.

Kalkulatory Odhnera, popularnie zwane w Polsce „kręciołkami", produkowano także w Polsce na licencji szwedzkiej firmy Facit. Dostępne były u nas również radzieckie kalkulatory Felix.

Mnożenie za pomocą kalkulatora Odhnera wykonuje się zgodnie z algorytmem skomasowanego dodawania, przedstawionym w tabeli 2. Przejście do kolejnych pozycji mnożnika następuje przez przesunięcie karetki z wynikiem. Kalkulatory Odhnera, podobnie jak kalkulatory Curta, zawierają licznik obrotów na poszczególnych pozycjach, a więc wartość drugiej liczby iloczynu.

3.5. Maszyny biurowe – kalkulatory klawiaturowe

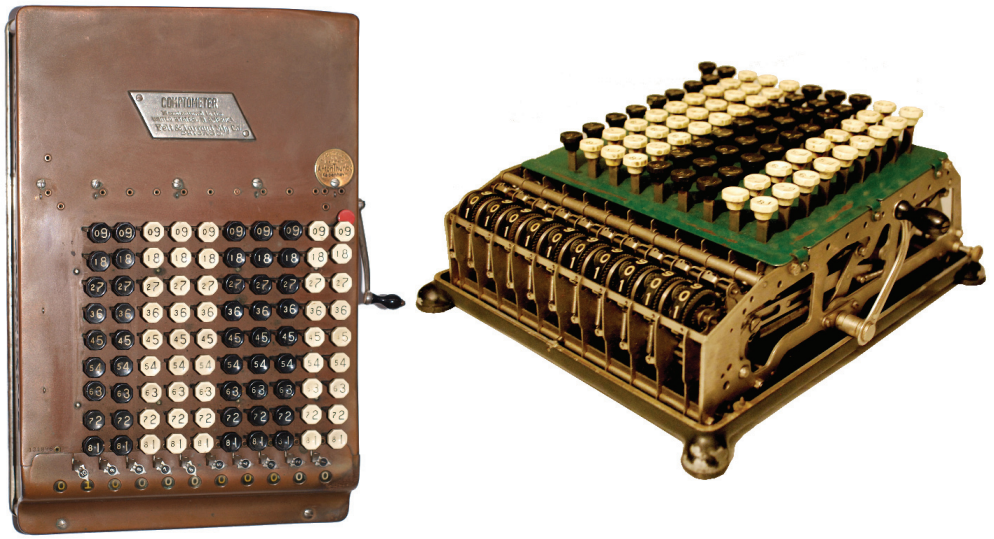
Rozwój przemysłu w erze industrialnej w XIX wieku spowodował duże zapotrzebowanie na maszyny liczące, przeznaczone do wykonywania rachunków finansowych i obliczeń inżynierskich. W biurach znalazły zastosowanie arytmometry Thomasa i kalkulatory Odhnera. Produkowano również kalkulatory, które ułatwiały szybkie wykonywanie dużych obliczeń. Jedną z grup takich maszyn stanowiły **kalkulatory klawiaturowe**, w których do ustawiania liczb służyły klawisze.

Od pierwszych kalkulatorów Schickarda, Pascala i Leibniza, przez następne konstrukcje Thomasa, Baldwina i Odhnera, 'wprowadzanie danych' i dokonywanie innych ustawień polegało na posługiwaniu się m.in. pokrętłami i suwakami, czasem z pomocą dodatkowego sztyftu. Dopiero pod koniec XIX wieku pojawiły się kalkulatory, w których ustawienia danych dokonywało się znacznie szybciej za pomocą klawiszy. Pierwszym kalkulatorem klawiszowym był **Comptometr** z roku 1885 (patrz rys. 14), opatentowany w 1887 roku, którego konstruktorem był **Dorr Eugene Felt** (1862-1930). W tym kalkulatorze Felt zastosował tzw. **pełną klawiaturę**, wynalezioną przez Thomasa Hilla w roku 1857, która składa się z 9 rzędów klawiszy, po jednym dla każdej cyfry na każdej pozycji (nie było klawisza, odpowiadającego cyfrze 0, zero było reprezentowane przez brak wyboru klawisza). Taka klawiatura ułatwiała wprowadzanie i dodawanie całych liczb w jednym ruchu obu rąk. Pełny układ klawiszy na kalkulatorze był bardzo popularny aż do połowy XX wieku.

Jeden z najpopularniejszych sumatorów z pełną klawiaturą (patrz rys. 14) skonstruował **William S. Burroughs** w roku 1884 i w pełni funkcjonalny model opatentował w 1892 roku. Założył on American Arithmometr Company, która

w roku 1905 zmieniła nazwę na Burroughs Adding Machine Company i szybko stała się największą firmą w świecie produkującą drukujące sumatory (w pierwszych 20 latach XX wieku sprzedano ponad milion sumatorów). Po II wojnie światowej firma rozszerzyła swoje działania na komputery, w roku 1953 zmieniła nazwę na Burroughs Corporation, a w roku 1986 połączyła się ze Sperry Corporation, tworząc Unisys Corporation.

W pierwszych pełnoklawiaturowych maszynach biurowych zastosowano mechanizmy, które na każdej pozycji powodowały obrót licznika o liczbę pozycji równą cyfrze na naciśniętym klawiszu i ewentualnie powodowały przeniesienie, jeśli wynik dodawania był większy od 9. Na klawiszach były umieszczone po dwie cyfry, wykorzystywane przy dodawaniu i przy odejmowaniu (patrz opis algorytmu w tabeli 3), zatem te kalkulatory umożliwiały szybkie odejmowanie za pomocą dodawania. Osobny zestaw cyfr na każdej pozycji umożliwiał wprowadzenie całej liczby w jednym naciśnięciu wszystkich klawiszy z cyframi liczby. Co więcej, jeśli ta sama liczba, miała być dodana wielokrotnie podczas wykonywania mnożenia, to wystarczyło utrzymać niezmieniony układ palców i naciskać klawisze wielokrotnie, również po przesunięciu się nad kolejne pozycje. To znacznie przyspieszało obliczenia.



Rysunek 14. Wczesne kalkulatory pełnoklawiaturowe: Comptometer i Burroughs model 5 (z odsłoniętym mechanizmem)

W maszynach biurowych stosowano także inne mechanizmy, na przykład bębny Leibniza czy mechanizm proporcjonalny (np. w maszynie Euklides), z czasem zredukowano także liczbę klawiszy do dziesięciu, patrz rys. 15.

Rysunek 15. Późniejsze kalkulatory biurowe: Remington i Burroughs

4. Suwaki logarytmiczne

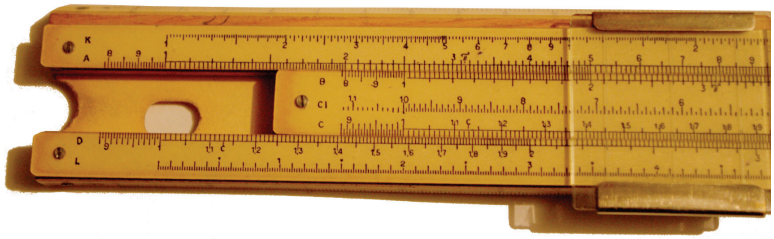
John Napier jest uznawany również za wynalazcę **logarytmów** (logarytm, od gr. słów *logos* i *arithmos*), dzięki którym mnożenie i dzielenie można zastąpić dodawaniem i odejmowaniem. Do tego przydatne są tablice zawierające logarytmy z liczb. Napier opublikował pierwsze **tablice logarytmiczne** w 1614 roku. Swój początkowy pomysł udoskonalił później wspólnie z matematykiem Henrym Briggsem, który główne dzieło na temat logarytmów wydał już po śmierci Napiera. Logarytmami Napiera zainteresował się astronom Johannes Kepler, pracujący nad tablicami astronomicznymi (opublikował je jako *Rudolphine Tables* w roku 1628). Niezależnie od Napiera, ale w nieco inny sposób, logarytmy wprowadził również szwajcarski konstruktor zegarów i aparatów matematycznych Jost Bürgi, którego pomysł spisał Kepler w 1620 roku. Kepler podał również ścisłe uzasadnienie poprawności zasad logarytmowania. Jak napisał francuski matematyk Pierre S. de Laplace (1749-1827):

wynalazek logarytmów podwoił długość życia astronomów.

Krótko po wynalezieniu logarytmów, w roku 1624 matematyk angielski **Edmund Gunter** (1581-1626) opisał **linijkę logarytmiczną**, czyli linijkę ze skalą

logarytmiczną. W roku 1632 inny matematyk angielski **William Oughtred** (1575-1660), skonstruował pierwszy **suwak logarytmiczny** w postać walca, a dwa lata później wykorzystał dwie linijki Guntera do zbudowania suwaka logarytmicznego w postaci dwóch przesuwających się wzdłuż siebie linijek logarytmicznych, co umożliwiło wykonywanie mnożeń i dzieleń jako dodawań i odejmowań (patrz rys. 16).

Suwak logarytmiczny jest przykładem **analogowego** narzędzia matematycznego (obliczeniowego), w którym wielkości występujące w obliczeniach i wyniki obliczeń określa się na podstawie pomiarów innych wielkości – długości odpowiednich odcinków.



Rysunek 16. Dwie linijki logarytmiczne ustawione obok siebie, służą do obliczania iloczynu, w tym przypadku na skali A można odczytać wartości iloczynów przy mnożeniu przez 2,5 (1 na skali B jest ustawione pod 2,5 na skali A)

Suwaków logarytmicznych używali najwięksi naukowcy, począwszy od Keplera, przez Newtona kończąc na Einsteinie. W latach 50. i 60. XX wieku umiejętności wykonywania obliczeń za pomocą suwaka były niezbędne do ukończenia studiów inżynierskich. Faktycznie suwakiem logarytmicznym można się posługiwać nie znając logarytmów, tak nielubianych przez wielu uczniów! Na linijkach suwaka umieszczano wiele różnych skal, związanych z różnymi funkcjami i działaniami. Suwak Faber-Casttel 2/83N zawiera 31 skal. Pomimo wielu wad – problem z ustaleniem miejsca dla przecinka w wyniku, braku możliwości dodawania i odejmowania oraz niewielkiej dokładności wyniku (do 3-4 cyfr znaczących) – posługiwano się nimi w obliczeniach towarzyszących największym przedsięwzięciom do końca lat 60. Suwak towarzyszył pierwszej wyprawie na Księżyc w roku 1969 – zauważmy, że wtedy nie było jeszcze kalkulatorów elektronicznych, nie mówiąc o komputerach osobistych. Z suwaków korzystano przy projektowaniu kalkulatorów, które stały za ich zagładą! Szacuje się, że wyprodukowano na świecie ponad 40 mln suwaków, w większości w postaci dwóch przesuwanych linijek logarytmicznych. Suwaki logarytmiczne mają także bardzo ciekawe realizacje w postaci kół, zegarków i walców (patrz rys. 17), które umożliwiają prowadzenie obliczeń z większą dokładnością dzięki dłuższej skali.

Rysunek 17. Przykłady suwaków logarytmicznych: liniowy, kołowy, w postaci zegarka (Fowlers), cylindryczny (Otis King – długość skali 1,5 m) i na walcu (Fuller – długość skali 12 m)

Logarytmy nie zniknęły wraz z suwakami – we współczesnej informatyce (m.in. w algorytmice i teorii informacji) logarytm jest jedną z najważniejszych funkcji, patrz p. 7.

Wyraźnie trzeba podkreślić, że oba wynalazki Napiera, pałeczki do mnożenia i logarytm, chociaż oba upraszczają obliczanie iloczynu, niewiele łączy w sensie matematycznym, oparte są bowiem na innych własnościach działań.

5. Utrwalanie wyników obliczeń i informacji

Chcemy przypomnieć tutaj również maszyny do pisania, które niemal zupełnie wyszły z użytku. Ten wynalazek odcisnął się zarówno na maszynach do wykonywania obliczeń, jak i na dzisiejszych komputerach. Ale wcześniej, kilka słów o utrwalaniu informacji.

5.1. Utrwalanie wiadomości i wyników obliczeń

Ocenia się, że około 5 tysięcy lat temu w dorzeczu Eufratu i Tygrysu pojawiło się **pismo**, czyli układ znaków, który umożliwia utrwalanie informacji. Na początku informacje utrwalano w kamieniu i na papirusie. Przed wynalezieniem druku, księgi zapisywano ręcznie i kopiowano w podobny sposób. Tak utrwalili swoje dzieła Al-Khorezmi, Fibonacci i inni, a później kopiowali je ich następcy, zanim nie został wynaleziony druk.

Pierwsze próby druku miały miejsce w Chinach w IX wieku. W roku 1436 niemiecki złotnik **Johannes Gutenberg** (ok. 1397-1468) wynalazł **prasę drukarską** i w roku 1452 ukazała się drukiem pierwsza książka, **Biblia Gutenberga**. Wynalazek prasy drukarskiej zapoczątkował bujny rozwój drukarstwa, materiały drukowane stały się dostępne dla szerokich mas, rozwinęło się czytelnictwo wśród różnych grup społecznych. Edukacja zaś, dla której pismo nadal stanowi podstawowe medium wymiany informacji, stała się bardziej demokratyczna, dostępna dla każdego. Na przełomie XX i XXI wieku rozwój technologii przyczynił się do powstania elektronicznych form zapisu i rozpowszechniania informacji, ale niewiele osłabił pozycję książki jako formy utrwalania myśli ludzkiej.

Urządzeniami do zapisywania i drukowania byli również zainteresowani wynalazcy maszyn do obliczeń. Wiele modeli kalkulatorów i biurowych maszyn do obliczeń było wyposażanych w specjalne, dodatkowe urządzenia umożliwiające drukowanie kolejnych rezultatów i końcowych wyników obliczeń, patrz rys. 7 i 15.

Najbardziej popularnym nośnikiem danych dla komputerów i wyników obliczeń, nie tylko liczbowych, stały się, wynalezione przez Josepha Jacquarda na przełomie XVIII i XIX wieku i później udoskonalone, **karty perforowane** (**dziurkowane**). Najpierw miały one służyć do zapisywania poleceń, czyli programu, dla maszyny analitycznej Babbage'a (1834), a pod koniec XIX wieku Hollerith wykorzystał praktycznie karty perforowane do zapisywania danych pochodzących ze spisu powszechnego i automatycznego ich przetwarzania za pomocą maszyn tabulacyjnych. Pod koniec lat 20. XX wieku, utrwalił się standard karty perforowanej zaproponowany przez IBM. Stosowane również były taśmy perforowane.

Osobną nieco krótszą historię ma **taśma papierowa** jako nośnik danych i informacji. Zaczęło się od telegrafu Morse'a (patrz p. 6), w którym nadchodzące wiadomości były zapisywane na poruszanej mechanizmem zegarowym taśmie

papierowej za pomocą sterowanego elektromechanicznie ołówka, który pozostawiał na papierze krótsze lub dłuższe kreski, odpowiadające kropkom i kreskom w kodzie Morse'a. Pod koniec XIX wieku intensywnie rozwijano pomysł **drukującego telegrafu**, w którym przesyłane wiadomości miały być drukowane w czytelnej postaci tekstowej. Na początku XX wieku z powodzeniem rozpoczęła się ekspansja systemu **telex**, opartego na urządzeniach typu **teleprinter**, w których wiadomości były nadawane za pomocą klawiatury przypominającej maszynę do pisania, a odbierane były drukowane na papierze podawanym z rolki. Do transmisji wykorzystywano zarówno linie telegraficzne, jak i telefoniczne. Pod koniec XX wieku urządzenia te zostały wyparte przez podłączoną do komputera klawiaturę i ekran monitora, a wiadomości przyjęły postać listów elektronicznych i faxów.

Karty i taśmy perforowane jako nośniki danych i informacji (np. programów) dla komputerów zostały wyparte dopiero w połowie lat 70. przez coraz tańsze nośniki elektroniczne, różnego rodzaju **dyskietki**. Z czasem, konkurencję dla dyskietek zaczęły stanowić **płyty CD**, a następnie **płyty DVD**, a na początku XXI wieku bardzo popularnym i niezmiernie wygodnym nośnikiem informacji w postaci elektronicznej stał się **pen-drive**.

5.2. Maszyny do pisania a procesory tekstu

Upłynęło ponad 400 lat zanim wynalazek Gutenberga z roku 1440 został pod koniec XIX wieku przekuty w maszynę do pisania – osobiste urządzenie do drukowania. To 'opóźnienie' było spowodowane m.in. brakami odpowiedniej technologii i wysokimi kosztami produkcji maszyn. Dopiero era industrialna w XIX wieku stworzyła odpowiednie warunki do budowy pierwszych maszyn do pisania i ich rozpowszechnienia.

Maszyna do pisania należy do historii ludzkich dążeń do komunikowania się, a w historii komputerów może być uznana za prekursora **klawiatury** (patrz p. 7.3), która umożliwia użytkownikowi komunikację z komputerem. Z drugiej strony, wynalazek ten miał oszczędzać czas. Eliphalet Remington, jeden z pierwszych producentów maszyn do pisania, reklamował je tym, że:

oszczędzając czas wydłużają życie.

Pierwszy patent maszyny do pisania uzyskał Anglik **Henry Mill** jeszcze w roku 1714, ale pierwszą praktyczną maszynę do pisania opatentowali w roku 1867 trzej amerykańscy wynalazcy **Christopher L. Sholes**, **Carlos Glidden** i **Samuel W. Soule**, a jej produkcji podjął się w roku 1873 producent broni i maszyn do szycia, firma **E. Remington & Sons**. W roku 1874 Sholes wprowadził układ klawiszy na klawiaturze maszyny do pisania znany obecnie jako QWERTY, patrz p. 7.3.

Pojawienie się maszyn do pisania rozpoczęło głębokie zmiany w organizacji pracy biurowej. Przede wszystkim przyczyniło się do zwiększenia zatrudnienia kobiet, które częściej niż mężczyźni podejmowały pracę w charakterze maszynistki, gdyż mężczyznom nie odpowiadały niskie zarobki na tym stanowisku.

Popularność maszyn do pisania spadła niemal do zera wraz z rozwojem komputerów osobistych, które, wyposażone w edytor tekstu, zastąpiły te maszyny i niestety czasem nie spełniają żadnej innej roli. Niewątpliwą korzyścią płynącą z używania komputera jako 'maszyny do pisania' jest możliwość pracy nad tekstem – doskonalenie jego formy i treści poprzez wielokrotne modyfikacje, wspomagane edytorem tekstu. Ta cecha komputerowych maszyn do pisania przesądza o ich wyższości nad tradycyjnymi maszynami, chociaż żal jest, że z biurek zniknęły tak piękne, a często majestatyczne maszyny, spotykane dzisiaj w biurach jedynie jako ozdoby.

6. Początki komunikacji

Przodkiem współczesnej komunikacji za pomocą **sieci komputerowych**, była wymiana informacji za pomocą telegrafu, telefonu i telexu, będących wynalazkami z połowy XIX wieku.

Telegraf (z greckiego oznacza *pisać na odległość*) można uznać za pierwsze w pełni elektroniczne medium do komunikacji na odległość. Pierwsze komercyjne systemy telegraficzne zostały opracowane w roku 1837, niezależnie w Anglii i w USA. W Anglii zrobili to znany fizyk **Charles Wheatstone** (1802-1875) i wynalazca **William F. Cooke** (1806-1879), a w USA – wynalazca i artysta **Samuel F.B. Morse** (1791-1872). Morse posłużył się jednym przewodem do przesyłania 'kropek' i 'kresek', za pomocą których były kodowane litery i cyfry zgodnie z **kodem Morse'a,** i w praktyce przyjął się jego system. Genialność tego kodu każe uznać Morse'a za ojca współczesnej kompresji, patrz p. 7.2.

W połowie XIX wieku zaczęto interesować się również przesyłaniem na odległość dźwięków i ludzkiego głosu. 14 lutego 1876 roku niemal w tej samej chwili (w przeciągu jednej godziny!) patent **elektrycznego telefonu** zgłosili dwaj Amerykanie **Alexander G. Bell** (1847-1922) i **Elisha Gray** (1835-1901). Sławny spór o pierwszeństwo wygrał jednak Bell. Impulsem do wynalezienia telefonu była chęć usprawnienia telegrafu tak, aby było możliwe wysyłanie jedną linią telegraficzną wielu wiadomości w tej samej chwili oraz przesyłanie dźwięków i głosu.

Sieć połączeń telefonicznych została również wykorzystana jako sieć elektroniczna dla takich usług, jak: przekazywanie wiadomości tekstowych za pomocą **dalekopisów** (system **telex**, w którym nadawcy i odbiorcy nie muszą kodować i dekodować wiadomości), przesyłanie obrazów za pomocą urządzeń typu **fax**, a pod koniec XX wieku stała się bazą dla komputerowych systemów sieciowych, takich jak **Internet**.

7. Co pozostało po erze urządzeń mechanicznych

Zatrzymujemy się tutaj nad wynalazkami, ideami i pomysłami, które pojawiły się w erze mechanicznych urządzeń do liczenia oraz komunikacji, a dzisiaj można je znaleźć we współczesnej informatyce, czasem w nieco przetworzonej postaci. Omawiamy kolejno: znaczenie logarytmu w algorytmice, kompresowanie informacji metodą podobną do użytej przez Morse'a przy tworzeniu jego alfabetu, oraz układ klawiszy na klawiaturze komputera i fonty w edytorach pochodzące od maszyn do pisania. Te idee i wynalazki nie tylko przetrwały wstrząs technologiczny ery cyfrowej, ale stanowią głęboko zakorzenione elementy współczesnej technologii i informatyki, bez których trudno wyobrazić sobie aktualny stan i dalszy rozwój tych dziedzin.

7.1. Logarytm

Logarytm nie odszedł do lamusa wraz ze zniknięciem suwaków logarytmicznych z biurek inżynierów i naukowców, ale na trwałe pozostał w rozważaniach dotyczących obliczeń. W przeszłości, uzasadnieniem dla posługiwania się logarytmem były własności, które legły u podstaw jego wprowadzenia do obliczeń. Ułatwia bowiem wykonywanie złożonych obliczeń 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.

Rok 1972 to początek agonii suwaków – zaczęły je wypierać kalkulatory elektroniczne stworzone za pomocą... suwaków. Ponad 40 milionów wcześniej wyprodukowanych suwaków stało się nagle bezużytecznych i obecnie stanowią głównie eksponaty kolekcjonerskie, jak te na rysunkach 16 i 17. 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?

Funkcja logarytm nie jest specjalnie lubiana przez uczniów, można ją jednak wprowadzić bardzo intuicyjnie posługując się znaną grą w zgadywanie liczby. Przypuśćmy, że mamy odgadnąć nieznaną liczbę, wybraną spośród 1000 liczb naturalnych z przedziału [1, 1000]: odgadujący podaje swój typ, a ukrywający liczbę – odpowiada, czy podana liczba jest dobra, za duża lub za mała od ukrytej przez niego. Interesujące jest pytanie, ile razy trzeba zapytać, by znaleźć ukrytą liczbę? Okazuje się, że najlepszą strategią jest metoda **dziel i zwyciężaj**, polegająca na podawaniu na każdym kroku liczby leżącej w połowie przedziału pozostałego do przeszukania. Przy takiej strategii, w naszym przykładzie kolejne przedziały zawierają następujące ilości liczb: 1000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1, a więc odgadujący liczbę podaje swoje typy tyle razy, ile razy trzeba podzielić przez 2 liczbę 1000 i kolejne ilorazy z jej dzielenia, by otrzymać przedział o długości 1, czyli szukaną liczbę. Ta ilość pytań jest równa akurat około $\log_2 1000$.

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 5. Zauważmy, że dla liczb, które mają około stu cyfr, wartość logarytmu wynosi **tylko** ok. 333.

Tabela 5.
Wartości funkcji logarytm dla przykładowych argumentów

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

Chociaż powszechnie uważa się Napiera za odkrywcę logarytmu, to jednak ideę prowadzącą do logarytmu można odnaleźć analizując algorytm Euklidesa, odkryty prawie 2500 lat temu. Przypomnijmy, **algorytm Euklidesa** służy do znajdowania największego wspólnego dzielnika (w skrócie NWD) dwóch liczb. Przypuśćmy, że chcemy obliczyć, ile wynosi NWD(n, m) dla dwóch liczb naturalnych n i m, gdzie możemy przyjąć, że $n > m$. Skorzystajmy z oczywistej równości. Jeśli n dzielimy w sposób całkowity przez m, to otrzymujemy całkowity **iloraz** q oraz **resztę** r, która jest mniejsza od dzielnej, czyli od m. A zatem zachodzi równość:

$n = qm + r$, gdzie $0 \le r < m$

Z tej równości wynika, że jeśli jakaś liczba dzieli n i m, to dzieli również m i r. Także na odwrót, liczba, która dzieli m i r dzieli także n i m. Stąd otrzymujemy zależność rekurencyjną:

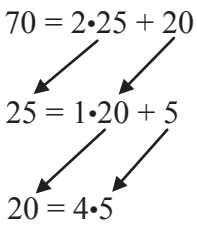
NWD(n, m) = NWD(m, r).

Zatem, NWD dzielnej i dzielnika jest równy NWD dzielnika i reszty. Tę zależność można kontynuować, aż dojdziemy do reszty r równej 0, a wtedy NWD(m, 0) = m, bo 0 jest podzielne przez każdą liczbę.

Wykonajmy proste obliczenia, np. dla NWD(70, 25) otrzymujemy:

NWD(70, 25) = NWD(25, 20) = NWD(20, 5) = NWD(5, 0) = 5,

gdyż obliczenia mają postać:



W tym przykładzie pojawia się ciąg liczb: n = 70, m = 25, r = 20, 5. Dla n = 34 i m = 21 ten ciąg ma postać: 34, 21, 13, 8, 5, 3, 2, 1, czyli NWD(34, 21) = 1, a więc te liczby są względnie pierwsze. Na tych ciągach trudno jest zauważyć pewną regularność, wyjaśnimy ją więc, pomagając sobie rysunkami. Chcemy pokazać, że

reszta r z dzielenia n przez m nie jest większa od połowy liczby n,

czyli każda liczba w tym ciągu nie jest większa, niż połowa liczby stojącej o dwie pozycje wcześniej. W naszym przykładach tak jest rzeczywiście, np. 13 < 34/2 = 17, 8 < 21/2, 5 < 13/2, 3 < 8/2, 2 < 5/2, 1 < 3/2.

A ogólnie rozważmy dwa przypadki, gdy m jest mniejsze od połowy n i większe od połowy n.

Przypadek I. $m \le n/2$.

n: ——————————

m: ———

Jeśli teraz n dzielimy przez m, to może pozostać reszta r, ale nie jest ona większa niż m, a ponieważ założyliśmy, że $m \leq n/2$, więc mamy również $r \leq n/2$.

Przypadek II. $m > n/2$.

n: ————————————

m: ——————————

Jeśli teraz n dzielimy przez m, to w tym przypadku reszta r jest równa $n - m$, a ponieważ $m > n/2$, więc pozostanie mniej niż $n/2$, czyli mamy również $r \leq n/2$.

Ta ilustracja potwierdza, że każda liczba w ciągu generowanym w algorytmie Euklidesa jest przynajmniej dwa razy mniejsza, niż liczba w tym ciągu stojąca o dwie pozycje wcześniej. Przypomina to naszą intuicyjną definicję logarytmu z tą różnicą, że tam każda liczba była połową poprzedniej, a tutaj jest nie większa niż połowa drugiej poprzedniej. To jednak ma tylko taki wpływ, że musimy wykonać co najwyżej dwa razy więcej kroków, by osiągnąć koniec ciągu. Stąd mamy wniosek:

W algorytmie Euklidesa dla obliczenia NWD(n, m), gdzie $n > m$, jest wykonywanych nie więcej niż $2\log_2 n$ iteracji (czyli dzieleń).

Nasuwa się tutaj dygresja, że Euklides był bardzo blisko wynalezienia logarytmu, co zrobiono dopiero prawie dwa tysiąclecia po nim!

Obecnie logarytm jest funkcją powszechnie występującą zwłaszcza w wyrażeniach na złożoność obliczeniową algorytmów, np. w przypadku algorytmów konstruowanych na zasadzie dziel i zwyciężaj, w szczególności w algorytmach połowienia przedziału lub zbioru. Można powiedzieć, że logarytm nie utraci nic ze swojej roli w informatyce dopóki komputery będą się rządziły arytmetyką binarną.

7.2. Kompresja informacji

Zasoby informacji rosną w zawrotnym tempie, równie szybko rosną objętości pojedynczych plików, zwłaszcza przechowujących informacje multimedialne. W tych informacjach jest jednak wiele wolnego (pustego) miejsca, dzięki czemu jest możliwa ich **kompresja** często do ułamka pierwotnej wielkości. Ma to olbrzymie znaczenie w komunikacji. Ale ten problem pojawił się znacznie wcześniej przed erą komunikacji internetowej – człowiek od zawsze starał się komunikować możliwie efektywnie i tworzył w tym celu odpowiednie kody.

Historia kompresji sięga wielu lat przed erą komputerów. Ideę oszczędnego reprezentowania informacji odnajdujemy w połowie XIX wieku, gdy Samuel Morse wynalazł telegraf (patrz p. 6), wtedy 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. W tym alfabecie kodem litery E jest kropka, a kodem litery T jest kreska, gdyż są to dwie najczęściej występujące litery w tekstach w języku angielskim i ogólnie, i inne litery mają tym krótsze kody

im częściej występują one w tekstach. 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 roku 1952 przez **Davida Huffmana** w jego pracy magisterskiej. W tym kodzie również często występujące znaki mają krótkie kody, ale żaden 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 obecnie wykorzystywany w wielu profesjonalnych metodach kompresji tekstu, obrazów i dźwięków, również w połączeniu z innymi metodami, wypada więc uznać Morse'a za ojca współczesnej kompresji informacji.

7.3. Komputer następcą maszyny do pisania

Na koniec tego punktu coś z całkiem innej półki, na której zamiast kalkulatorów stoją maszyny do pisania. Czy ktoś może sobie wyobrazić, że w pewnym momencie zostałby zmieniony układ klawiszy na klawiaturze komputera albo że zaczną one być rozmieszczane w inny sposób? To może być bardzo trudne do zaakceptowania – nasze palce tak przyzwyczaiły się do obecnego układu klawiszy, że nawet zamiana miejscami dwóch znaków na klawiaturze, jak jest w klawiaturach niemieckich (zamiana miejscami liter z i y), powoduje, że te dwie litery są zamieniane w tekście, nawet przy bacznej uwadze piszącego – „pamięć w palcach" dominuje nad zwielokrotnioną uwagą piszącego. Nawet twórcy smartfonów i tabletów nie odważyli się zmienić tego układu przycisków klawiszy klawiatury na ekranach dotykowych.

Układ klawiszy na klawiaturze komputera pochodzi od maszyn do pisania i jest popularnie zwany **QWERTY** – od pięciu kolejnych liter w drugim od góry rzędzie (w trzecim rzędzie w przypadku klawiatury komputera). W tym układzie oddalone są od siebie pary liter (a dokładniej ich czcionek), które często występują obok siebie w słowach języka angielskiego. Układ klawiszy w maszynie do pisania miał zapobiec blokowaniu się czcionek tych liter przy uderzaniu o papier podczas szybkiego pisania wieloma palcami. Podobno jednak w układzie klawiatury QWERTY zrealizowano dodatkowy wymóg, by wszystkie litery dość długiego słowa, jakim jest w języku angielskim TYPEWRITER, znalazły się w jednym rzędzie. Miało to zmniejszyć liczbę ewentualnych pomyłek, jakie

mogli popełnić sprzedający maszyny do pisania, demonstrując swoje produkty zainteresowanym. Produkowano jednak także maszyny, w których klawisze z najczęściej występującymi literami w tekstach były najłatwiej dostępne i znajdowały się blisko klawisza odstępu (patrz rys. 18).

Problem blokowania się czcionek został już we wczesnych maszynach do pisania rozwiązany w jeszcze inny sposób – czcionki wszystkich liter umieszczono na głowicy, co zapobiegało wybraniu dwóch liter jednocześnie. Ponieważ głowice były wymienne, można było umieszczać na nich czcionki różnych krojów. Tak ponad 100 lat temu pojawiły się **fonty**, których dziesiątki mamy dzisiaj do dyspozycji w edytorach tekstu. Jedną z pierwszych maszyn z wymiennymi fontami była **Blickensderfer**, patrz rys. 18, a elektryczne maszyny do pisania z wymiennymi głowicami (np. **IBM Selectric**) zostały wyparte dopiero przez komputerowe edytory tekstu.

Faktycznie więc klawiatura komputera to kopia klawiatury maszyny do pisania, a fonty w elektronicznych edytorach to pomysł przeniesiony z „mechanicznych edytorów" z końca XIX wieku.

Rysunek 18. Maszyna do pisania Blickensderfer (USA, koniec XIX wieku), z wymiennymi głowicami z czcionkami (w drewnianych pudełkach) oraz klawiaturą Morse'a

Przedstawiliśmy w tym punkcie tylko trzy spośród wielu przykładów idei, pomysłów i inwencji z okresu sprzed elektronicznych komputerów, które nie tylko przetrwały kolejne przełomy związane z rozwojem elektroniki, ale stanowią fundamenty współczesnej informatyki i dalej ugruntowują swoją pozycję

i znaczenie. W odniesieniu do logarytmu można sparafrazować słowa z arabskiego powiedzenia[3] mówiąc:

Człowiek boi się czasu,
Lecz czas lęka się logarytmu.

8. Wkład Polaków

Historia polskiej informatyki rozpoczyna się na długo przed pojawieniem się pierwszych komputerów. Wspominamy tutaj o polskich konstrukcjach kalkulatorów z XIX wieku i dwóch osiągnięciach Polaków w XX wieku, które wywarły olbrzymi wpływ na rozwój współczesnej informatyki.

Polskie kalkulatory mechaniczne

W XIX wieku na ziemiach polskich i pod zaborem rosyjskim powstało wiele urządzeń (maszyn) służących do obliczeń, które można zaliczyć do grupy kalkulatorów mechanicznych. Niestety niewiele z tych urządzeń się zachowało, na ogół nie towarzyszyły im zarejestrowane i upublicznione patenty, dzisiaj więc trudno je odtworzyć. Ponadto maszyny te powstały w pojedynczych egzemplarzach, jako prototypy, jednak nikt nie pokusił się o ich zwielokrotnienie. Dzisiaj historycy i pasjonaci starają się odtworzyć te urządzenia z dostępnych informacji. Wymieńmy konstruktorów tych maszyn, więcej szczegółów znajduje się w bogatym serwisie [12].

Jewna Jakobson, mechanik i zegarmistrz, żyjący w XVIII wieku w Nieświeżu (obecnie Białoruś), zbudował, jak się przypuszcza przed rokiem 1770, mechaniczną maszynę, wzorowaną na maszynie Schickarda, która służyła do wykonywania czterech podstawowych działań arytmetycznych. Jedyny egzemplarz tej maszyny znajduje się w Muzeum Nauki w Petersburgu.

Abraham Izrael Stern (1769-1842), zegarmistrz z Hrubieszowa, za namową Stanisława Staszica przeniósł się do Warszawy, gdzie zbudował wiele różnych urządzeń, m.in. maszynę do wykonywania czterech podstawowych operacji (1813), maszynę do wyciągania pierwiastków kwadratowych (1817) i wreszcie maszynę, w której połączył możliwości dwóch pierwszych maszyn. Poza opisem tych maszyn i jedną ilustracją z wystawy w Krakowie, nie odnaleziono żadnych śladów tych urządzeń.

Chaim Zelig Słonimski (1810-1905), matematyk samouk, znawca Talmudu, popularyzator nauki wśród Żydów w Europie Wschodniej, prywatnie – zięć Abrahama Sterna. Najpierw zbudował maszynę do dodawania i odejmowania, a później maszynę do mnożenia. Ta druga maszyna była realizacją jego twier-

[3] To powiedzenie arabskie brzmi: *Człowiek boi się czasu, lecz czas lęka się piramid*.

dzenia z teorii liczb – wykorzystał w niej tabele liczb, wynikające z tego twierdzenia. Maszyna nie zachowała się, ale ostatnio została zbudowana jej replika, patrz rys. 19. Słonimski uzyskał wiele zaszczytów za swoją maszynę do mnożenia.

Izrael Abraham Staffel (1814-1885), zegarmistrz, wynalazca wielu maszyn i instrumentów, w szczególności maszyn do liczenia, za które otrzymał wiele nagród. Zbudował maszynę służącą do wykonywania czterech działań arytmetycznych i wyciągania pierwiastka. W roku 1851 otrzymał za tę maszynę złoty medal na wystawie w Londynie, gdzie oceniono jego dzieło wyżej niż arytmometr Thomasa de Colmar. Maszynę Staffela można oglądać w Muzeum Techniki w Warszawie, patrz rys. 19.

Bruno Abdank-Abakanowicz (1852-1900), matematyk, wynalazca i elektrotechnik, w roku 1878 wynalazł **integraf**, rodzaj integratora, służący do obliczania wartości całek metodą graficzną.

Rysunek 19. Replika maszyny Jakobsona z kolekcji Włodka (Waltera) Szreka (USA), wykonana przez Valérego Monnier (Paryż) i maszyna Staffela (Muzeum Techniki, Warszawa)

Odwrotna notacja polska

Najbardziej znanym osiągnięciem Polaka w informatyce jest **notacja polska**, zainicjowana przez logika **Jana Łukasiewicza** (1876-1956). W tej notacji każde wyrażenie może być jednoznacznie zapisane bez użycia nawiasów. Oparta została na niej **odwrotna notacja polska** (**ONP**; ang. *Reverse Polish Notation – RPN*), wprowadzona przez informatyka z Australii Charlesa Hamblina w połowie lat 50. XX wieku, mająca szerokie zastosowania w informatyce. Notacja polska jest stosowana m.in. w niektórych językach programowania (np. Forth), w języku opisu stron PostScript. Stosowana jest również w kalkulatorach elektronicznych,

takich firm jak Hewlett-Packard czy Sinclair. Niestety nie wyprodukowano w Polsce kalkulatora z polską notacją.

Początki kryptografii komputerowej

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. Wielokrotnie w historii ludzkości szyfrowanie i łamanie szyfrów miało istotny wpływ na bieg wydarzeń. Najbardziej spektakularnym przykładem jest chyba historia rozpracowania niemieckiej maszyny szyfrującej **Enigma** (łac. *zagadka*), 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, patrz [1], [6], [9]. Jednym z rezultatów ich prac była Polska Bomba, urządzenie, które służyło dopasowywania kluczy do szyfrogramów. Anglicy i Amerykanie posłużyli się ich pomysłem w czasie wojny i budowali wielokrotnie większe i efektywniejsze takie bomby, a Anglicy w roku 1943 uruchomili specjalny komputer **Colossus** do łamania szyfrów, który faktycznie był pierwszym komputerem elektronicznym zbudowanym przez człowieka. Można uznać Polskich kryptoanalityków za prekursorów kryptografii komputerowej.

Wcześniej, polscy matematycy Stanisław Leśniewski, Wacław Sierpiński i Stefan Mazurkiewicz złamali sowieckie szyfry w roku 1920 we Lwowie, podczas wojny polsko-radzieckiej. Dzięki temu wiadomo było, co zamierza zrobić Siemion Budionny ze swoimi wojskami. Marian Rejewski uczestniczył w kursach kryptograficznych prowadzonych przez Leśniewskiego.

9. Epilog – rzut oka na historię i przyszłość komputerów

Nurt rozwoju kalkulatorów mechanicznych wniósł niewielki, aczkolwiek istotny wkład w rozwój dzisiejszych komputerów i informatyki. Wymieńmy na zakończenie wydarzenia w historii informatyki o przełomowym znaczeniu dla rozwoju idei, które złożyły się na powstanie współczesnego komputera (polecamy tutaj [5] i [7]).

- Projekt **maszyny analitycznej Charlesa Babbage'a** (1834) z **pierwszym programem** napisanym dla tej maszyny przez **Adę** córkę Bayrona (1843), uznawaną za pierwszą programistkę. Chociaż ta maszyna nie powstała w pełni, to uznaje się ją za pierwowzór współczesnych komputerów. Pomysłu Babbage'a nie wszyscy jednak wynalazcy po nim zauważyli i docenili, a architekturę współczesnych komputerów przypisuje się najczęściej Johnowi von Neumannowi.

- **Mechanograficzny system tabulacyjny Hermana Holleritha**, dzięki któremu było możliwe zliczanie wyników spisów powszechnych w USA i w innych krajach (m.in. w Rosji) od końca XIX wieku. Zapoczątkowana została w ten sposób automatyzacja przetwarzania dużych ilości danych. Specjalizowała się w tym na początku swojego istnienia firma **IBM**, wyrosła w roku 1924 na firmach tworzonych przez Holleritha.
- **Praca magisterska Claude E. Shannona** (1938), uznawana za najdonioślejszą pracę magisterską XX wieku, dotycząca wykorzystania **algebry Boole'a** do analizy i syntezy układów przełączających i binarnych.
- **Komputery Konrada Zuse**, z wykształcenia inżyniera budownictwa, który w czasach kryzysu materiałowego podczas II wojny światowej i tuż po niej, stosował w swoich komputerach (Z4) mechaniczny bit! Jego konstrukcje komputerów, a zwłaszcza idee i pomysły (języka programowania – 1946, komputera równoległego – 1958, gridu – 1970) wyprzedziły o wiele lat podobne wynalazki i ich realizacje, zasłużenie więc uważa się go za ojca współczesnych komputerów.
- Fundamentalne dla teorii obliczalności prace **Alana Turinga** (1936), a także jego wkład do badań nad sztuczną inteligencją ery komputerowej (Test Turinga, 1950), więcej na te temat można przeczytać w rozdziale Jarosława Grytczuka, polecamy też [2].
- Pierwsze komputery elektroniczne – komputer **ABC** Johna Atanasoffa (1942), komputery **Colossus**, budowane w Wielkiej Brytanii od roku 1943 na potrzeby kryptologów i rozmontowane na polecenie Winstona Churchilla bezpośrednio po zakończeniu wojny, **ENIAC** – ukończony w roku 1946 i inne, jak EDSAC (1949), ACE (Wielka Brytania, 1950), Harvard MARK III (1950), SEAC (1950), EDVAC (1951), IBM 701 (1953). Pod koniec lat 40. XX, Thomas J. Watson Jr., szef firmy IBM, miał podobno wątpić, by ludzkość potrzebowała kiedykolwiek więcej niż 5 dużych komputerów!
- Fundamentalne dla ery elektronicznej wynalazki: **tranzystor** (John Bardeen, Walter H. Brattain, William B. Shockley – 1948; Nagroda Nobla dla wynalazców w 1952 roku) i **układ scalony – chip** (Jack S.C. Kilby, Robert Noyce – 1958; Nagroda Nobla dla Kilby'ego w 2000 roku).
- Rozwój **Internetu** – ARPANET (1969), poczta elektroniczna (1972), protokół TCP/IP (1983), serwis WWW (1991).

Wszystkie te wynalazki i dokonania wywarły wpływ na współczesny stan informatyki. Teoretyczne podstawy budowy komputerów (Boole i Shannon) i teoretyczne podstawy obliczalności (Turing) stworzyły solidną bazę dla rozwoju tej dziedziny. Z kolei odkrycia i innowacje w elektronice (tranzystor, chip) doprowadziły do miniaturyzacji komputerów, co spowodowało jednocześnie znaczne ich przyspieszenie i zwiększenie możliwości. Mikroprocesor jest sercem komputerów osobistych, jak i – zwielokrotniony – superkomputerów.

A jaka będzie przyszłość?

Technologia komputerowa rozwija się nieustannie i w olbrzymim tempie. Pojawiają się nowe idee i gadżety, które powoli zmieniają nasze życie niemal we wszystkich sferach. Jest to jednak tylko ewolucja, której efekty można przewidywać, a która nie burzy i nie rewolucjonizuje zachowania jednostki i stosunków społecznych, odsyłając do lamusa dotychczasowe rozwiązania, uznawane za tradycyjne.

Można sobie jednak wyobrazić, że technologia będzie coraz bardziej integrować się nie tylko z tradycyjnymi czynnościami człowieka na zasadzie ich wspierania, ale że w pewnym momencie całkowicie przejmie wykonywanie wybranych czynności. Znane już są tego przykłady, np. protezy uruchamiane impulsami z mózgu, często odruchowo, bez angażowania działań z pełną świadomością. Można sobie wyobrazić, że tak się może stać z czytaniem – odpowiedni chip, zainstalowany w kąciku oka będzie odbierał sygnały („czytał" je) przychodzące do oka i przekazywał bezpośrednio do mózgu. Te sygnały mogą pochodzić nie tylko z ekranu przed naszymi oczyma, czy z otwartej książki, ale mogą być wysyłane bezpośrednio do chipa z globalnej biblioteki światowych zasobów informacji. Na przeszkodzie temu dopełnianiu mózgu informacjami może stanąć jednak natura człowiek – Stanisław Lem powoływał się na badania, które uzasadniały, że obecnie człowiek wcale więcej nie absorbuje informacji, niż robił to na przykład w starożytności. Ale przecież ten chip może mieć podręczną pamięć, która będzie zapełniana na wszelki wypadek i szybko dostępna dla jego właściciela, nawet nieświadomie, w reakcji na impuls z mózgu. Podobny los może spotkać inne czynności człowieka, odruchy bezwarunkowe i warunkowe, a także te z pogranicza myślenia i świadomości.

Literatura

1. Grajek M., *ENIGMA. Bliżej prawdy*, Rebis, Poznań 2007
2. Hodges A., *Enigma. Życie i śmierć Alana Turinga*, Prószyński i S-ka, Warszawa 2002 (1983 oryginalne wydanie), http://www.turing.org.uk/
3. Ifrah G., *Dzieje liczby, czyli historia wielkiego wynalazku*, Ossolineum, Wrocław 1990
4. Ifrah G, *Historia powszechna cyfr*, tomy I i II, W.A.B., Warszawa 2006
5. Kaufmann H., *Dzieje komputerów*, PWN, Warszawa 1980
6. Kozaczuk W., *W kręgu Enigmy*, Książka i Wiedza, Warszawa 1986
7. Lingonnière R., *Prehistoria i historia komputerów*, Ossolineum, Wrocław 1992
8. Madey J., Sysło M.M., *Początki informatyki w Polsce*, „Informatyka" 2000, nr 9 i 10, http://www.mmsyslo.pl/Historia/Artykuly-i-prezentacje/Artykuly-maszyny/
9. Singh S., *Księga szyfrów*, Albatros, Warszawa 2001
10. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998, dostępna pod adresem: http://www.mmsyslo.pl/Materialy/Ksiazki-i-podreczniki/Ksiazki

11. Sysło M.M., *Historia rachowania – ludzie, idee, maszyny. Historia komputerów i informatyki w zarysie, Plansze*, WSiP, Warszawa 2006, http://www.mmsyslo.pl/Historia/Plansze-z-historii-informatyki
12. Zalewski J. (Opiekun projektu), Polish Contributions to Computing, http://chc60.fgcu.edu/

Prof. dr hab. Maciej M. Sysło

Na studia matematyczne trafił do Uniwersytetu Wrocławskiego, gdy instalowano tam jedną z trzech pierwszych w Polsce maszyn matematycznych seryjnej produkcji, angielską maszynę Elliott 803. Obok w Elwro powstawały polskie komputery i też obok – odbywały się pierwsze zajęcia z informatyki w szkołach. Wtedy jeszcze, komputer nie nazywał się komputerem, a informatyka – informatyką.

Był obserwatorem, a później uczestnikiem niemal całej historii informatyki w Polsce. Poza krajem, miał szczęście spotkać na swojej drodze wiele osobistości ze świata początków komputerów, jak Billa Tutte'a – współpracownika Alana Turinga z Bletchley Park, gdzie w czasie wojny wspólnie łamali niemieckie szyfrogramy, czy George'a Danziga – twórcę metody simpleks, uznawanej dzisiaj za najważniejszy algorytm w erze obliczeń komputerowych. Tak zrodziły się zainteresowania historią myśli ludzkiej, historią odkryć, historią informatyki.

W pierwszym polskim podręczniku do informatyki z roku 1988, napisał rozdział o historii informatyki, jako tło dla rozważań informatycznych początków ery komputerów osobistych. Z czasem zaczął interesować się wczesnymi konstrukcjami maszyn do rachowania i rozpoczął ich zbieranie. Obecnie jego kolekcja liczy kilkaset sztuk różnych urządzeń. Kolekcjonuje również maszyny do pisania. Zaprojektował plansze historyczne pt. *Historia komputerów: ludzie, idee, maszyny*, organizuje wystawy swoich maszyn, opowiada o historii maszyn i informatyki na spotkaniach w szkołach, uczelniach i dla zainteresowanych osób.

syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl
http://mmsyslo.pl/

Addiator 282
al-Chorezmi 270, 275
alfabet Morse'a 269, 295-296, 299-301
alfabetyzacja komputerowa 211-212
algebra relacji 133
algorytm 275
algorytm Dijkstry 86-88
algorytm Euklidesa 102-103, 274-275, 297
algorytm wielomianowy 104
architektura usługowa SOA 158-159
arytmometr 285

baza danych 118
baza danych, dokumentowa 142
baza danych, grafowa 143
baza danych, hierarchiczna 119
baza danych, klucz-wartość 141
baza danych, obiektowa 121
baza danych, relacyjna 123
baza danych, sieciowa 119
baza danych, strumieniowa 140
baza wiedzy 13
baza danych NOSQL 123, 141
bęben schodkowy 285
biegłość komputerowa 212
BNF 61
Brunsviga 287
BYOD 223

chmura edukacyjna 220, 227
Curta 286
człowiek Turinga 201

Dijkstra Edsger 64, 68, 86
dowód nie wprost 16

edukacja 199
edukacja informatyczna 202, 245
EDVAC 51, 268, 305
e-książka 225
elektroniczna gospodarka 149
elektroniczny biznes 149-150
elektroniczny handel 150
elektroniczny rekord pacjenta 169, 191
ENIAC 51, 268, 305
epoka Gutenberga 199+200
era obrazu 200
e-szkoła 219-220

faktoryzacja 105-106
Felix 288
Fibonacci 270
font 301

GNU/Linux 66
Gödel Kurt 11, 111-112
gramatyka 47, 62

haszowanie 78, 136
hipoteza Poincarégo 109
historia informatyki 200, 268

implikacja 10
informacja elektroniczna 150
informatyka 204
instrukcja 59
inżynieria biomedyczna 171

język Algol 60
język BNF 62
język COBOL 64
język Datalog 31
język deklaratywny 25
język Fortran 59
język funkcyjny 11, 25, 72
język imperatywny 11
język maszynowy 49
język mnemoniczny 52
język obiektowy 72
język Pascal 46
język Plankalkül 53
język pomocniczy 73
język powłoki 72
język programowania 39, 65, 73-74
język Prolog 28
język przetwarzania danych 72
język regułowy27, 36
język skryptowy 72
język SQL 31, 72, 120-121
język zapytań 26, 31, 121

kalkulator mechaniczny 277
kariera w ICT 231
klasa problemów NP 107
klasa problemów P 104
klasyczny rachunek zdań 14
klauzula 17, 61
klucz 125
kod Huffmana 300
koło zębate 287
kompetencje XXI wieku 217
kompresja 299
komputyka (computing) 205
komunikacja elektroniczna 152
kopiec Fibonacciego 88, 92
kształcenie informatyczne 202, 245

kubełki 78, 89
kwadratura koła 105-106

Leibniz, Gottfried 105, 267-268, 285
leksyka 47
liczba 41-43, 269-271
liczba pierwsza 104
liczba Mersenne'a 106
liczydło 272
liczydło, abak 271-172
liczydło, schoty 272
liczydło, soroban 272
liczydło, suan-pan 272
lista 30
logarytm 290, 296-297, 302
logika 10

Łukasiewicz Jan 303

maszyna do pisania 294
maszyna Schickarda 278
maszyna Turinga 112
McLuhan Marshall 199
mechanizm licznikowy 280
metoda resolucji 19
metoda wstępująca 32
Minsky Martin 40
mobilna edukacja 218
model 11
model zmian w edukacji 207
Morse Samuel F.B. 295
myślenie algorytmiczne 220
myślenie komputacyjne 212

największy wspólny dzielnik (NWD) 102, 297-299
Napier John 8, 267, 273, 278, 290
nauczanie programowane 202
NP-zupełność 110

odwrotna notacja polska (ONP) 303
odwrócona klasa 224
optymalizator 134
organizacja wirtualna 155-157
outreach 211

pałeczki Napiera 273-274, 278
Pascal, Blaise 267, 280
Pascalina 290
pętla 67, 70
platforma edukacyjna 219
problem, czy P = NP.? 115
problem NP-zupełny 110
problem SAT 108
problem stopu 113
Problem Milenijny 108
program 49
program powłoki (shell) 65
programowanie strukturalne 68
programowanie w logice 24
przetwarzanie w chmurze 159, 259

QWERTY 300

reguła 10, 18
rekursja 28
rozkaz 59
rozkaz skoku (goto) 66, 68-70
rynek pracy ICT 256

semantyka 14, 47
sortowanie kubełkowe 78
spełnialność (SAT) 17
spójność bazy danych 130
strategia 1:1 220-222
sudoku 96
sumator 281
suwak logarytmiczny 291
system operacyjny 65
system OSZBD 118
system SZBD 120

tabela 123
tablica logiczna 14
tautologia 15
technologia 199
technologie ICT 233
technologia informacyjno-komunikacyjna (TIK) 205
technopol 201
telegraf 293-295
telemedycyna 172, 189
telepraca 150
tomograf 174-175
transakcja 130
Turing Alan 11, 112-114, 268, 305

Unix 65

von Neumann John (Janos) 11, 51, 54, 114

Wirth Niklaus 24, 37, 46, 74
wykluczenie informacyjne 228
wyszukiwanie AND 23
wyszukiwanie OR 24

zakleszczenie 131
zawód informatyk 252
Zuse Konrad 53-54, 268, 305