informatyka+

Algorytmika i programowanie

Bazy danych

Multimedia, grafika i technologie internetowe

Sieci komputerowe

Tendencje w rozwoju informatyki i jej zastosowań

*Człowiek – najlepsza inwestycja*

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

informatyka+

# Wszechnica Popołudniowa: Algorytmika i programowanie

## Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew

*Maciej M Sysło*

*Człowiek – najlepsza inwestycja*

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

# Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew

**Rodzaj zajęć:** Wszechnica Popołudniowa
**Tytuł:** Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew
**Autor:** prof. dr hab. Maciej M Sysło

**Redaktor merytoryczny:** prof. dr hab. Maciej M Sysło

Zeszyt dydaktyczny opracowany w ramach projektu edukacyjnego **Informatyka+** — ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych w zakresie technologii informacyjno-komunikacyjnych (ICT).
**www.informatykaplus.edu.pl**
**kontakt@informatykaplus.edu.pl**

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

Projekt graficzny: FRYCZ I WICHA

Warszawa 2010


KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

# Znajdowanie najkrótszych dróg oraz najniższych i najkrótszych drzew

**Maciej M. Sysło**
Uniwersytet Wrocławski, UMK w Toruniu
syslo@ii.uni.wroc.pl, syslo@mat.uni.torun.pl

**Streszczenie**

Wykład jest poświęcony elementom teorii grafów i obliczeń na grafach. Grafy odgrywają podwójną rolę w informatyce. Z jednej strony, są modelami obliczeń – w tej roli najczęściej występują drzewa – lub odzwierciedlają strukturę połączeń komunikacyjnych, a z drugiej – wiele problemów o praktycznych zastosowaniach jest definiowanych na grafach jako strukturach połączeń (zależności) między elementami. W pierwszej części wykładu zostaną przedstawione problemy, które miały wpływ na rozwój teorii grafów oraz dużą ich popularność. Druga część jest poświęcona wykorzystaniu drzew jako schematów obliczeń i algorytmów, a w trzeciej części zostaną przedstawione klasyczne problemy obliczeniowe na grafach, takie jak: znajdowanie najkrótszych dróg i znajdowanie najkrótszego drzewa rozpinającego w grafie z obciążonymi połączeniami. Rozwiązania problemów będą prezentowane w specjalnym oprogramowaniu.

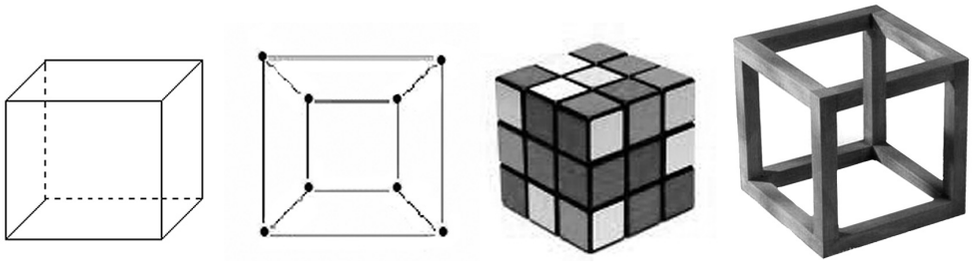
**Spis treści**

## 1. GRAFY JAKO MODELE RÓŻNYCH SYTUACJI

Ten wykład jest poświęcony grafom i ich zastosowaniom. W potocznym sensie grafem jest pewien zbiór elementów, między którymi istnieją połączenia. Na przykład, miasta i sieć połączeń między nimi tworzą graf. Połączeniami z kolei mogą być drogi dla samochodów, sieć połączeń kolejowych lub sieć połączeń lotniczych. Na rysunku 1 są przedstawione fragmenty połączeń lotniczych. W rozważaniach informatycznych (algorytmicznych) na ogół nie interesuje nas, czemu odpowiadają elementy w takich sieciach połączeń (miasta, stacje, lotniska) i jaki jest charakter połączeń (drogowy, kolejowy, lotniczy). Elementy w takich sieciach połączeń nazywamy **wierzchołkami**, połączenia – **krawędziami** (jeśli są symetryczne, czyli nieskierowane) lub **łukami** (jeśli są skierowane), a sieć – **grafem** (gdy zawiera symetryczne połączenia) lub **digrafem** (gdy zawiera skierowane połączenia). Graf jest **modelem** sytuacji, w której mamy zbiór elementów i połączenia między nimi.


*Rysunek 1.*
Przykład sieci połączeń lotniczych z Bydgoszczy

Na rysunku 2 przedstawiono inny przykład pojawiania się grafów. Jest on związany z Eulerem, który rozważał siatki wielościanów, czyli grafy utworzone z wierzchołków i krawędzi wielościanów – stąd pochodzą nazwy wierzchołek i krawędź.



*Rysunek 2.*
Sześcian i graf jego siatki. Wierzchołki sześcianu są wierzchołkami grafu, a krawędzie sześcianu są krawędziami grafu. Dwa inne przykłady sześcianów są pokazane obok

Elementarnym wprowadzeniem do teorii grafów jest książka [8]. Zastosowania grafów w informatyce zilustrowano w książkach [2], [5], [6]. Grafy związane z zagadkami można znaleźć w [6]. Głębsze rozważania na temat grafów i ich algorytmów zamieszczono w książkach [1], [3], [7].

W rozdziale 2 przytaczamy najbardziej znane przykłady związane z grafami, które przyczyniły się do ich popularności i wzrostu zainteresowania nimi. W rozdziale 3 ilustrujemy posłużenie się drzewami – specjalnymi rodzajami grafów, które mają duże zastosowania w informatyce, a dokładniej – w konstrukcji i analizie algorytmów. Natomiast w rozdziale 4 przedstawiamy kilka problemów i algorytmów związanych z dowolnymi gra-

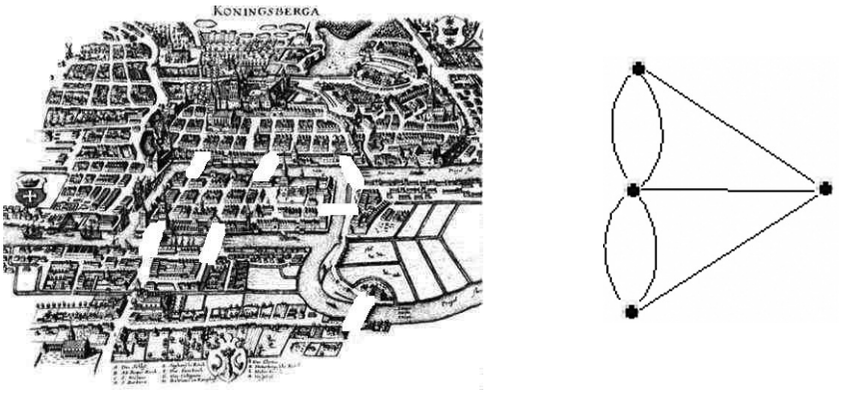
fami. Rozważania na wykładzie będą ilustrowane za pomocą oprogramowania edukacyjnego, które służy do demonstracji działania algorytmów na grafach i ich komputerowych realizacji.

## 2 TRZY KLASYCZNE ZAGADNIENIA

Przedstawiamy tutaj trzy przykłady klasycznych zagadnień, które uważa się, że najbardziej przyczyniły się do olbrzymiej popularności grafów jako modeli różnych sytuacji.

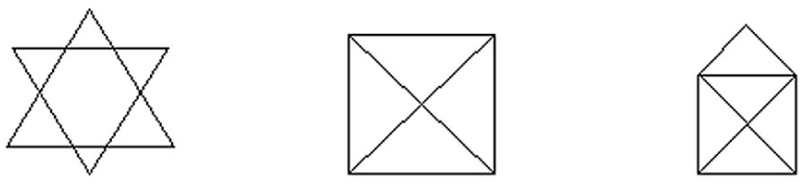
### 2.1 GRAFY EULERA

Wspomniany już wcześniej Leonhard Euler (1707-1783) jest uważany za ojca teorii grafów. W roku 1736, gdy mieszkał w obecnym Kaliningradzie (Rosja), zainteresował się, czy można zorganizować wycieczkę po tym mieście w taki sposób, aby przejść każdy most na rzece Pregoła, każdy dokładnie jeden raz. Zagadka ta nosi nazwę **mostów królewieckich**, gdyż Kaliningrad przez jakiś czas od połowy XV wieku nosił nazwę Królewiec; za czasów Eulera zaś nazywał się Königsberg. Na rysunku 3, na starej mapie tego miasta z czasów Eulera, naniesiono wspomniane w zagadce mosty, było ich siedem.




*Rysunek 3.*
Rzeka Pregoła w Königsberg z zaznaczonymi na niej mostami (białe grube kreski), a obok graf (dokładniej – multigraf) odpowiadający tej sytuacji

Zagadkę Eulera można przetłumaczyć na język innej zagadki, w której pytamy, czy daną figurę można narysować na kartce papieru bez odrywania ołówka przechodząc każdą linię dokładnie jeden raz. Taka figura nazywa się **jednobieżną** lub **unikursalną**. Łatwo zauważyć, że w każdym wierzchołku takiej figury liczba połączeń, którymi wchodzimy do wierzchołka musi być równa liczbie połączeń, którymi wychodzimy z tego wierzchołka, a więc liczba połączeń w każdym wierzchołku – tę liczbę nazywamy **stopniem wierzchołka** – musi być parzysta, jeśli mamy powrócić do punktu startu, lub graf może zawierać dokładnie tylko dwa wierzchołki o stopniach nieparzystych, wtedy droga zawierająca wszystkie połączenia musi się zaczynać i kończyć w tych wierzchołkach. W pierwszym przypadku, graf zawiera **cykl Eulera**, a w drugim – **drogę Eulera**. Graf zawierający cykl Eulera nazywamy **grafem Eulera**. Łatwo spostrzec, że graf mostów królewieckich na rys. 3 nie jest jednobieżny, gdyż wszystkie jego cztery wierzchołki mają stopnie nieparzyste.



*Rysunek 4.*
Odpowiedz, która z figur jest jednobieżna?

## 2.2 MALOWANIE MAP

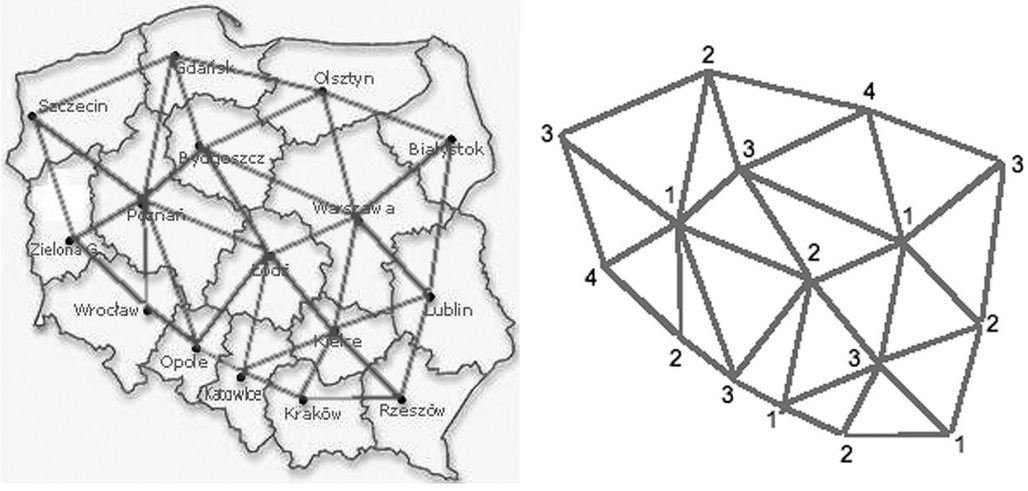
W połowie XIX wieku w Anglii duże zainteresowanie wzbudził **Problem Czerech Kolorów**, który przez ponad sto lat był **Przypuszczeniem Czterech Kolorów**. Ten problem, jak większość problemów dotyczących grafów, ma bardzo proste sformułowanie. Bierzemy mapę na przykład mapę Polski z podziałem administracyjnym i chcemy tak pomalować województwa kolorami, by żadne dwa sąsiadujące ze sobą województwa nie zostały pomalowane tym samym kolorem, staramy się przy tym użyć możliwie jak najmniejszej liczby kolorów. Francis Guthrie, student De Morgana, w 1852 roku postawił przypuszczenie, że zawsze cztery kolory wystarczą.

Związek Problemu Czterech Kolorów z grafami jest zilustrowany na rysunku 5. Najpierw tworzymy graf, który będzie odzwierciedlał sąsiedztwo województw. A zatem wierzchołki w tym grafie mogą odpowiadać stolicom województw i dwa wierzchołki połączymy krawędzią, jeśli odpowiadające im województwa bezpośrednio graniczą ze sobą. Zauważmy, że w tak utworzonym grafie, żadne dwie krawędzie nie przecinają się poza wierzchołkami – graf, który można tak narysować, nazywa się **grafem planarnym**, a jego rysunek – **grafem płaskim**. Specyfikacja Problemu Czterech Kolorów dla grafów ma teraz następującą postać:

**Problem Czterech Kolorów dla grafów**

*Dane:* Graf planarny *G*.

*Wynik:* Pomalowanie wierzchołków grafu *G* co najwyżej 4 kolorami w taki sposób, że żadne dwa wierzchołki połączone krawędzią, nie zostały pomalowane tym samym kolorem.




*Rysunek 5.*
Mapa polski z podziałem na województwa. Na mapie naniesiono połączenia odpowiadające sąsiedztwu województw. Obok graf województw i jego pokolorowanie czterema kolorami (kolory są oznaczone liczbami 1, 2, 3, 4)

Wróćmy do malowania grafu województw (patrz rys. 5). Wierzchołki tego grafu pomalowaliśmy 4 kolorami (oznaczonymi liczbami 1, 2, 3, 4) stosując algorytm, w którym wierzchołki są malowane w kolejności stopni, od największego i malowanemu wierzchołkowi dajemy kolor o możliwie najmniejszej liczbie. W taki sposób, najpierw Poznań (sąsiaduje z 7 województwami) otrzymał kolor 1, później Łódź (sąsiaduje z 6 województwami) otrzymała kolor 2, następnie Warszawa otrzymała kolor 1, Kielce – kolor 3, Bydgoszcz – kolor 3, Gdańsk – kolor 2, Olsztyn – kolor 4, Lublin – kolor 2, Katowice – kolor 1, Opole – kolor 3, Szczecin – kolor 3, Białystok – kolor 3, Rzeszów – kolor 1, Kraków – kolor 2, Wrocław – kolor 2, Zielona Góra – kolor 4. Jako ćwiczenie proponujemy sprawdzenie, czy ten graf nie można pomalować trzema kolorami.

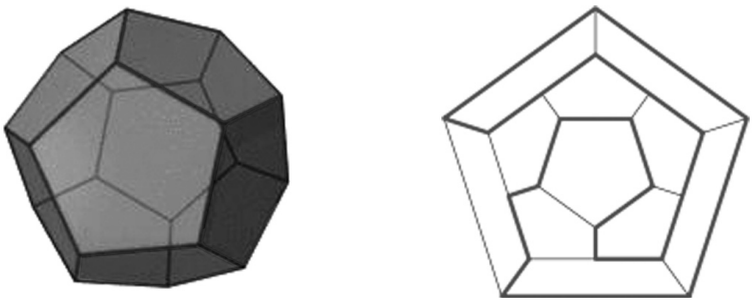
Zastosowany przez nas algorytm malowania grafu to **algorytm sekwencyjny**, który może być stosowany do malowania wierzchołków dowolnego grafu. Stosowana jest w nim **strategia zachłanna** – na każdym kroku jest używany możliwie najmniejszy kolor. Problem kolorowania grafów jest bardzo trudny algorytmicznie – więcej szczegółów na temat tego problemu można znaleźć w książkach [8] i [7].

Wróćmy jeszcze do historii Problemu Czterech Kolorów. W 1879 roku Alfred B. Kempe opublikował dowód, że każdy graf planarny można pomalować co najwyżej 4 kolorami. Jednak w 1890 roku Percy J. Heawood znalazł błąd w dowodzie Kempego, ale był w stanie udowodnić, że 5 kolorów wystarczy dla grafów planarnych (dowód jest bardzo prosty – patrz [8]). Przez niemal 100 lat tym problemem interesowali się niemal wszyscy matematycy, rozwijając wiele różnych metod, aż dopiero w 1976 roku Przypuszczenie Czterech Kolorów zostało potwierdzone przez ... **komputer**. Dokonali tego Kenneth Appel i Wolfgang Haken przy współpracy z programistą Johnem Kochem. Komputer pracował przez 1200 godzin. Dotychczas nie udało się podać dowodu tego twierdzenia, który nie korzystałby z komputera.

## 2.3 GRAFY HAMILTONA I PROBLEM KOMIWOJAŻERA

Bardzo podobnie do problemu Eulera z punktu 2.1 brzmi problem postawiony w 1859 roku przez Williama R. Hamiltona (1805-1865). W tym przypadku, zagadka dotyczyła dwunastościanu formalnego i należało znaleźć drogę zamkniętą przechodzącą po krawędziach tej bryły, która zawiera każdy jej wierzchołek dokładnie jeden raz (patrz rys. 6). Taką drogę nazywamy **cyklem Hamiltona**, a graf zawierający taki cykl – **grafem Hamiltona**.

**Problem Cyklu Hamiltona**
*Dane:* Dowolny *G*.
*Wynik:* Znaleźć cykl Hamiltona w grafie *G* albo stwierdzić, że takiego cyklu nie ma w grafie *G*.



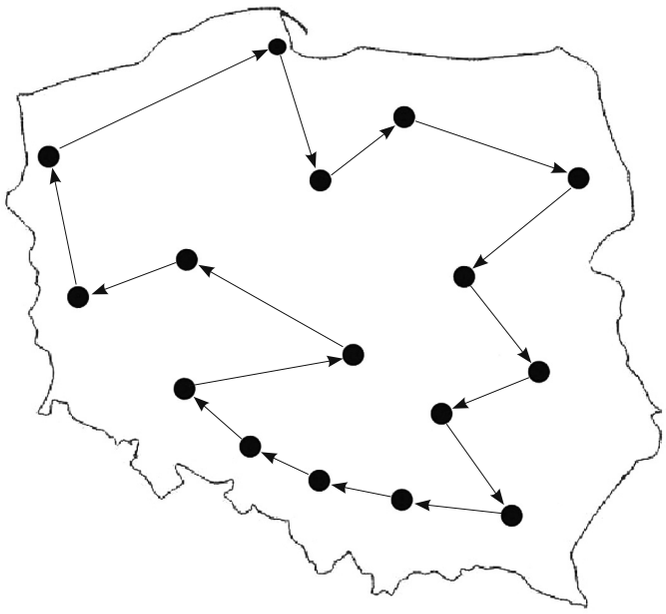
*Rysunek 6.*
Dwunastościan foremny i jego graf z zaznaczonym cyklem Hamiltona

Różnica między problemami Eulera i Hamiltona tkwi w tym, że w pierwszym przypadku poszukujemy cyklu zawierającego wszystkie połączenia, każde dokładnie raz, a w drugim – cyklu zawierającego wszystkie wierzchołki, również każdy dokładnie raz. Różnica wydaje się być niewielka, jednak obliczeniowo te dwa problemy należą do diametralnie różnych klas złożoności. Problem Eulera ma łatwe rozwiązanie – wystarczy sprawdzić, czy stopnie wszystkich wierzchołków są parzyste i czy graf jest spójny (tzn. jest w „jednym kawałku"). Natomiast dla problemu Hamiltona nie podano dotychczas efektywnego algorytmu rozwiązywania. Dalej w tym punkcie zajmiemy się następującą, praktyczną wersją problemu Hamiltona.

**Problem komiwojażera (TSP)**
*Dane:* *n* miast (punktów) i odległości między każdą parą miast.
*Wyniki:* Trasa zamknięta, przechodząca przez każde miasto dokładnie jeden raz, której długość jest możliwie najmniejsza.

Przykładem zastosowania problemu komiwojażera (**TSP** – *Travelling Salesman Problem*) może być zadanie wyznaczenia najkrótszej trasy objazdu prezydenta kraju po wszystkich stolicach województw (stanów – w Stanach zjednoczonych, landów – w Niemczech itp.). Na tej trasie, prezydent wyjeżdża ze stolicy kraju, ma odwiedzić stolicę każdego województwa dokładnie jeden raz i wrócić do stolicy kraju.

Na rysunku 7 przedstawiliśmy jedną z możliwych tras dla stolic województw, ale nie jesteśmy pewni, czy jest ona najkrótsza. Obsługa biura prezydenta może jednak chcieć znaleźć najkrótszą trasę. W tym celu postanowiono generować wszystkie możliwe trasy – zastanówmy się, ile ich jest. To łatwo policzyć. Z Warszawy można się udać do jednego z 15 miast wojewódzkich. Będąc w pierwszym wybranym mieście, do wyboru mamy



*Rysunek 7.*
Przykładowa trasa przejazdu prezydenta po stolicach województw

jedno z 14 miast. Po wybraniu drugiego miasta na trasie, kolejne miasto można wybrać spośród 13 miast i tak dalej. Gdy osiągamy ostatnie miasto, to czeka nas tylko powrót do Warszawy. A zatem wszystkich możliwych wyborów jest: 15*14*13*...*2*1. Oznaczmy tę liczbę następująco:

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

a ogólnie

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

Oznaczenie $n!$ czytamy **„n silnia”**, a zatem $n!$ jest iloczynem kolejnych liczb całkowitych, od jeden do $n$. Wartości tej funkcji dla kolejnych $n$ rosną bardzo szybko, patrz tabela 1.

*Tabela 1.*
Wartości funkcji $n!$

| $n$ | $n!$ |
|---|---|
| 10 | 3628800 |
| 15 | $1.30767*10^{12}$ |
| 20 | $2.4329*10^{18}$ |
| 25 | $1.55112*10^{25}$ |
| 30 | $2.65253*10^{32}$ |
| 40 | $8.15915*10^{47}$ |
| 48 | $1.24139*10^{61}$ |
| 100 | $9.3326*10^{157}$ |

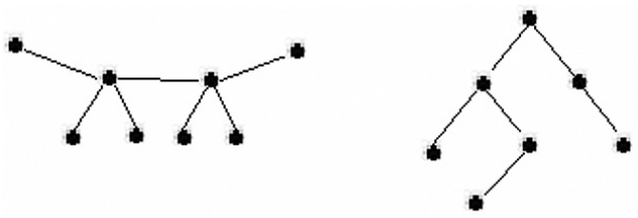
Z wartości umieszczonych w tabeli 1 wynika, że posługując się superkomputerem o mocy 1 PFlops, czyli wykonującym $10^{15}$ operacji na sekundę, w realizacji naszkicowanej metody, służącej do znalezienia najkrótszej trasy dla prezydenta, otrzymanie takiej trasy w przypadku Polski zabrałoby mniej niż sekundę. Jednak w olbrzymim kłopocie znajdzie się prezydent Stanów Zjednoczonych chcąc taką samą metodą znaleźć najkrótszą trasę objazdu po stolicach wszystkich kontynentalnych stanów (jest ich 49, z wyjątkiem Hawajów) trwałoby to $3.9*10^{38}$ lat.

Znane są metody rozwiązywania problemu komiwojażera szybsze niż naszkicowana powyżej, jednak problem TSP pozostaje bardzo trudny. W takich przypadkach często są stosowane metody, które służą do szybkiego znajdowania rozwiązań przybliżonych, nie koniecznie najkrótszych. Jedna z takich metod, zwa-

na **metodą najbliższego sąsiada**, polega na przejeżdżaniu w każdym kroku do miasta, które znajduje się najbliżej miasta, w którym się znajdujemy – jest to typowe podejście zachłanne, gdyż na każdym kroku chcemy wykonywać możliwie najlepszy ruch. W rozwiązaniu naszego problemu tą metodą, pierwszym odwiedzonym miastem powinna być Łódź, później Kielce, Kraków, Katowice, ... Niestety, trasa otrzymana metodą najbliższego sąsiada nie jest krótsza niż trasa naszkicowana na rys. 7.

## 3 PRZYKŁADY ZASTOSOWAŃ GRAFÓW W INFORMATYCE

W tym rozdziale przedstawimy wybrane zastosowania grafów w informatyce. Głównie tymi grafami będą drzewa. **Drzewo** jest grafem, który **nie zawiera cyklu**, czyli drogi zamkniętej, i jest **spójny**, czyli jest w „jednym kawałku". Ze spójności wynika, że każde dwa wierzchołki w drzewie są połączone drogą, a z braku cyklu – że dla każdych dwóch wierzchołków istnieje dokładnie jedna droga, która je łączy. Na ogół w zastosowaniach informatycznych drzew, drzewa mają wyróżniony wierzchołek, który nazywamy **korzeniem**. Na rysunku 8 są przedstawione przykładowe drzewa.

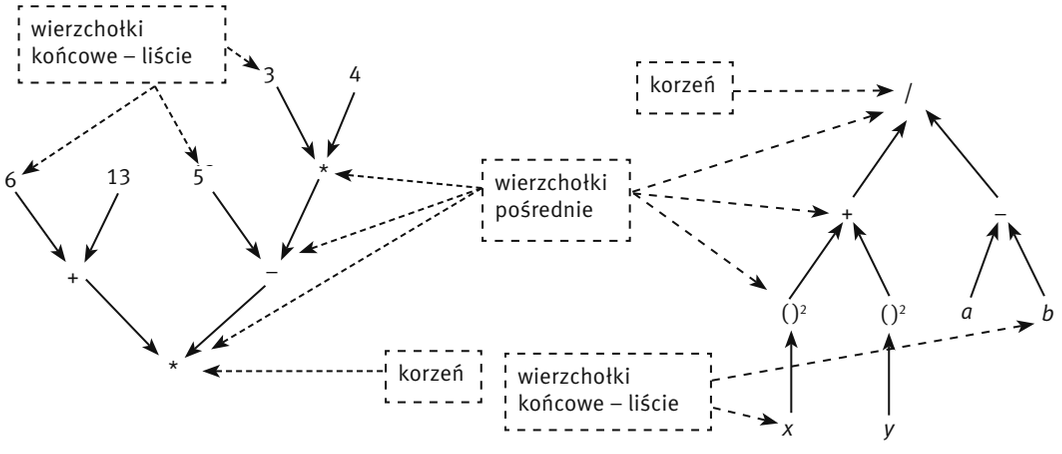


Rysunek 8.
Przykładowe drzewa

W tym rozdziale przedstawimy trzy proste zastosowania drzew. W punkcie 3.1.omówimy krótko reprezentowanie wyrażeń na drzewie. Następnie wykorzystamy drzewa do reprezentowania algorytmów (punkt 3.2) i na końcu (punkt 3.3) naszkicujemy metodę tworzenia krótkich kodów.

### 3.1 DRZEWA WYRAŻEŃ

Na początku nauki matematyki w szkole podstawowej spotkaliście się z drzewem jako schematem obliczania wartości wyrażenia. W takim drzewie, zwanym **drzewem wyrażenia**, argumenty wyrażenia (liczby lub zmienne) znajdują się w **wierzchołkach końcowych** (zwanych **wiszącymi** a także **liśćmi**), a działania są umieszczone w **wierzchołkach pośrednich**. Przykłady takich drzew są pokazane na rys. 9. Wyróżniony wierzchołek drzewa nazywa się **korzeniem**. W tekstach informatycznych, drzewo jest na ogół rysowane z korzeniem u góry, jak po prawej stronie rysunku. Do elementów drzewa często stosuje się nazewnictwo wzięte z dendrologii – korzeń, liście.



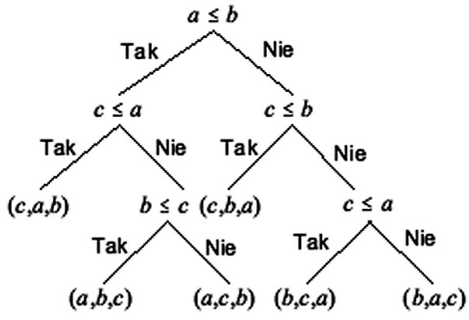

*Rysunek 9.*
Drzewa wyrażeń: (6 + 3)*(5 – 3*4) i $(x^2 + y^2)/(a - b)$

Wykonanie obliczeń zapisanych w postaci drzewa wyrażenia polega na „zwinięciu" tego drzewa, począwszy od liści, czyli przejściu od liści do korzenia, w którym jest otrzymywana wartość wyrażenia. Stosujemy przy tym oczywistą zasadę: aby móc wykonać dane działanie (umieszczone w wierzchołku pośrednim) musimy znać jego argumenty. A zatem, w wyrażeniu po lewej stronie na rys. 9. nie można wykonać odejmowania, zanim nie będzie znana wartość odjemnika w wierzchołku powyżej, czyli najpierw należy wykonać mnożenie umieszczone w tym wierzchołku. Proponujemy prześledzenie kolejności wykonania działań podczas obliczania wartości wyrażeń reprezentowanych przez drzewa na rys. 9.

Drzewa wyrażenia są przydatne przy interpretacji i tworzeniu postaci wyrażenia w tak zwanej **odwrotnej notacji polskiej** (**ONP**), w której zbędne są nawiasy. Nazwa tej notacji pochodzi od twórcy notacji polskiej, polskiego logika Jana Łukasiewicza (1878-1956). Ta notacja jest stosowana w wielu językach programowania, była wykorzystana również w kalkulatorach Hewlett-Packard i Sinclair. Postać ONP wyrażenia tworzy się stosując rekurencyjnie następującą zasadę: Zacznij w korzeniu i wypisz to, co jest w wierzchołku dopiero po wypisaniu tego co jest w lewym poddrzewie a później tego, co jest w prawym poddrzewie. Dla przykładu, zastosujmy tę zasadę do drzewa po lewej stronie na rys. 9. Przed wypisaniem znaku mnożenia musimy najpierw wypisać to, co jest w lewym poddrzewie i w prawym poddrzewie. Przechodzimy więc do lewego poddrzewa i stosujemy w nim tę samą zasadę: aby wypisać znak dodawania najpierw wypisujemy oba argumenty tego działania, czyli dla lewego poddrzewa wyrażenia ma postać 6 13 +. Podobnie, dla prawego poddrzewa otrzymujemy 5 3 4 * – . A zatem całe wyrażenia ma następującą postać ONP: 6 13 + 5 3 4 * – *.

## 3.2 DRZEWA ALGORYTMÓW

Drzewo algorytmu jest jednym ze sposobów zapisywania algorytmów, przypominającym schematy blokowe. Na rysunku 10 jest przedstawione drzewo algorytmu porządkowania trzech liczb *a*, *b*, *c*.




*Rysunek 10.*
Drzewo porządkowania trzech liczb

Wierzchołki końcowe drzewa algorytmu zawierają wszystkie możliwe rozwiązania – w naszym przykładzie są to wszystkie możliwe uporządkowania trzech elementów. Takich uporządkowań jest 6, a wykonywanie algorytmu zapisanego w postaci drzewa, rozpoczyna się w korzeniu tego drzewa. Dzieje się zatem nieco inaczej niż w drzewie wyrażenia. Zapisanie tego algorytmu w postaci drzewa jest przejrzystym opisem wykonywanych operacji i ich kolejności. Ponadto drzewo algorytmu może służyć do konstruowania i analizowania algorytmów.

Drzewo na rysunku 10 może być łatwo rozszerzone do drzewa algorytmu, który służy do porządkowania czterech liczb *a*, *b*, *c* i *d*. Wystarczy w tym celu wstawić czwarty element *d* do uporządkowanych ciągów trzech pierwszych elementów – w tym celu należy wykonać dwa dodatkowe porównania zaczynając od środkowego elementu.

Drzewo algorytmu może służyć do określenia liczby operacji wykonywanych przez algorytm. Zilustrujemy to na przykładzie drzewa przedstawionego na rysunku 10. Dla uzyskania konkretnego wyniku tego algorytmu, który znajduje się w wierzchołku wiszącym tego drzewa, liczba wykonanych porównań równa się liczbie wierzchołków pośrednich na drodze od korzenia (licząc również korzeń) do tego wierzchołka – tę liczbę nazywamy **długością** takiej drogi. Na przykład, jeśli dane trzy liczby *a*, *b* i *c* spełniają nierówności $c \le a \le b$ lub $c \le b \le a$, to algorytm wykonuje 2 porównania, a w pozostałych przypadkach – wykonuje 3 porównania. Naj-

większa długość drogi z korzenia do wierzchołka końcowego w drzewie nazywa się **wysokością drzewa**. Drzewo na rysunku 10 ma wysokość 3. Wysokość drzewa to największa liczba operacji wykonywanych w algorytmie dla jakiegokolwiek układu danych. W algorytmice wielkość ta nazywa się **złożonością obliczeniową algorytmu**. Jak ilustruje to nasz przykład, jest to **złożoność pesymistyczna** lub **złożoność najgorszego przypadku**, gdyż w niektórych przypadkach algorytm może działać szybciej. Dla danego problemu mamy tym lepszy (szybszy) algorytm, im mniejszą wysokość ma jego drzewo.

Drzewo algorytmu może być wykorzystane do wykazania, ile operacji musi wykonać jakikolwiek algorytm rozwiązywania danego problemu, co może być wykorzystane przy dowodzeniu optymalności algorytmów, patrz [5].

### 3.3 DRZEWA HUFFMANA – KRÓTKIE KOD

Pojemność pamięci komputerów rośnie w jeszcze większym tempie niż szybkość procesorów. Możliwość zapisania w pamięci komputera całej książki spowodowała chęć zapisania całych bibliotek. Możliwość pokazania na ekranie monitora dobrej jakości obrazu rozwinęła się do rozmiarów całego filmu. Alternatywą dla powiększenia pamięci jest **kompresja danych**, czyli minimalizowanie ich objętości przez reprezentowanie w zwięzłej postaci. Gwałtowny rozwój metod i form komunikowania się nie byłby możliwy bez ciągłego ulepszania metod kompresji danych. Odnosi się to zarówno do tradycyjnych form wymiany informacji, takich jak: faks, modem czy telefonia komórkowa, jak i do wymiany informacji za pośrednictwem sieci Internet, w tym zwłaszcza do wymiany informacji multimedialnych. Kompresja danych jest możliwa dzięki wykorzystaniu pewnych własności danych, na przykład często powtarzające się fragmenty można zastępować umownym symbolem lub im częściej jakiś fragment występuje tym mniejsza (krótsza) powinna być jego reprezentacja. W dalszej części tego punktu zajmiemy się jedynie kompresją tekstu, która polega na odpowiednim kodowaniu znaków.

Trudno uwierzyć, ale historia kompresji informacji ma swój początek ... w połowie XIX wieku. 24 maja 1844 roku Samuel Morse po raz pierwszy posłużył się zbudowanym przez siebie telegrafem i przesłał na odległość 37 mil (z Kapitolu w Waszyngtonie do Baltimore) depeszę, w której zacytował Biblię „To, co uczynił Bóg" (ang. *What Hath God Wrought*!). Jego osiągnięciem było nie tylko zbudowanie telegrafu, ale również opracowanie specjalnego alfabetu, zwanego **alfabetem Morse'a**, umożliwiającego kodowanie znaków w tekście za pomocą dwóch symboli – kropki i kreski, którym wtedy odpowiadały krótsze lub dłuższe impulsy elektryczne, a dzisiaj mogłyby to być cyfry 0 i 1. Morse przy konstrukcji swojego alfabetu przyjął założenie, że im częściej znak występuje w informacji, tym krótszy powinien mieć kod. Dlatego w jego alfabecie litera E ma kod • (kropka), a litera T ma kod – (kreska), gdyż są to najczęściej występujące litery w tekstach języka angielskiego. W języku polskim byłyby to odpowiednio litery A oraz I. Wadą alfabetu Morse'a jest konieczność stosowania znaku oddzielającego litery, gdyż kod danej litery może być początkową częścią kodu innej litery. Na przykład zapis •• może oznaczać dwie litery EE a także literę I, która ma taki kod.

Zauważmy, że stosując kod ASCII wszystkie znaki są kodowane za pomocą jednakowej liczby bitów.

Przedstawimy tera krótko kod Huffmana, który jest zbudowany na podobnych zasadach co alfabet Morse'a, ale nie ma wspomnianej wady tego alfabetu, czyli nie muszą być stosowane znaki do rozdzielania kodów liter tekstu. W k**odzie Huffmana**:

- znaki są zapisywane również za pomocą dwóch znaków (cyfr 0 i 1);
- kod żadnego znaku nie jest początkiem kodu żadnego innego znaku – nie trzeba więc stosować znaków rozdzielających;
- średnia długość kodu znaku w tekście jest możliwie najmniejsza.

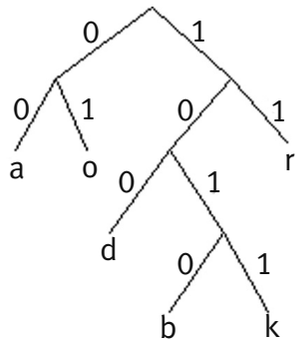
Kod Huffmana tworzy się za pomocą specjalnego drzewa, **drzewa Huffmana**, w którym powyższe własności są realizowane w następujący sposób:

- na gałęziach drzewa są umieszczone cyfry 0 i 1;
- znaki, dla których jest tworzony kod, znajdują się w wierzchołkach wiszących drzewa;
- wierzchołki wiszące ze znakami o większej częstości występowania w tekstach znajdują się wyżej niż wierzchołki ze znaki o mniejszych częstościach.

Drzewo Hoffmana dla danego zestawu znaków o podanych częstotliwościach jest budowane sukcesywnie przez łączenie ze sobą na każdym etapie dwóch poddrzew o najmniejszych częstotliwościach i zastępowaniu ich poddrzewem o sumie ich częstotliwości. Szczegółowy opis algorytmu Huffmana można znaleźć w podręczniku [2] i w książce [6]. Na rysunku 11 przedstawiamy drzewo Huffmana otrzymane tą metodą dla następujących znaków i ich częstotliwości (to są rzeczywiste częstotliwości występowania tych liter w tekstach w języku polskim):

a 8.71
b 1.29
d 3.45
k 3.10
o 7.90
r 4.63

W pierwszym kroku tworzenia tego drzewa zgodnie z naszkicowanym algorytmem, łączone są dwie najmniejsze częstotliwości odpowiadające literom b oraz k i zastępowane są przez częstotliwość 1.29 + 3.10 = 4.39. Następnie jest łączona częstotliwość 3.35 (litera d) z otrzymaną przed chwilą częstotliwością, itd.




*Rysunek 11.*
Drzewo Huffmana dla sześciu liter

Na podstawie drzewa z rysunku 11 otrzymujemy następujące kody liter:

a 00
b 1010
d 100
k 1011
o 01
r 11

Słowo ABRAKADABRA ma w otrzymanym kodzie postać 00101011001011001000010101100, złożoną z 29 cyfr, a stosując kod ASCII – postać tego słowa miałaby długość 88 znaków. Odpowiada to kompresji o wielkości 60%.

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

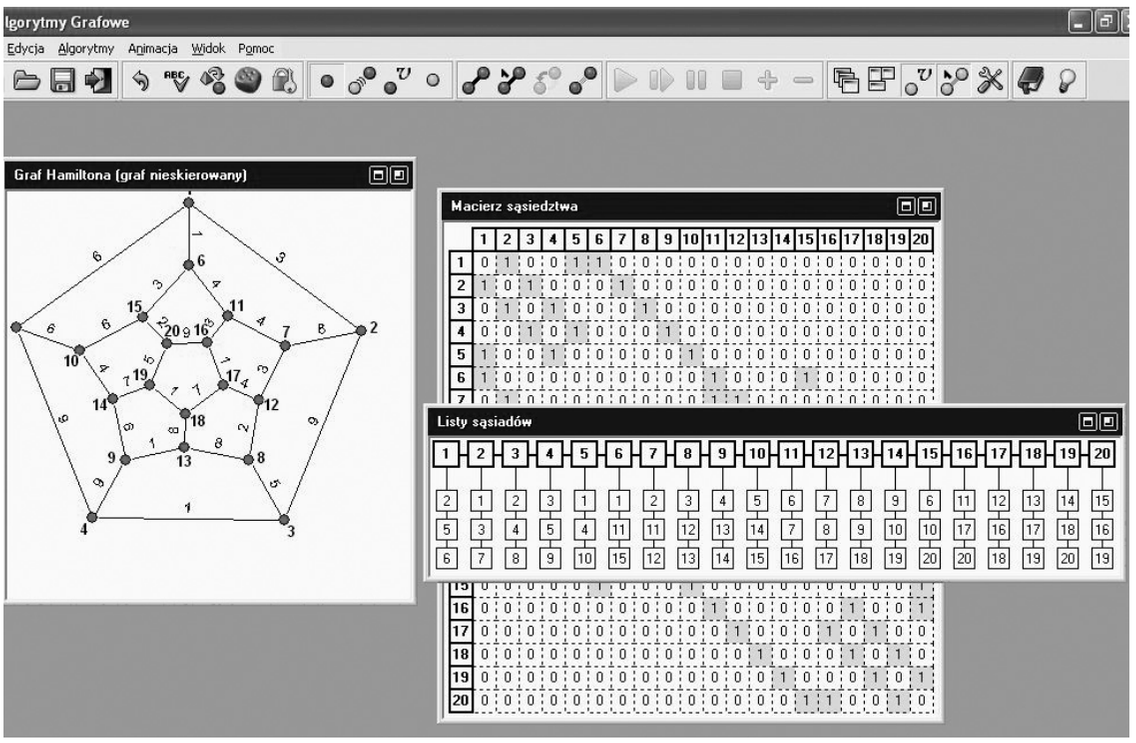
## 4 ALGORYTMY NA GRAFACH

W tej części wykładu przedstawiamy algorytmy na grafach, które mają wiele zastosowań praktycznych, są jednocześnie ilustracją typowych sposobów rozwiązywania problemów definiowanych na grafach. Szczegółowe rozważania dotyczące przedstawionych tutaj algorytmów można znaleźć w książce [7].

W czasie wykładu, działanie prezentowanych algorytmów będzie ilustrowane za pomocą specjalnego programu edukacyjnego **Algorytmy Grafowe**, w którym można:

- budować i modyfikować grafy,
- zapoznać się z podstawowymi sposobami reprezentowania grafów w komputerze,
- śledzić przebieg działania podstawowych algorytmów grafowych, w tym m.in. związanych z przeszukiwaniem grafów, znajdowaniem najkrótszych dróg w grafach i znajdowaniem najkrótszego drzewa rozpinającego grafu; podczas działania algorytmu, jego przebieg jest zsynchronizowany z demonstracją wykonywania programu w pseudo-języku programowania, pokazywane są również zmieniające się stany struktur danych, wykorzystywanych w algorytmach.

Na rysunku 12 jest przedstawione okno programy **Algorytmy Grafowe**, w którym widać graf i jego komputerowe reprezentacje.




*Rysunek 12.*
Okno w programie **Algorytmy Grafowe** z grafem dwunastościanu i jego komputerowymi reprezentacjami

W dalszej części tego rozdziału, najpierw definiujemy grafy i omawiamy ich komputerowe reprezentacje. Następnie przedstawiamy dwie grupy problemów definiowanych na grafach i demonstrujemy działanie wybranych algorytmów ich rozwiązywania.

### 4.4 GRAFY I ICH KOMPUTEROWE REPREZENTACJE

Jak już pisaliśmy, każdy graf składa się ze zbioru wierzchołków i zbioru połączeń. Wyróżniamy dwa rodzaje grafów: nieskierowane i skierowane.

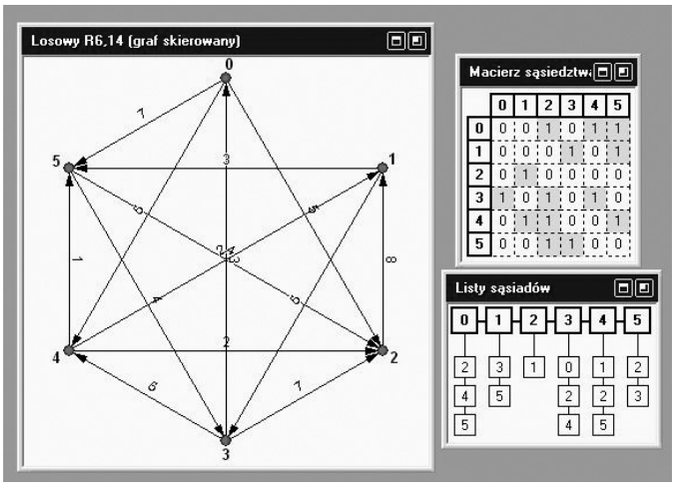
Grafy nieskierowane nazywamy po prostu **grafami**. Wszystkie połączenia w nich są nieskierowane, a to oznacza, że mogą być przechodzone w obie strony. Połączenia w grafach nieskierowanych nazywamy **krawędziami**. Przykładowy graf jest pokazany w oknie programu na rysunku 12.

Z kolei, grafy skierowane nazywamy **digrafami**. Wszystkie w nich połączenia mają zaznaczony kierunek przechodzenia. Połączenia w digrafach nazywamy **łukami**. Przykładowy digraf jest pokazany na rysunku 13.

Grafy i digrafy można reprezentować w komputerze na wiele sposobów. Na początku przyjmijmy, że wierzchołki są ponumerowane kolejnymi liczbami 1, 2, 3, ... W każdym ze sposobów reprezentowania digrafów

i grafów w komputerze jest zapisywane **sąsiedztwo wierzchołków**, czyli dla każdego wierzchołka jest podane, z którymi innymi wierzchołkami jest on połączony. W przypadku grafów symetrycznych, dla każdego wierzchołka podajemy numery wierzchołków, z którymi jest on połączony, a dla digrafów – podajemy numery wierzchołków, do których prowadzi skierowane połączenie.

Najpopularniejsze są dwie reprezentacje: macierzowa i w postaci list sąsiadów. W reprezentacji macierzowej, jeśli graf zawiera połączenie z wierzchołka *j* do wierzchołka *k*, to w **macierzy sąsiedztwa** na przecięciu wiersza *j* z kolumną *k* jest 1, a w przeciwnym razie jest 0. Z kolei w **listach sąsiedztwa**, w liście o numerze *j* występują wszystkie wierzchołki, które są sąsiednie do wierzchołka *k*. Na rysunkach 12 i 13 ilustrujemy te komputerowe reprezentacje dla grafów przedstawionych na tych rysunkach. Sąsiedztwo w grafach należy traktować jako relację symetryczną, natomiast w digrafach sąsiedztwo jest wskazywany przez zwrot łuków.




*Rysunek 13.*
Digraf i jego komputerowe reprezentacje

## 4.5 ZNAJDOWANIE NAJKRÓTSZYCH DRÓG W GRAFACH

**Drogą** w grafie nazywamy ciąg wierzchołków, dla których graf zawiera połączenia między kolejnymi wierzchołkami tego ciągu. Zauważmy na rysunkach 12 i 13, że połączenia mają przypisane im liczby – są to **wagi połączeń**, które mogą oznaczać na przykład rzeczywistą długość połączenia. **Długość drogi** definiujemy jako sumę długości połączeń, które tworzą tę drogę. Na przykład, droga (2, 3, 8, 13, 9, 4) w grafie na rysunku 12 ma długość 9+5+8+1+9 = 32, a droga (1, 3, 4, 5, 2) w digrafie na rysunku 13 ma długość 5+5+1+4 = 15.

Istnieje wiele różnych problemów, w których celem jest znalezienie najkrótszej drogi. Jeden z takich problemów był przedmiotem naszych rozważań w punkcie 2.3, gdzie zastanawialiśmy się, jak znaleźć trasę dla komiwojażera, czyli najkrótszy cykl przechodzący przez każdy wierzchołek w grafie dokładnie jeden raz. Klasyczne problemy najkrótszych dróg można podzielić na następujące grupy:

1. znaleźć najkrótszą drogę między wybraną parą wierzchołków w grafie;
2. znaleźć najkrótsze drogi z wybranego wierzchołka w grafie do wszystkich pozostałych wierzchołków w grafie;
3. znaleźć najkrótsze drogi między każdą parą wierzchołków w grafie.

W tych sformułowaniach problemów, „znaleźć najkrótszą drogę" oznacza znaleźć długość najkrótszej drogi oraz ciąg wierzchołków drogi o najkrótszej długości.

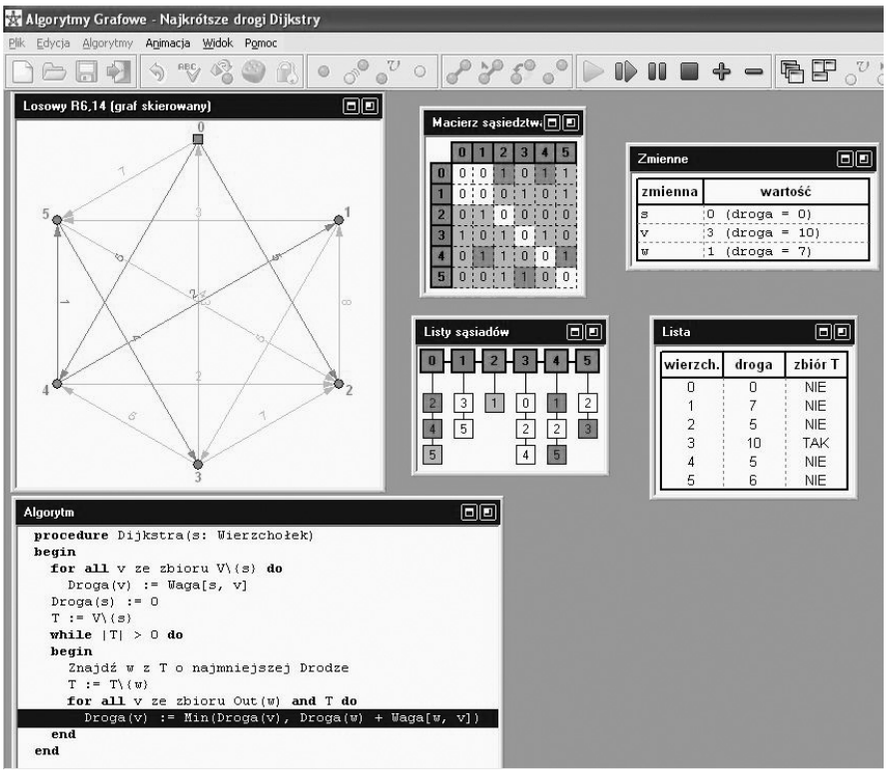
Łatwo zauważyć, że rozwiązanie problemu typu 1 może być wykorzystane w rozwiązaniu problemów typu 2 i 3. Podobnie, rozwiązanie problemu typu 2 może być wykorzystane do rozwiązania problemu typu 3. Na ogół, rozwiązywanie problemu typu 2 może być przerwane, gdy mamy już rozwiązanie problemu typu 1. Dlatego zatrzymamy się jedynie nad problemem typu 2 i dodatkowo założymy, że wszystkie wagi połączeń są nieujemne (w ogólności nie muszą być). A zatem, rozważymy następujący problem, przyjmując dodatkowo, że szukamy najkrótszych dróg w obciążonych digrafach. Jeśli chcemy znaleźć najkrótsze drogi w obciążonym grafie symetrycznym, to każdą krawędź traktujemy jako parę łuków przeciwnie skierowanych.

**Problem najkrótszych dróg z ustalonego wierzchołka w digrafie obciążonym**
*Dane:* digraf *G* z łukami obciążonymi nieujemnymi wagami; wybrany wierzchołek *s*.
*Wynik:* najkrótsze drogi w *G* z *s* do wszystkich pozostałych wierzchołków w digrafie *G*.

Do rozwiązania tego problemu posłużymy się algorytmem Dijkstry. Nie zamieszczamy tutaj jednak szczegółowego opisu tego algorytmu, tylko opiszemy jego główną ideę, a następnie w czasie wykładu zilustrujemy działanie tego algorytmu posługując się programem **Algorytmy Grafowe**, patrz rysunek 14.

**Algorytm Dijkstry** ma charakter metody zachłannej. Startujemy z danego wierzchołka *s*, zaliczamy go do rozwiązania i w pierwszym kroku obliczamy długości dróg z *s* do wszystkich jego wierzchołów sąsiednich (te drogi składają się z pojedynczych łuków) a następnie wybieramy wierzchołek sąsiedni *w*, który jest najbliżej *s* – wierzchołek *w* zaliczamy do rozwiązania i dla każdego wierzchołka *v*, który nie należy jeszcze do rozwiązania, sprawdzamy, czy droga z *s* do *v* przez *w* nie jest krótsza od dotychczasowej drogi najkrótszej z *s* do *v*. Jeśli tak jest, to poprawiamy długości dróg. Następnie, ponownie wybieramy wierzchołek spośród tych, które nie należą jeszcze do rozwiązania, a który jest najbliższy *s* i powtarzamy to postępowanie. Liczba iteracji w tej metodzie wynosi $n - 1$, bo tyle wierzchołków trzeba dołączyć do rozwiązania w kolejnych krokach.

Polecamy prześledzić działania algorytmu Dijkstry w programie **Algorytmy Grafowe** na wybranych przykładach digrafów i grafów.




*Rysunek 14*.
Demonstracja działania algorytmu Dijkstry w programie **Algorytm Grafowe**, zastosowana do digrafu z rysunku 13

## 4.6 ZNAJDOWANIE NAJKRÓTSZYCH DRZEW ROZPINAJĄCYCH W GRAFACH

Jak pamiętamy, drzewo jest grafem, który jest spójny i nie zawiera cykli, a graf jest spójny, jeśli każda para jego wierzchołków jest połączona drogą. Zakładamy tutaj że graf jest symetryczny. Jeśli graf jest spójny, to można znaleźć w nim podgraf, który jest drzewem – takie drzewo w grafie nazywamy **drzewem rozpinającym**. Drzewo rozpinające w grafie spójnym można znajdować na dwa sposoby. Drzewo o *n* wierzchołkach ma dokładnie $n - 1$ krawędzi:

1. usuwać krawędzie, które nie powodują rozpojenia grafu, tak długo jak długo graf ma więcej niż $n - 1$ krawędzi, gdzie *n* jest liczbą wierzchołków w grafie;
2. wybierać krawędzie w grafie, ale tylko takie, które nie powodują powstania cyklu wśród wybranych krawędzi; postępować tak długo, aż wybranych zostanie $n - 1$ krawędzi, gdzie *n* jest liczba wierzchołków w grafie.

Rozważany przez nas problem ma następującą postać:

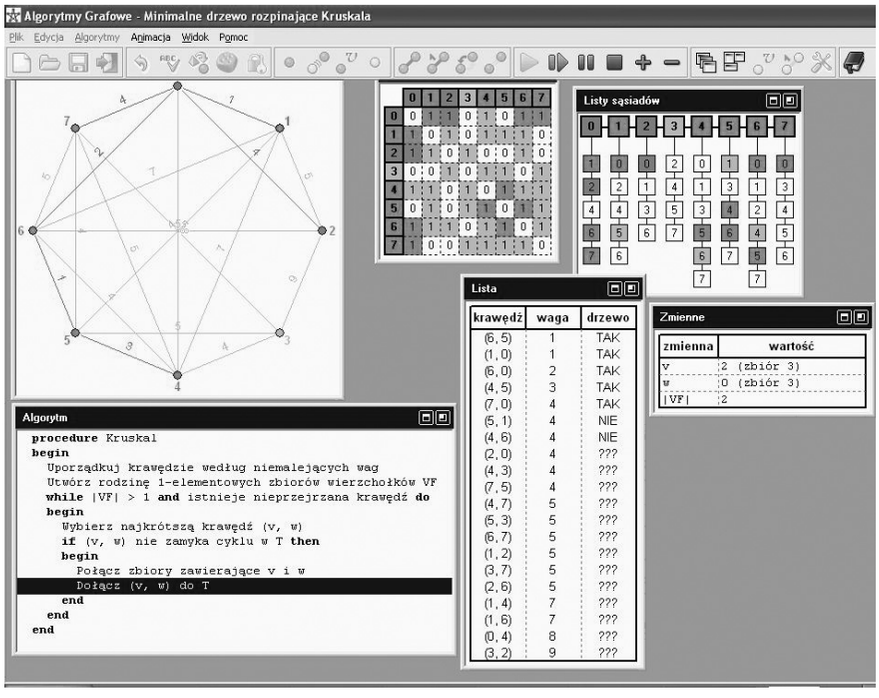
**Problem najkrótszego drzew rozpinającego w grafie obciążonym**
*Dane:* graf *G* z krawędziami obciążonymi wagami.
*Wynik:* najkrótsze drzewo rozpinające w grafie *G*. Za długość drzewa przyjmujemy sumę wag (długości) jego krawędzi.

Najbardziej znanym algorytmem rozwiązywania tego problemu jest **zachłanny algorytm Kruskala**. Polega on tworzeniu drzewa rozpinającego metodą nr 2 przy założeniu, że krawędzie są rozpatrywane w kolejności od najlżejszych (najkrótszych) Opiszemy ten algorytm w pseudo-języku programowania. Zakładamy, ze rozważany graf jest spójny.

```
algorytm Kruskala;
begin
   T:=pusty;  {T – zbiór krawędzi tworzonego drzewa}
   E – zbiór krawędzi grafu G;
   while T ma mniej elementów niż n – 1 do begin
      wybierz najkrótszą krawędź e w zbiorze E;
      usuń e z E;
      if e nie tworzy cyklu z krawędziami w T then
         dołącz krawędź e do zbioru T
   end
end
```

Na rysunku 15 jest przedstawione okno z demonstracją algorytmu Kruskala na losowym grafie.

Ciekawą modyfikacją algorytmu Kruskala jest algorytm Prima-Dijkstry, który jest również algorytmem zachłannym, ale podobnie jak algorytm Dijkstry dla znajdowania najkrótszych dróg, może rozpocząć budowanie najkrótszego drzewa rozpinającego zaczynając w dowolnym wierzchołku grafu, jako korzeniu. Proponujemy prześledzić działanie tego algorytmu w programie **Algorytmy Grafowe**.




*Rysunek 15.*
Demonstracja działania algorytmu Kruskala w programie **Algorytm Grafowe**

## LITERATURA

1. Cormen T.H., Leiserson C.E., Rivest R.L., *Wprowadzenie do algorytmów*, WNT, Warszawa 1997
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M., *Informatyka, Część 1 i 2, Podręcznik dla LO*, WSiP, Warszawa 2002-2003
3. Harel D., *Algorytmika. Rzecz o istocie informatyki*, WNT, Warszawa 1992
4. Knuth D.E., *Sztuka programowania*, Tomy 1 – 3, WNT, Warszawa 2003
5. Sysło M.M., *Algorytmy*, WSiP, Warszawa 1997
6. Sysło M.M., *Piramidy, szyszki i inne konstrukcje algorytmiczne*, WSiP, Warszawa 1998. Kolejne rozdziały tej książki są zamieszczone na stronie: http://www.wsipnet.pl/kluby/informatyka_ekstra.php?k=69
7. Sysło M.M., Deo N., Kowalik J.S., *Algorytmy optymalizacji dyskretnej z programami w języku Pascal*, WN PWN, Warszawa 1997
8. Wilson R.J., *Wprowadzenie do teorii grafów*, WN PWN, Warszawa 1998

W projekcie **Informatyka +**, poza wykładami i warsztatami, przewidziano następujące działania:

- 24-godzinne kursy dla uczniów w ramach modułów tematycznych
- 24-godzinne kursy metodyczne dla nauczycieli, przygotowujące do pracy z uczniem zdolnym
- nagrania 60 wykładów informatycznych, prowadzonych przez wybitnych specjalistów i nauczycieli akademickich
- konkursy dla uczniów, trzy w ciągu roku
- udział uczniów w pracach kół naukowych
- udział uczniów w konferencjach naukowych
- obozy wypoczynkowo-naukowe.

Szczegółowe informacje znajdują się na stronie projektu

**www.informatykaplus.edu.pl**