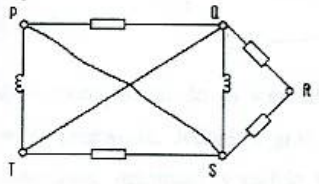
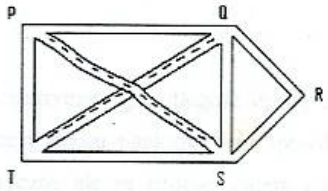
1. Definicje i pojęcia. Typy grafów.

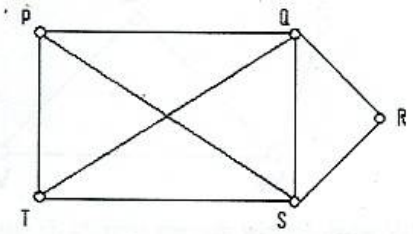
1.1 Intuicyjne pojęcie grafu.

a) 

b) 

Rozpatrzmy rysunek a) i b), które przedstawiają fragment obwodu elektrycznego i mapy drogowej. Każdy z nich można przedstawić symbolicznie za pomocą punktów P, Q, R, S, T, tzw. wierzchołków i linii – krawędzi. Cały taki diagram nazywamy grafem. Zauważmy, że przecięcie linii PS i QT nie jest wierzchołkiem grafu, ponieważ nie odpowiada połączeniu dwóch przewodów, ani skrzyżowaniu dróg. Stopień wierzchołka jest liczbą krawędzi, które zawierają ten wierzchołek jako jeden ze swych końców – np. stopień wierzchołka Q wynosi 4.

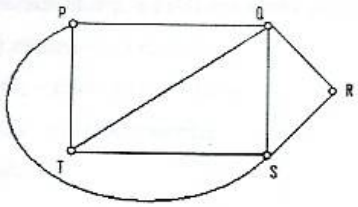
Sytuacje z rysunków a) i b) można ogólnie przedstawić jako graf:

c) 

Oczywiście graf c) może przedstawiać również inne sytuacje: np. P, Q, R, S, T reprezentują drużyny piłkarskie, a istnienie krawędzi może oznaczać rozegranie spotkania między odpowiednimi zespołami np. drużyna P grała z S, lecz nie grała z R.

Inaczej można przedstawić te sytuacje w postaci grafu.

d)

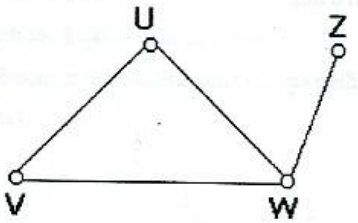


W grafie d) w porównaniu do c) straciliśmy własności metryczne tzn. długość drogi, czy prostoliniowość przewodu. Jednakże graf jest reprezentacją zbioru punktów oraz sposobu, w jaki są połączone, natomiast wszelkie własności metryczne nie są istotne. Zatem grafy c) i d) są uważane za ten sam graf.

1.2 Definicje

1° **Grafem prostym** G nazywamy parę (V(G), E(G)), gdzie V(G) jest niepustym skończonym zbiorem wierzchołków (węzłów, punktów), a E(G) jest skończonym zbiorem nieuporządkowanych par różnych (w parach) elementów zbioru V(G) zwanych krawędziami (liniami).
Mówimy, że V(G) jest zbiorem wierzchołków, a E(G) zbiorem krawędzi grafu G.
Np. poniższy rysunek

e)



przedstawia graf prosty G, którego zbiorem wierzchołków V(G) jest {U, V, W, Z} natomiast zbiór krawędzi E(G) składa się z par {U, V}, {V, W}, {U, W} i {W, Z}. Krawędź {V, W} można symbolicznie zapisać: VW. Ponieważ E(G) jest zbiorem (a nie rodziną), zatem w grafie G nie może łączyć dwóch wierzchołków więcej niż jedna krawędź.
Zdefiniujmy teraz pojęcie grafu ogólnego lub po prostu grafu, w którym dwa wierzchołki mogą być połączone więcej niż jedną krawędzią oraz mogą istnieć krawędzie łączące dany wierzchołek ze sobą.

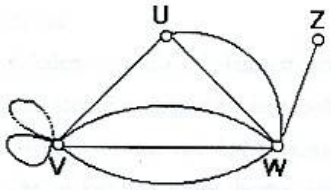
2° Grafem G nazywamy parę (V(G), E(G)), gdzie V(G) jest niepustym skończonym zbiorem elementów, zwanych wierzchołkami, a E(G) jest skończoną rodziną nieuporządkowanych par elementów zbioru V(G) zwanych krawędziami.

V(G) – zbiór wierzchołków

E(G) – rodzina krawędzi

Zatem w poniższym grafie

f)



V(G) = {U, V, W, Z}

E(G) jest rodziną krawędzi: {U, V}, {V, V}, {V, V}, {V, W}, {V, W}, {V, W}, {U, W}, {U, W}, {W, Z}

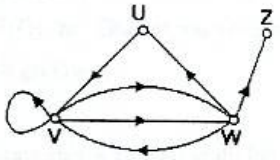
3° Digrafem D nazywamy parę (V(D), A(D)), gdzie V(D) jest skończonym niepustym zbiorem elementów zwanych wierzchołkami, zaś A(D) jest skończoną rodziną uporządkowanych par elementów zbioru V(D) zwanych łukami (lub krawędziami skierowanymi).

Łuk, którego pierwszym elementem jest a, a drugim b nazywa się łukiem z a do b, co zapisujemy (a, b) lub ab. Łuki ab i ba są różne.

Jeżeli A(D) jest zbiorem, a nie rodziną oraz nie ma pętli to D nosi nazwę digrafu prostego (analogia do grafów)

Rysunek:

g)



przedstawia digraf o krawędziach UV, VV, VW, VW, WV, WU, WZ

Uwaga: Można się spotkać z innymi definicjami grafów, np. graf jest utożsamiany z grafem prostym lub digrafem.

4° Dwa **wierzchołki** V i W grafu G są **sąsiednie** jeśli istnieje łącząca je krawędź (tzn. krawędź VW). Mówimy wtedy, że wierzchołki V i W są **incydentne** z tą krawędzią

5° Dwie **krawędzie** grafu G są **sąsiednie** jeśli mają przynajmniej jeden wspólny wierzchołek.

6° **Stopień wierzchołka** V jest liczbą krawędzi incydentnych z V. Stopień wierzchołka V oznaczamy przez p(V).

Licząc stopień wierzchołka każdą pętlę liczymy 2 razy.

7° **Wierzchołkiem izolowanym** nazywamy wierzchołek stopnia zero.

8° **Wierzchołkiem końcowym** (lub wiszącym) nazywamy wierzchołek stopnia 1.

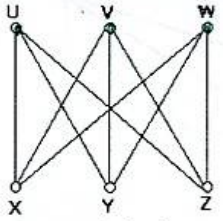
Zatem graf na rysunku f) ma:

1 wierzchołek końcowy, jeden - stopnia 3, jeden – stopnia 6 i jeden stopnia 8 (wskaż!).

Łatwo zauważyć, że suma stopni wszystkich wierzchołków grafu jest liczbą parzystą – równą podwojonej liczbie krawędzi. (ponieważ każda krawędź uwzględniona jest dwukrotnie). Wynika stąd również, że w każdym grafie liczba wierzchołków stopnia nieparzystego jest parzysta (uzasadnij).

9° Dwa **grafy** G_1 i G_2 są **izomorficzne** jeśli istnieje wzajemnie jednoznaczna odpowiedniość pomiędzy wierzchołkami grafu G_1 i wierzchołkami grafu G_2 taka, że liczba krawędzi łączących dowolne dwa wierzchołki w G_1 jest równa liczbie krawędzi łączących odpowiadające im wierzchołki w G_2.

h)



Grafy narysowane powyżej są izomorficzne, a odpowiednie przyporządkowanie ma postać:

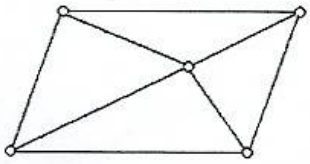
U$\leftrightarrow$ L, V $\leftrightarrow$ M, W$\leftrightarrow$ N, X$\leftrightarrow$ P, Y$\leftrightarrow$ Q, Z$\leftrightarrow$ R

10° Graf G_1 jest **podgrafem** grafu G, gdy jego wszystkie wierzchołki należą do V(G), a krawędzie należą do E(G), np. Graf e) jest podgrafem f), ale nie jest podgrafem żadnego z ostatnio przedstawionych grafów.

Przykłady:

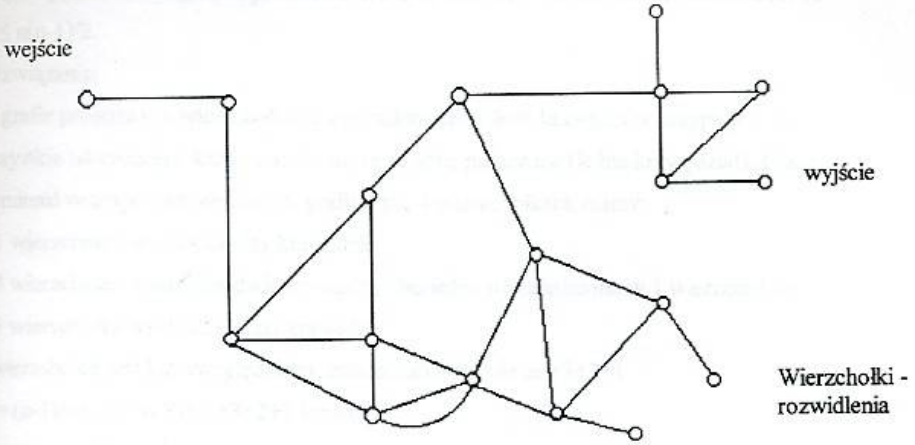
1. Przykłady przedstawienia w postaci grafu lub digrafu następujących zagadnień:

a) wierzchołki i krawędzie wielościanu – ostrosłup czworokątny



ostrosłup

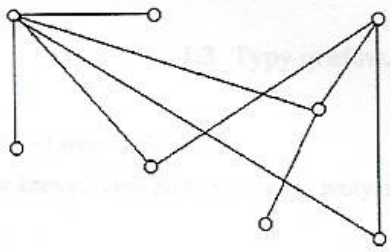
b) Plan labiryntu (wierzchołki – rozwidlenia)



c) Znajomość pomiędzy uczestnikami przyjęcia

wierzchołki – osoby

krawędzie – relacja znajomości między osobami



d) Drzewo genealogiczne pewnej rodziny

a $\rightarrow$ b oznacza, że b jest dzieckiem a (do samodzielnego ćwiczenia)

e) Drzewo dzielników liczby 48

a $\rightarrow$ b oznacza, że a jest podzielne przez b (do samodzielnego ćwiczenia)

Zadanie.

Przyjmując, że G jest grafem prostym o m krawędziach i n wierzchołkach udowodnić, że $m \leq n(n-1)/2$.

Rozwiązanie:

W grafie prostym o n wierzchołkach jest maksymalna ilość krawędzi w przypadku, gdy wszystkie wierzchołki, każdy z każdym, są ze sobą połączone (jedną krawędzią!). Chcąc wymienić wszystkie krawędzie dla grafu o np. 4 wierzchołkach mamy:

Z 1 wierzchołka wychodzą trzy krawędzie

Z 2 wierzchołka wychodzą dwie krawędzie (bo jedna uwzględniona w 1 wierzchołku)

Z 3 wierzchołka wychodzi jedna krawędź

4 wierzchołek jest już uwzględniony, zatem maksymalnie m =3+2+1

m = (n-1)+(n-2)+(n-3)+...+3+2+1 krawędzi

Ze wzoru na sumę ciągu arytmetycznego mamy

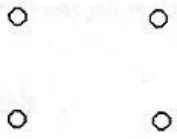
m = (1+n-1)x(n-1)/2 = n(n-1)/2

Zatem w dowolnym grafie prostym $m \leq n(n-1)/2$ a równość zachodzi, gdy każdy wierzchołek jest połączony z każdym

1.3 Typy grafów.

1. **Grafy puste** (N_i – i wierzchołków)

Graf, którego zbiór krawędzi jest pusty nazywamy pustym (lub bezkrawędziowym)



Każdy wierzchołek jest izolowany

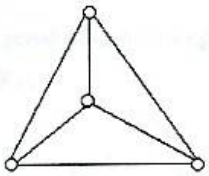
2. Grafy pełne

Są to takie grafy proste, że każde dwa wierzchołki grafu są sąsiednie. Graf pełny o n wierzchołkach oznaczamy K_n

Np.

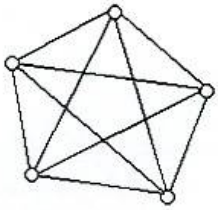
K_4

i)



K_5

j)

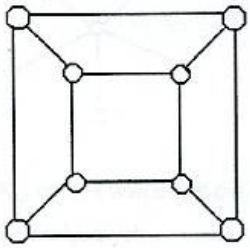


Z poprzedniego ćwiczenia wiemy, że liczba krawędzi w grafie pełnym jest równa n(n-1)/2

3. Grafy regularne

Graf G jest regularny, gdy każdy jego wierzchołek ma ten sam stopień. Szczególnie ważne są grafy regularne stopnia 3, które nazywamy kubicznymi np. grafy k) i i) lub

k)



Zauważmy, że graf pusty jest regularny stopnia 0 a graf pełny jest regularny stopnia n-1.

4. Grafy platońskie

Są to grafy utworzone przez wierzchołki i krawędzie 5 wielościanów foremnych: czworościan, ośmiościan, sześcian, dwunastościan, dwudziestościan np. i) i j)

5. Grafy dwudzielne

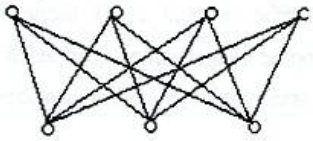
Graf jest dwudzielny jeżeli zbiór jego wierzchołków można podzielić na dwa rozłączne zbiory V_1 i V_2 w taki sposób, że każda krawędź grafu G łączy dowolny wierzchołek zbioru V_1 z dowolnym wierzchołkiem zbioru V_2 – oznaczamy $G(V_1, V_2)$; (można pokolorować wierzchołki na niebiesko lub czerwono tak, aby każda krawędź miała jeden koniec czerwony, a drugi niebieski).

Jeśli graf $G(V_1, V_2)$ jest prosty oraz każdy wierzchołek zbioru V_1 jest połączony z każdym wierzchołkiem zbioru V_2, to nazywamy go pełnym grafem dwudzielnym i oznaczamy go $K_{r,s}$; gdzie r i s są liczbami wierzchołków w V_1 i V_2

Przykłady

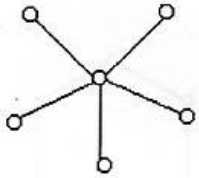
- na rys. h) przedstawiono dwa grafy $K_{3,3}$
- przykład $K_{4,3}$

l)



- pełny graf dwudzielny postaci $K_{1,n}$ nazywamy gwiazdą np.

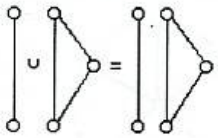
m)



Zauważmy, że graf $K_{r,s}$ ma r+s wierzchołków i r x s krawędzi.

Przykład sumy grafów rozłącznych

n)



(sumujemy zbiory wierzchołków i zbiory krawędzi)

6. Grafy spójne

Graf jest spójny, jeśli nie może być przedstawiony w postaci sumy dwóch grafów. W przeciwnym razie graf nazywamy niespójnym. (np. n) jest grafem niespójnym). Wszystkie pozostałe były spójne.

Jest oczywiste, że każdy graf niespójny może być przedstawiony jako suma skończonej liczby grafów spójnych (składowych spójnych).

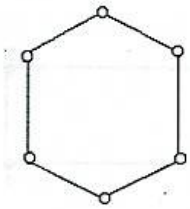
7. Zespolenie grafów G_1 i G_2 polega na tym, że sumujemy zbiory wierzchołków i łączymy każdy wierzchołek grafu G_1 z każdym wierzchołkiem grafu G_2 oznaczamy $G_1 + G_2$. Zauważmy, że $K_{r,s}$ może być rozumiany jako zespolenie K_r i K_s.

Działania sumy i zespalania grafów są łączne i przemienne.

8. Grafy cykliczne i koła

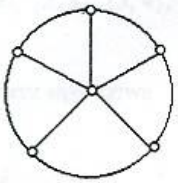
Graf spójny, regularny stopnia 2 nazywamy grafem cyklicznym i oznaczamy C_n, gdzie n jest liczbą wierzchołków. Np.

C_6



Zespolenie grafów N_1 i C_{n-1} ($n \geq 3$) nazywamy kołem o n wierzchołkach i oznaczamy przez W

W_6



2. Reprezentacje komputerowe grafu. Przeszukiwanie grafów.

Niech w grafie G = (V, E), n = |V| oraz m = |E|. Oznaczmy przez *N(v)* zbiór wierzchołków sąsiednich z $v \in V$.

2.1 Reprezentacje grafów.

1. Macierz sąsiedztwa: $A = (a_{ij})_{n \times n}$, gdzie

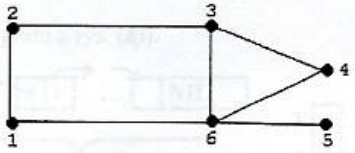
$$a_{ij} = \begin{cases} 1 & : \text{ gdy wierzchołek } i \text{ jest połączony z } j \\ 0 & : \text{ gdy brak połączenia } i \text{ z } j. \end{cases}$$

Jeżeli para wierzchołków *i, j* jest w digrafie połączona łukiem, wówczas w macierzy sąsiedztwa $a_{ij} = 1$ oraz $a_{ij} = -1$, a pozostałe elementy są zerami.

Przykład

a) Graf i jego macierz sąsiedztwa

(1)

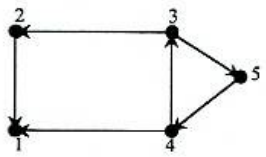


0	1	0	0	0	1
1	0	1	0	0	0
0	1	0	1	0	1
0	0	1	0	0	1
0	0	0	0	0	1
1	0	1	1	1	0

V={1, 2, 3, 4, 5, 6}
E={{1, 2}, {1, 6}, {2, 3}, {3, 4}, {3, 6}, {4, 6}, {5, 6}}

b) Digraf i jego macierz sąsiedztwa

(2)

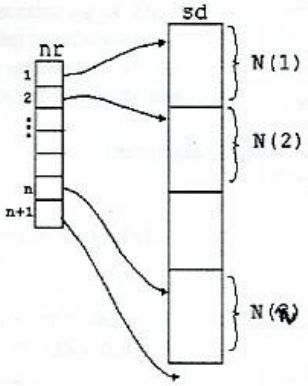


0	-1	0	-1	0
1	0	-1	0	0
0	1	0	-1	1
1	0	1	0	-1
0	0	-1	1	0

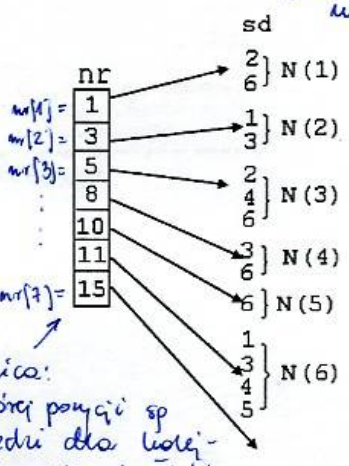
V={1, 2, 3, 4, 5}
E={{2, 1}, {4, 1}, {3, 2}, {4, 3}, {3, 5}, {5, 4}}

2. Zbiory sąsiadów w postaci tablicowej.

Przykład (dla grafu z rys. (1))



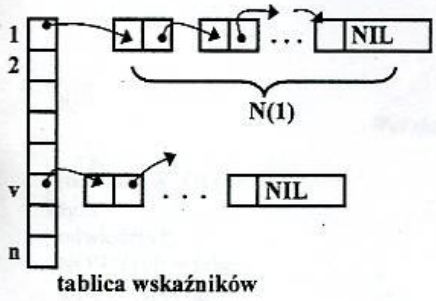
tablica sąsiadów dla wierzchołka wierzchołka



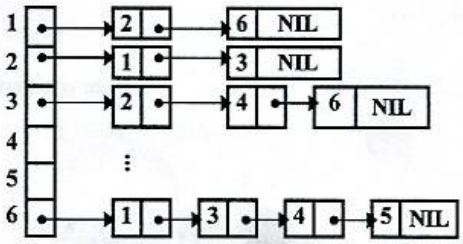
tablica:
od drugiej pozycji są sąsiedzi dla kolejnych wierzchołków

3. Zbiory sąsiadów w postaci listowej (także uporządkowane)

Przykład (dla grafu z rys. (1)).



tablica wskaźników



Przeszukiwanie w głąb

Przeszukiwanie w głąb, przeszukiwanie metodą DSF, Deep-First Search, metoda przeszukiwania grafu polegająca na sięganiu do wierzchołków możliwie jak najodleglejszych od źródła przeszukiwania. Podczas przeszukiwania w głąb tworzy się las rozłącznych drzew przeszukiwań wgłąb, a odwiedzanym wierzchołkom przypisuje etykiety czasowe, numerujące kroki obliczeń. Przeszukiwanie w głąb pozwala zebrać dużo informacji o strukturze grafu.

Przeszukiwanie wszerz

Przeszukiwanie wszerz, przeszukiwanie metodą BFS, Breadth-First Search, metoda przeszukiwania grafu, w której następuje systematyczne odwiedzenie każdego wierzchołka osiągalnego z wyróżnionego wierzchołka początkowego, zwanego źródłem, i obliczanie odległości (najmniejszej liczby krawędzi) od źródła do osiągalnych wierzchołków. W wyniku przeszukiwania wszerz otrzymuje się też zakorzenione u źródła drzewo przeszukiwania wszerz zawierające wszystkie osiągalne wierzchołki.

2.2. Przeszukiwanie grafu w głąb.

Przeszukiwanie grafu polega na tym, że należy wystartować z dowolnego wierzchołka i przemieszczając się po krawędziach grafu, odwiedzić wszystkie wierzchołki.

Poniżej przedstawiony jest algorytm przeszukiwania grafu w głąb. Startujemy w nim z pewnego ustalonego wierzchołka v, a graf reprezentowany jest poprzez listy sąsiadów.

Algorytm napisany został w pseudojęzyku:

Przeglądanie grafu w głąb – dfs (ang. depth first search)
Wersja nierekurencyjna.

```
procedure W_GL(v);
begin
STOS: = v;
NOWY[v]: = false;
While STOS ≠ 0 do
   begin
      t:=STOS;                {pobranie i usunięcie elementu ze szczytu stosu}
      odwiedź(t);
      for u ∈ N(t) do
         if NOWY[u] then
            begin
               NOWY[u]:=false;
               STOS:=u;
            end;
   end;
```

Wersja rekurencyjna.

```
procedure W_GL(v);
begin
 odwiedź(v);
 NOWY[v]: = false;
 for u ∈ N(v) do
        if NOWY[u] then W_GL(u);
end;
```

Przed użyciem procedur przeszukiwania grafu należy nadać wartość *true* wszystkim elementom tablicy NOWY : for v ∈ V do NOWY[v]:=true;

2.3 Przeszukiwanie grafu wszerz (drzewo bfs).

```
procedure WSZERZ(v);
begin
KOLEJKA: = v;
NOWY[v]: = false;
While KOLEJKA ≠ 0 do
   begin
      t:= KOLEJKA;                    {pobranie i usunięcie elementu z kolejki}
      odwiedź(t);
      for u ∈ N(t) do
         if NOWY[u] then
            begin
               NOWY[u]:=false;
               KOLEJKA:=u;
            end;
   end;
```

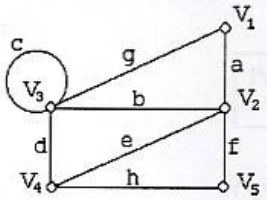
```
D[1..N] tabl. odległ. (-1 dla wierzch. białych)
P[1..N] tabl. poprzednikow (wierzch. i-ty wzięty odległość od P[i])
procedure BFS;
  begin
    Initialize_Queue;
    for i:=1 to N do D[i]:=-1; {każdy wierzch. biały}
  D[b]:=0;
  P[b]:=-1                    } b -> wierzch. początkowy,
  Put_On_Queue(b);
  While Queue_not_empty do
      begin
        Get_from_queue(v); {jak usuwamy z kol. robimy czarny}
        for i:= successor(v) do
                if D[i]=-1 then
                begin
                  D[i]:=D[v]+1;
                  P[i]:=v;
                Put_on_Queue(i); {jak do kolejki to
                end;                robimy szary}
      end;
  end;
```

3. Drogi, ścieżki, obwody.

3.1 Pojęcia wstępne.

Droga jest to skończony ciąg występujących na przemian wierzchołków i krawędzi, rozpoczynający się i kończący wierzchołkami, taki, że każda krawędź jest incydentna do wierzchołków poprzedzających i następujących po niej. Żadna krawędź nie jest przebywana w drodze więcej niż jeden raz.

Na rysunku



Przykładem drogi jest: V_1 A V_2 B V_3 C V_3 D V_4 E V_2 F V_5

Drogę można też nazwać ciągiem krawędzi lub łańcuchem. W danym grafie G zbiór wierzchołków i krawędzi tworzących jakąś drogę jest podgrafem grafu G. Jeżeli droga rozpoczyna się lub kończy w tym samym wierzchołku (wierzchołki końcowe się pokrywają), to taką drogę nazywamy **drogą zamkniętą**.

Drogę, która nie jest zamknięta nazywamy **drogą otwartą**.

Ścieżka jest to droga otwarta, w której żaden wierzchołek nie pojawia się więcej niż jeden raz.

W przykładzie wypisana droga nie jest ścieżką. Oto przykład ścieżki V_1 A V_2 B V_3 D V_4

Długością ścieżki nazywamy liczbę krawędzi w ścieżce.

Łatwo zauważyć, że krawędź, która nie jest pętlą własną jest ścieżką o długości 1. Jasne jest również, że pętlę własną można włączyć do drogi, ale nie do ścieżki. Wierzchołki końcowe ścieżki są stopnia jeden, a reszta wierzchołków (tzw. pośrednich) jest stopnia 2.

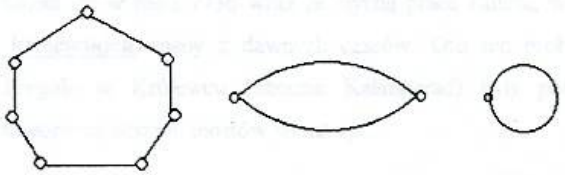
– te stopnie liczymy w stosunku do ścieżki, a nie do grafu wyjściowego, w którym ścieżka może się zawierać

Obwodem nazywamy drogę zamkniętą, w której żaden z wierzchołków (z wyjątkiem początkowego i końcowego) nie pojawia się więcej niż jeden raz.

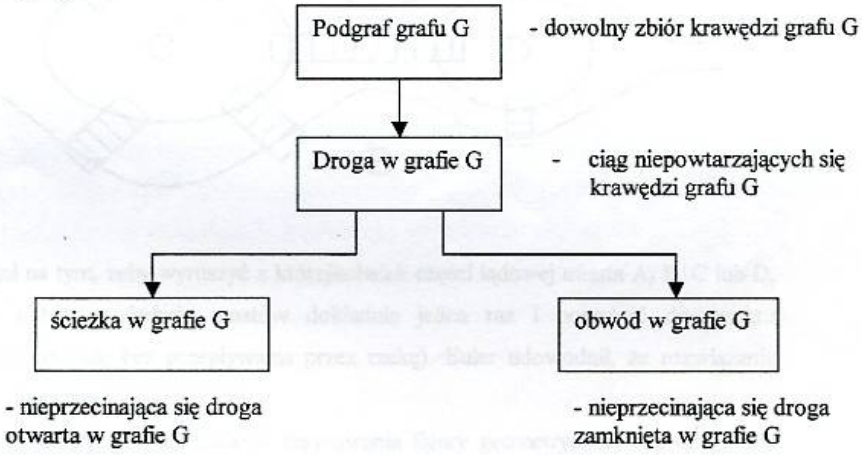
Innymi słowy: obwód jest zamkniętą, nie przecinającą się drogą,

np. na rys. V_2 B V_3 D V_4 E V_2

Przykłady obwodów:



Obwód nazywać można cyklem, drogą okrężną, wielokątem. Omówione wyżej pojęcia można przedstawić następująco:

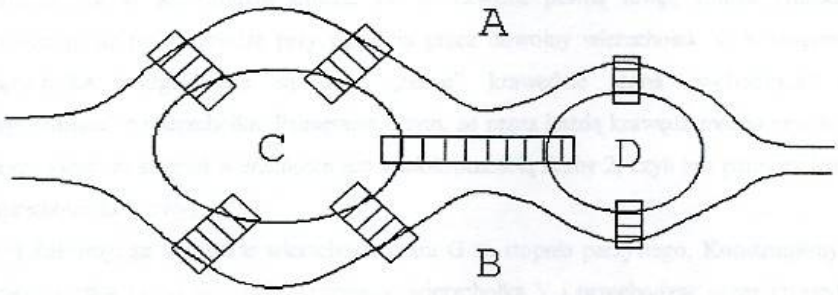


Inna definicja spójności:

Graf G jest spójny, jeżeli istnieje co najmniej jedna ścieżka między każdą parą wierzchołków w G

3.2 Grafy Eulera

Teoria grafów narodziła się w roku 1736 wraz ze słynną pracą Eulera, w której rozwiązał **problem mostów królewskich** znany z dawnych czasów. Oto ten problem:Dwie wyspy C i D, na rzece Pregole w Królewcu (obecnie Kaliningrad) były połączone ze sobą i z brzegami A i B za pomocą siedmiu mostów jak na rys.



Problem polegał na tym, żeby wyruszyć z którejkolwiek części lądowej miasta A, B, C lub D, przejść przez każdy z siedmiu mostów dokładnie jeden raz i powrócić do punktu wyjściowego (oczywiście bez przepływania przez rzekę). Euler udowodnił, że rozwiązanie tego problemu nie istnieje.

Problem ten jest identyczny, jak zadanie narysowania figury geometrycznej bez odrywania ołówka od kartki i bez wielokrotnego rysowania tych samych boków.

Euler rozwiązał zadanie ogólniejsze: w jakim typie grafu można znaleźć drogę zamkniętą przechodzącą dokładnie jeden raz przez każdą jego krawędź ?

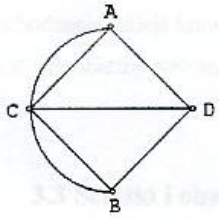
Def. Drogą Eulera nazywamy drogę zamkniętą, która zawiera wszystkie krawędzie danego grafu. Graf który składa się z drogi Eulera nazywamy **grafem Eulera.**

Grafy Eulera są spójne (o ile nie zawierają wierzchołków izolowanych)

Twierdzenie 1

Dany graf G jest grafem Eulera wtedy i tylko wtedy, gdy wszystkie wierzchołki G są stopnia parzystego.

Uwaga: Zagadnienie mostów królewskich można przedstawić na grafie



Twierdzenie 1 mówi nam, że ten graf nie jest grafem Eulera (istnieją wierzchołki stopnia nieparzystego)

Dowód Tw.1

(=>)

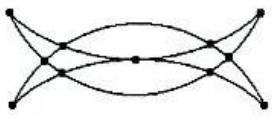
Załóżmy, że G jest grafem Eulera. Zatem zawiera pewną drogę Eulera (zamkniętą). Poruszając się po tej drodze przy przejściu przez dowolny wierzchołek V, w stopniu tego wierzchołka uwzględniane są dwie „nowe" krawędzie jedna „wchodząca" druga „wychodząca" z wierzchołka. Pamiętając o tym, że przez każdą krawędź można przejść tylko 1 raz, widać, że stopień wierzchołka jest wielokrotnością liczby 2, czyli jest parzysty (również wierzchołek końcowy).

(<=) Załóżmy, że wszystkie wierzchołki grafu G są stopnia parzystego. Konstruujemy teraz drogę począwszy od dowolnie wybranego wierzchołka V i przechodząc przez krawędzie G tak, że żadnej krawędzi nie przebywa się dwukrotnie. Powtarzamy te przejścia tak długo, jak to jest możliwe. Ponieważ stopień każdego wierzchołka jest parzysty, możemy wyjść z każdego wierzchołka, do którego weszliśmy. Wędrówka musi się zakończyć w wierzchołku V. Przeszliśmy zatem pewną zamkniętą drogę h. Jeśli zawiera ona wszystkie krawędzie z G, to G jest grafem Eulera. Jeżeli nie, to usuwamy z G wszystkie krawędzie należące do drogi h i otrzymujemy podgraf h' grafu G utworzony z pozostałych krawędzi. Oczywiście stopnie wierzchołków z h' są również parzyste (jak w G). Ponadto ponieważ G jest spójny h' musi się stykać z h co najmniej w jednym wierzchołku A. Zaczynając od tego wierzchołka możemy znowu skonstruować nową drogę w h', która zakończy się w wierzchołku A. Tę drogę z h' można połączyć z drogą h tworząc nową drogę dłuższą niż h, która zaczyna się i kończy w V. Proces ten można powtarzać, dopóki nie otrzymamy drogi zamkniętej, która przechodzi przez wszystkie krawędzie grafu G. Zatem G jest grafem Eulera.

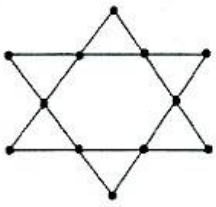
c. n. d.

Przykłady grafów Eulera (narysowane bez odrywania ołówka, każda krawędź 1 raz)

szabla Mahometa



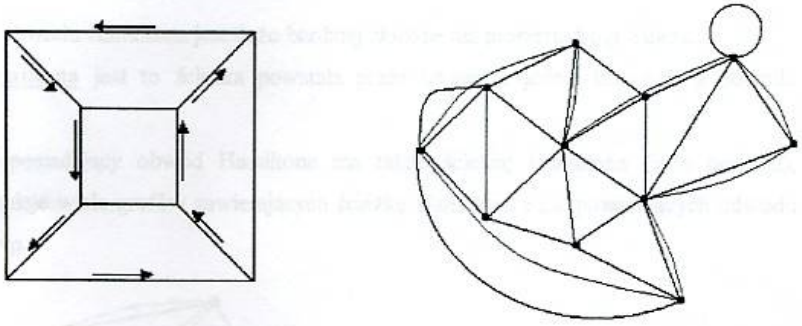
gwiazda Dawida



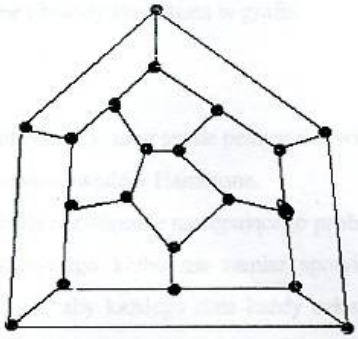
Droga jednobieżna (otwarta droga Eulera) jest to droga otwarta, która zawiera wszystkie krawędzie grafu bez przechodzenia jakiejś krawędzi dwukrotnie.
Oczywiście graf spójny jest jednobieżny <=> ma dokładnie dwa wierzchołki stopnia nieparzystego.

3.3 Ścieżki i obwody Hamiltona.

Obwód Hamiltona – jest to droga zamknięta w grafie spójnym G, która przechodzi przez każdy wierzchołek grafu G. Dokładnie jeden raz (z wyjątkiem wierzchołka początkowego)
Np.



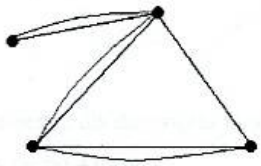
Zaznaczono obwody Hamiltona
Obwód Hamiltona w grafie o n wierzchołkach składa się dokładnie z n krawędzi.
Słynny matematyk irlandzki Sir William Rowan Hamilton w r. 1859 po raz pierwszy postawił problem: jaki jest warunek konieczny i dostateczny na to, aby graf spójny G miał obwód Hamiltona? Problem ten jest dotąd nierozwiązany....
Hamilton zrobił dwunastościan regularny z drewna, którego każdy z 20 wierzchołków oznaczono nazwą pewnego miasta. Łamigłówka polegała na wystartowaniu z jakiegokolwiek miasta i znalezieniu drogi wzdłuż krawędzi dwunastościanu tak, żeby przejść przez każde miasto dokładnie raz i powrócić do miasta początkowego. Graf dwunastościanu i jeden z obwodów Hamiltona jest pokazany na rys:



Zagadnienie obwodu Hamiltona jest dużo bardziej złożone niż problem drogi Eulera.

Ścieżka Hamiltona jest to ścieżka powstała przez usunięcie jednej krawędzi z obwodu Hamiltona.

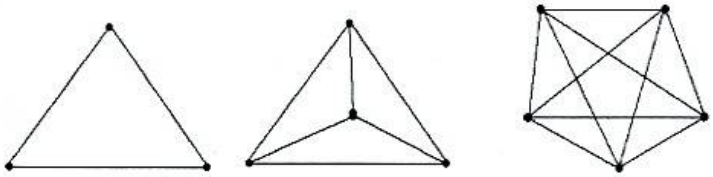
Każdy graf posiadający obwód Hamiltona ma także ścieżkę Hamiltona (jako podgraf), natomiast istnieje wiele grafów zawierających ścieżkę Hamiltona i nie posiadających odwodu Hamiltona. Np.



(nie ma obwodu ale ma ścieżkę Hamiltona)

Długość ścieżki Hamiltona w grafie spójnym o n wierzchołkach wynosi n-1.

Jest pewna klasa grafów, które na pewno mają obwód Hamiltona. Są to grafy pełne (przyp. Każdy wierzchołek połączony jest z każdym jrdną krawędzią) o liczbie wierzchołków ≥ 3 np.



W grafie pełnym łatwo skonstruować obwód Hamiltona: Niech wierzchołki będą ponumerowane V_1, V_2, ...V_n. Ponieważ istnieje krawędź między każdymi dwoma wierzchołkami możemy wystartować z V_1 i przejść do V_2, V_3, ... itd. do V_n. Na koniec z V_n do V_1. Jest to obwód Hamiltona

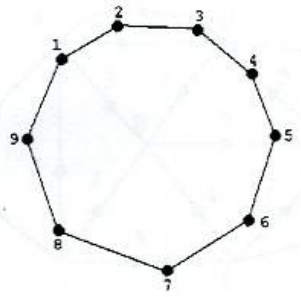
Dany graf może mieć więcej niż jeden obwód Hamiltona. Interesują nas wszystkie krawędziowo rozłączne obwody Hamiltona w grafie.

Twierdzenie 2

Jeśli n jest liczbą nieparzystą ≥3, to w grafie pełnym o n wierzchołkach jest (n-1)/2 rozłącznych krawędziowo obwodów Hamiltona.

Twierdzenie to umożliwia rozwiązanie następującego problemu:

Dziewięciu członków nowego klubu ma zamiar spotykać się codziennie na lunchu przy okrągły stole i siadać tak, aby każdego dnia każdy członek klubu miał innych sąsiadów. Ile dni to może trwać? Traktując członków klubu jako wierzchołki, a relacje siedzenia obok siebie jako krawędzie otrzymujemy graf



Jedno ułożenie to: 1 2 3 4 5 6 7 8 9, inne to: 1 3 5 2 7 4 9 6 8

Zgodnie z twierdzeniem 2 dla dziewięciu ludzi istnieją 4 takie rozmieszczenia. Zatem zmiana sąsiadów zakończy się po 4 dniach.

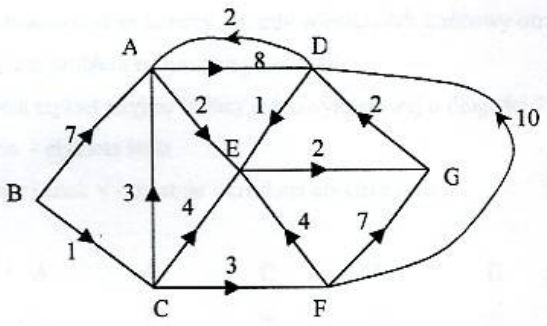
Twierdzenie 3 (G. A. Durac)

Jeśli w grafie G o n wierzchołkach (prostym) stopień każdego wierzchołka wynosi co najmniej n/2, to w grafie G istnieje obwód Hamiltona.

4. Znajdowanie najkrótszych dróg w grafie.

4.1. Znajdowanie najkrótszej drogi między dwoma miastami (Algorytm Dijskry).

Przypuśćmy, że dana jest „mapa" jak na rysunku, na której litery odpowiadają miastom, które są połączone drogami jednokierunkowymi. Jaka jest długość najkrótszej drogi z B do G, jeśli znane są długości dróg między miastami. Szukamy drogi, w której miasta się nie powtarzają, a zatem szukamy ścieżki.



Zauważmy, że liczby na rysunku mogą równie dobrze oznaczać czas lub koszt przejazdu, wtedy szukalibyśmy najszybszej lub najtańszej drogi. Problem ten będziemy rozpatrywali traktując mapę jako graf spójny, w którym każdej krawędzi przypisano nieujemną liczbę rzeczywistą. Graf taki nazywamy **grafem obciążonym** zaś liczbę przypisaną krawędzi **wagą** tej krawędzi i oznaczamy dij (waga krawędzi skierowana z wierzchołka I do J). Problem polega na znalezieniu ścieżki z B do G o najmniejszej wadze sumarycznej.

Opis algorytmu Dijskry

W algorytmie tym etykietuje się wierzchołki danego grafu skierowanego. W każdym kroku w algorytmie niektóre wierzchołki mają stałe etykiety, a inne etykiety tymczasowe. Algorytm rozpoczyna działanie przypisując stałą etykietę równą 0 wierzchołkowi początkowemu S, zaś pozostałym n-1 wierzchołkom etykietę tymczasową ∞ (może to być np. b. duża liczba). Następnie w każdej iteracji inny wierzchołek otrzymuje etykietę stałą zgodnie z zasadami:

1) Każdy wierzchołek j, który nie posiada jeszcze stałej etykiety otrzymuje nową etykietę tymczasową, której wartość określona jest jako:

min{stara etykieta wierzchołka j, stała etykieta wierzchołka i + dij},

gdzie i jest ostatnim wierzchołkiem o stałej etykiecie nadanej w poprzedniej iteracji, a dij jest bezpośrednią odległością między wierzchołkami i oraz j. Jeśli i oraz j nie są połączone krawędzią, to dij = ∞.

2). Znajdujemy najmniejszą wartość spośród wszystkich etykiet tymczasowych i staje się ona etykietą stałą odpowiedniego wierzchołka. (Jeśli takich wierzchołków jest więcej, to należy wybrać którykolwiek z nich).

Etykieta stała danego wierzchołka jest najkrótszą odległością tego wierzchołka od wierzchołka S. (można to udowodnić przy pomocy indukcji matematycznej).

Proces nadawania etykiet kończy się, gdy wierzchołek końcowy otrzyma etykietę stałą.

Rozwiążmy ten problem na naszym przykładzie.

Do określania etykiet użyjmy tablicy jednowymiarowej o długości 7.

Podkreślenie - etykieta stała

Podkreślenie i znak √ - ostatnio określona etykieta stała

Wierzchołki	A	B	C	D	E	F	G
	∞	**0√**	∞	∞	∞	∞	∞
	7	**0**	1	∞	∞	∞	∞
	7	**0**	**1√**	∞	∞	∞	∞
	4	**0**	**1**	∞	5	4	∞
	4	**0**	**1**	∞	5	**4√**	∞
	4	**0**	**1**	14	5	**4**	11
	4√	**0**	**1**	14	5	**4**	11
	4	**0**	**1**	12	5	**4**	11
	4	**0**	**1**	12	**5√**	**4**	11
	4	**0**	**1**	12	**5**	**4**	7
	4	**0**	**1**	12	**5**	**4**	**7√**

Wierzchołek końcowy otrzymuje etykietę stałą

Zatem długość najkrótszej ścieżki wynosi 7.

Powyższy algorytm wyszukuje najkrótszą długość ścieżki. Chcąc znaleźć samą ścieżkę należy poruszać się od wierzchołka końcowego wstecz do wierzchołka początkowego zaliczając do

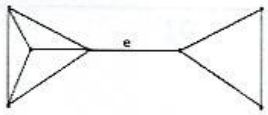
ścieżki tego poprzednika, którego etykieta różni się dokładnie o długość krawędzi łączącej. W naszym przykładzie jest to droga B→C→E→G

Uwaga Gdybyśmy algorytm ten kontynuowali, aż do momentu, gdy wszystkie wierzchołki miałyby etykiety stałe, to zrealizowalibyśmy zadanie znajdowania **najkrótszej drogi z danego wierzchołka do wszystkich pozostałych wierzchołków**

4.2. Problem chińskiego listonosza

W problemie tym listonosz chce dostarczyć całą korespondencję tak, aby przebyć możliwie najkrótszą drogę. Musi on przejść oczywiście każdą z ulic co najmniej 1 raz i chciałby uniknąć przechodzenia przez ulicę wielokrotnie (o ile to możliwe). Problem znów rozważymy w terminach grafów obciążonych, gdzie graf odpowiada sieci ulic, a waga każdej krawędzi odpowiada długości ulicy. Zadanie polega na znalezieniu zamkniętej marszruty („droga", w której krawędzie mogą się powtarzać) zawierającej każdą krawędź i mającą najmniejszą możliwie wagę sumaryczną.
Podamy rozwiązanie tego problemu dla grafów eulerowskich. Należy wtedy znaleźć dowolną drogę Eulera w tym grafie. Podamy jeszcze jedną definicję.

Mostem nazywamy krawędź, której usunięcie powoduje podział grafu na dokładnie dwie składowe spójności, np.



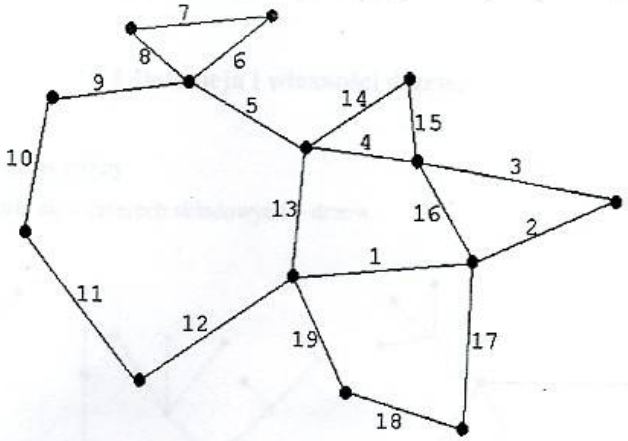
Algorytm Fleury'ego wyznaczania drogi Eulera w grafie eulerowskim.

Twierdzenie 4 Niech G będzie grafem eulerowskim. Następująca konstrukcja jest zawsze wykonalna i prowadzi do drogi Eulera w grafie G. Startując z wierzchołka u (dowolnego) należy przebywać krawędzie w dowolny sposób z zachowaniem jedynie następujących reguł.

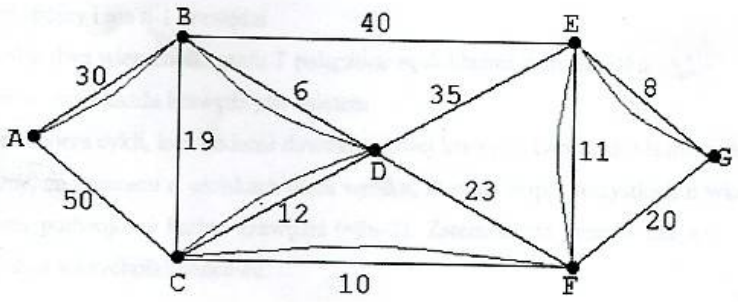
1) usuwać krawędzie, które zostały przebyte i wierzchołki izolowane, które się pojawiają.

2) korzystać z mostu jedynie wtedy, gdy nie ma innej możliwości

np. W poniższym grafie zaznaczono kolejnymi numerami przykładową trasę listonosza.



ćw. (1) Znaleźć najkrótszą drogę z A do G

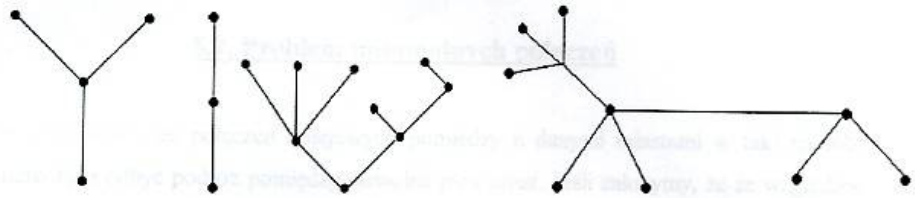


5. Drzewa spinające.

5.1 Defenicja i własności drzew.

Drzewem nazywamy las spójny

Las na rysunku składa się z czterech składowych – drzew.



Twierdzenie o własnościach drzew

Niech T będzie grafem o n wierzchołkach. Wówczas następujące twierdzenia są równoważne.

(1) T jest drzewem

(2) T nie zawiera cykli i ma n-1 krawędzi

(3) T jest spójny i ma n-1 krawędzi

(4) dowolne dwa wierzchołki grafu T połączone są dokładnie jedną ścieżką

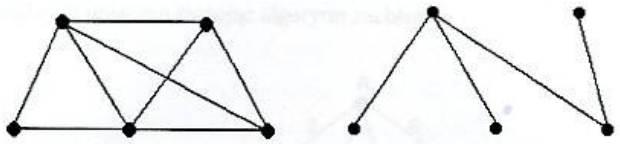
(5) T jest spójny i każda krawędź jest mostem

(6) T nie zawiera cykli, lecz dodanie dowolnej nowej krawędzi tworzy dokładnie jeden cykl

Zauważmy, że z tematu o uściskach dłoni wynika, iż suma stopni wszystkich n wierzchołków jest równa podwojonej liczbie krawędzi (=2n-2). Zatem każde drzewo dla $n \geq 2$ posiada co najmniej dwa wierzchołki końcowe.

Mając dowolny graf spójny G, możemy wybrać w nim cykl i usunąć jedną z jego krawędzi zachowując spójność grafu G. Procedurę powtarzamy dla dowolnych pozostałych cykli do chwili, gdy nie ma już żadnych cykli. Graf, który pozostanie będzie drzewem łączącym wszystkie wierzchołki grafu G, nazywamy go **drzewem spinającym grafu G.**

Np. graf i jedno z jego drzew spinających

Graf moż 

5.2. Problem minimalnych połączeń

Chcemy zbudować sieć połączeń kolejowych pomiędzy n danymi miastami w taki sposób, aby można było odbyć podróż pomiędzy dowolną parą miast. Jeśli założymy, że ze względów ekonomicznych długość torów ma być minimalna, to jest oczywiste, że graf, którego wierzchołki odpowiadają miastom, a krawędzie liniom kolejowym – musi być drzewem.

Problem polega na znalezieniu dobrego algorytmu dla określenia, które z możliwych drzew łączących te miasta wymaga najmniejszej długości torów (znamy odległości między każdą parą miast).

Problem ten znowu rozpatrujemy w terminach grafów obciążonych.

W(e) – waga krawędzi e.

Musimy znaleźć drzewo spinające T o najmniejszej możliwej wadze sumarycznej. W(T)

Istnieje prosty algorytm rozwiązania tego problemu – tzw. **algorytm zachłanny** (lub algorytm **Kruskala**). Algorytm ten podany jest w twierdzeniu:

Twierdzenie

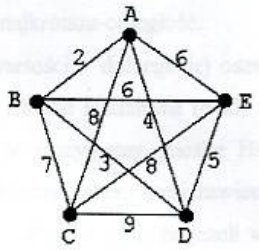
Niech G będzie grafem spójnym o n wierzchołkach. Następująca procedura daje rozwiązanie problemu minimalnych połączeń:

1) niech e_1 będzie krawędzią o najmniejszej wadze w grafie G

2) określić e_2, e_3, ..., e_{n-1} wybierając na każdym etapie krawędź (jeszcze nie wybraną) o najmniejszej możliwej wadze i taką, że nie tworzy ona cykli z poprzednimi krawędziami e_i.

Poszukiwanym drzewem spinającym jest wówczas podgraf T grafu G, którego krawędziami są $e_1, e_2, ..., e_{n-1}$.

Przykład Dany jest graf przedstawiający 5 miast i odległości między nimi. Rozwiązać problem minimalnych połączeń stosując algorytm zachłanny.



e_1 = AB (w(e)=2)

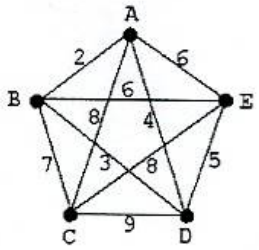
e_2 = BD (w(e)=3)

e_3 = DE (w(e)=5, nie może być AD gdyż utworzyłby się cykl)

e_4 = BC (w(e)=7, nie bierzemy AE i BE, bo utworzyłyby się cykle)

5.3. Problem komiwojażera

W problemie tym komiwojażer chce odwiedzić pewną liczbę miast i wrócić do punktu startu tak, aby przebyć w sumie najkrótszą drogę. Np. jeśli mamy 5 miast: A, B, C, D, E i odległości między nimi takie jak na rysunku,



To najkrótszą możliwą trasą jest A→B→D→E→C→A co daje łączną długość 26.

Problem ten znów rozpatrujemy w terminach grafów obciążonych. W tym przypadku należy znaleźć najkrótszy obwód Hamiltona w obciążonym grafie pełnym. Zauważmy, że tak jak w problemie najkrótszej drogi wagi mają oznaczać czas lub koszt przejazdu.

Jeden z możliwych algorytmów mógłby polegać na obliczeniu długości wszystkich obwodów Hamiltona, ale okazuje się, że jest to zbyt skomplikowane dla większej ilości miast (>5). Zaproponowano szereg innych algorytmów, ale są one albo bardzo wolne, albo tak skomplikowane, że się ich nie stosuje. Z drugiej strony istnieją bardzo efektywne procedury, które podają jaka jest w przybliżeniu najkrótsza odległość.

W celu obliczenia tej przybliżonej wartości – dolnego jej oszacowania – można zastosować algorytm zachłanny. Jeśli weźmiemy obwód Hamiltona (cykl) w obciążonym grafie pełnym i usuniemy dowolny wierzchołek V, to otrzymamy ścieżkę Hamiltona, która jest drzewem. Zatem każde rozwiązanie problemu komiwojażera musi zawierać drzewo spinające tego typu plus dwie krawędzie incydentne z V. Wynika stąd, że jeżeli weźmiemy wagę „najlżejszego" drzewa spinającego (w grafie bez wierzchołka V) otrzymaną za pomocą algorytmu zachłannego i dodamy dwie najmniejsze wagi krawędzi incydentnych z V, to otrzymamy dolne oszacowanie dla rozwiązania problemu komiwojażera.

Np. Jeśli weźmiemy graf przedstawiony na rys. i usuniemy wierzchołek C, to pozostały graf zawierać będzie 4 wierzchołki A, B, D, E. Minimalne („najlżejsze") drzewo spinające te 4 wierzchołki składa się z krawędzi AB, BD, DE. A zatem jego waga wynosi 10. Dwiema krawędziami incydentnymi z C o najmniejszej wadze są CB i CA (lub CE) co daje w sumie 15. A więc dolne oszacowanie długości trasy komiwojażera wynosi 25.

Zatem ta metoda daje bardzo dobre przybliżenie faktycznej długości (26)

6. Rozwiązania przykładowych zadań.

Zad 1.

```
{Program znajduje wszystkie składowe spójności w grafie
skierowanym metoda przeszukiwania wszerz (BFS).}
Program skladowe;
TYPE
Pkrawedz  =^krawedz;
krawedz = record
nast.     :pkrawedz;
nr  :byte;                          { wierzchołka docelowego}
end;
kolejka= record
Q       : array[1..100] of byte;
Head    : byte;
Tail    : byte;
          end;
VAR
n, m    :byte;  {ilość wierzchołków ( n ) i krawędzi ( m ) }
graf    :array[1..100] of pkrawedz; {graf reprezentowany
                                     przez listy sąsiedztwa}
l_skladowych :byte;              {liczba składowych spójności}
sklad :array[1..100] of pkrawedz; {tutaj są  pamiętane
                                   wszystkie składowe spójności}
Queue :kolejka;                  {potrzebna do przeszukiwania wszerz}

procedure Wczytaj_Graf;
{Procedura czyta z pliku 'bfs.in' opis grafu.
Opis grafu składa się z: Podanej ilości wierzchołków w grafie ( n
) i ilości krawędzi ( m )
- Opisu krawędzi złożonego z:* wierzchołka początkowego
 * wierzchołka końcowego}
var
inn :text;
i :byte;
n1, n2  :byte;
nowy  :pkrawedz;
begin
  assign(inn,'bfs.in');
  reset(inn);
  readln(inn,n,m);
  for i := 1 to n do
    graf[i]:=nil;
  for i := 1 to m do

    begin
      readln(inn, n1, n2);
      new(nowy);
      nowy^.nr:= n2;
      nowy^.nast:= graf[n1];
      graf[n1]:= nowy;
    end;
```

```
  close(inn);
end;

procedure Zapisz_Skladowe;
{ Procedura zapisuje do pliku 'bfs.out' liczbę składowych
spójności,
 a następnie (w kolejnych wierszach) wszystkie wierzchołki
kolejnych
 składowych.}
var
out   : text;
i, j, k   : byte;
tmp   : pkrawedz;
tabl    : array[1..100] of byte;
begin
  assign(out,'bfs.out');
  rewrite(out);
  writeln(out,l_skladowych);
  for i := 1 to l_skladowych do
    begin
      tmp := sklad[i];
      k:= 1;
      while not (tmp = nil) do
        begin
          tabl[k] := tmp^.nr;
          inc(k);
          tmp := tmp^.nast;
        end;
      for j := k-1 downto 1 do
        write(out,tabl[j],' ');
      writeln(out);
    end;
  close(out);
end;

{ Procedury do obsługi kolejki. Kolejki są implementowane
w pojedynczej
 tablicy. Działanie poszczególnych procedur:
 QueueFirst(x)  - opróżnia całą kolejkę wstawiając x jako pierwszy
element
 EnQueue(x)    - dodaje element x do kolejki
 DeQueue       - ściąga element z kolejki
 Czy_Pusta     - sprawdza czy kolejka jest pusta}
procedure QueueFirst(x: byte);
begin
  Queue.Q[1] := x;
  Queue.Head := 1;
  Queue.Tail := 1;
end;

procedure EnQueue(x: byte);
begin
```

```
  inc(Queue.Tail);
  if Queue.Tail = 101 then Queue.Tail := 1;
  Queue.Q[Queue.Tail] := x;
end;

function  DeQueue: byte;
begin
  DeQueue := Queue.Q[Queue.Head];
  inc(Queue.Head);
  if Queue.Head = 101 then Queue.Head := 1;
end;

function Czy_Pusta: boolean;
begin
  if (Queue.Head = Queue.Tail + 1) or
    ( (Queue.Head = 1) and (Queue.Tail = 100) )
      then
        Czy_Pusta := true
      else
        Czy_Pusta := false;
  end;

{ Dodatkowa procedura dodająca wierzchołek do listy składowych
spójności}

procedure Dodaj(co, gdzie: byte);
var nowy  : pkrawedz;
begin
  new(nowy);
  nowy^.nr      := co;
  nowy^.nast.   := sklad[gdzie];
  sklad[gdzie]  := nowy;
end;

type tkolor = (BIALY, SZARY, CZARNY);
{ Główna procedura programu. Wyszukuje składowe poprzez
uruchamianie algorytmu przeszukiwania wszerz na wszystkich białych
wierzchołkach (wierzchołek jest:  BIALY   jeśli nie został jeszcze
odwiedzony, SZARY   jeśli został odwiedzony, ale jeszcze nie
wszystkie jego następniki zostały odwiedzone, CZARNY  jeśli jego
wszystkie następniki zostały odwiedzone)}

procedure Znajdz_Skladowe;
var
    kolor   : array[1..100] of tkolor;  {kolor wierzchołka -
                                         początkowo biały}
  i, u    : byte;
  tmp   : pkrawedz;
begin
  for i := 1 to n do                        {krótka pętla mająca
    kolor[i]    := BIALY;                  na celu ustawienie koloru}
  l_skladowych := 0;
  sklad[1]      := nil;
```

```
{Musimy wykonać przeszukiwanie wszerz dla każdego wierzchołka, aby
znaleźć wszystkie składowe spójności. Pomijamy już odwiedzone
wierzchołki (szare i czarne), gdyż skoro zostały już odwiedzone,
to zostały zapisane na liście składowych.}
  for i := 1 to n do
    begin
      if kolor[i] = BIALY then
{W tym miejscu zaczyna się algorytm BFS (przeszukiwania wszerz)}
        begin
          kolor[i] := SZARY;
          QueueFirst(i);
          while not Czy_Pusta do
            begin
              u := Queue.Q[Queue.Head];
              tmp := graf[u];
              while not (tmp = nil) do
                begin
                    if kolor[tmp^.nr] = BIALY then
                     begin
                        kolor[tmp^.nr] := SZARY;
                        EnQueue(tmp^.nr);
                     end;
                    tmp := tmp^.nast;
                end;
              DeQueue;
              kolor[u] := CZARNY;
              Dodaj(u,l_skladowych+1);
            end;
          inc(l_skladowych);
        end; {Tu następuje zakończenie podstawowego algorytmu BFS}
    end;
end;
BEGIN {początek programu głównego}
  Wczytaj_Graf;
  Znajdz_Skladowe;
  Zapisz_Skladowe;
END. {koniec programu głównego}
```

Zad 2.

```
{------------------------------------------------------------
                                          ----------}
      {Zadanie - szlaki turystyczne }
{------------------------------------------------------------
                                          ----------}
Program
obliczanie_ilosci_wszystkich_cykli_w_grafie_skierowanym;
uses crt,graph;
type graf=array[1..50,1..50] of byte;
var   n1,ilosc :integer;
      macierz :graf;
procedure generuj_graf(var matrix:graf;var n:integer);
var i,j,k:integer;
begin
 Randomize;
  Randomize;
  writeln('Program obliczajacy ilosc cykli w grafie
skierowanym');
  writeln;
  writeln('1 - Losowe generowanie grafu spojnego');
  writeln('2 - Graf przykladowy dla n=5');
  writeln;
  readln(k);
  case k of
    1:begin
        clrscr;
        Writeln('Podaj ilosc wierzcholkow: (  n<8 )');
        readln(n);
        for i:=1 to n do
          for j:=1 to n do
            matrix[i,j]:=random(2);
      end;
    2:begin
        n:=5;
        matrix[1,2]:=1;
        matrix[2,4]:=1;
        matrix[2,5]:=1;
        matrix[3,1]:=1;
        matrix[3,2]:=1;
        matrix[3,3]:=1;
        matrix[4,3]:=1;
        matrix[4,5]:=1;
        matrix[5,2]:=1;
      end;
  end;
  clrscr;
  writeln('Macierz sasiedztwa  :');
  for i:=1 to n do
    begin
```

```
        for j:=1 to n do
          write(matrix[i,j],' ');
      writeln;
    end;
end;
end;

procedure cykle(matrix:graf;n:integer;var ile:integer);
label 1;
var H:graf;
{ H[i,j]=1 , gdy wierzchołek j jest zabroniony z wierzchołka
i. Dzięki temu ta sama ścieżka nie jest rozpatrywana przy
przedłużaniu więcej niż jeden raz}
P:array [1..50] of byte;          { zapamiętuje ścieżkę skierowana
}
W:array [1..50] of byte;          { w[i]=1 , gdy wierzchołek i jest
już                                          w ścieżce }
k,k1,i,j,e,f:integer;                  { k= długość ścieżki }
begin
  ile:=0;
  for i:=1 to n do p[i]:=0;
  k:=1;
  p[1]:=1;
  writeln;
  while p[1]<n do
    begin
      for i:=1 to n do
        for j:=1 to n do H[i,j]:=0;
          for i:=1 to n do W[i]:=0;
      repeat
        k1:=0;
        i:=p[1]+1;
        repeat
          if (matrix[p[k],i]=1) and (h[p[k],i]=0) and( W[i]=0)
            then
              begin
                k:=k+1;
                p[k]:=i;
                W[i]:=1;
                i:=p[1]+1;
              end
            else
              i:=i+1;
        until i>n;
        if k=1
          then k1:=1;
        if matrix[p[k],p[1]]=1
          then
            begin
              ile:=ile+1;
              write('cykl  ',ile,' : ');
```

```
            for e:=1 to k do write(p[e], ' ');
            writeln;
          end;
        if k>1
          then
            begin
              for e:=1 to n do H[p[k],e]:=0;
              H[p[k-1],P[k]]:=1;
              w[p[k]]:=0;
              P[k]:=0;
              k:=k-1;
            end;
      until k1=1;
      p[1]:=p[1]+1;
      p[k]:=p[1];
    end;
end;

BEGIN           {początek programu głównego}
  clrscr;
  generuj_graf(macierz,n1);
  cykle(macierz,n1,ilosc);
  readkey;
END.            {koniec programu głównego}
```

Zad 3.

```
{Program na najkrótszą drogę w grafie algorytm Dijskry}

const max=20;                              {Maxymalna ilość
wierzchołków}
var   k:array[1..max]of array[1..max]of byte;     {Macierz

  krawędzi}
    w:array[1..max]of record
    dr:byte;                                      {Tablica
najkrótszych
      dróg do wierzchołka}
    st:byte;
    end;
    dro:array[1..max]of byte;
    i,n,l,d:integer;
    dokt,odkt:byte;
    ilw:byte;                                  {Ilość wierzchołków}

procedure wczytaj;
var p:text;
begin
  assign(p,'gwaz.txt');
  reset(p);
readln(p,ilw);
for i:=1 to max do  begin
      for n:=1 to max do k[i][n]:=0;
      w[i].dr:=0;
      w[i].st:=0;
    end;
  for i:=1 to ilw do
    begin
      read(p,d);
      repeat
        read(p,n);
        read(p,l);
        if (k[d][n]>l)or(k[d][n]=0)
          then
            begin
              k[d][n]:=l;
              k[n][d]:=l;
            end
      until eoln(p)or eof(p);
      if not eof(p)
        then
          readln(p);
  end;
end;
```

```
procedure next(n:byte;var a:byte);
begin
  repeat
    inc(a)
  until ((k[n][a]<>0)and(w[a].st=0))or (a>ilw)
end;


procedure Dijskra(odkt,dokt:byte);
var num,akt,dl:byte;
begin
  dl:=odkt;
  odkt:=dokt;
  dokt:=dl;
  w[odkt].dr:=0;
  w[odkt].st:=1;
  num:=odkt;
  repeat
    akt:=0;
    repeat
      next(num,akt);
      if
(akt<=ilw)and((w[akt].dr>(w[num].dr+k[num][akt]))or(w[akt].dr=
0))
        then
          w[akt].dr:=w[num].dr+k[num][akt]
    until akt>ilw;
    num:=0;
    akt:=0;
    dl:=255;

    repeat
      inc(num);
      if (w[num].dr<dl)and(w[num].st=0)and(w[num].dr<>0)
        then
          begin
            dl:=w[num].dr;
            akt:=num;
          end
    until num=ilw;
    if akt=0
      then
        begin
          write('Nie da sie przejsc');
          readkey;
          halt;
        end;
    num:=akt;
    w[num].st:=1;
  until num=dokt;
```

```
  writeln('Dlugosc=>',w[num].dr);
  for i:=1 to ilw do
    begin
      w[i].st:=0;
      dro[i]:=0;
    end;
  dro[ilw+1]:=0;
end;


procedure znajdz(n,nr:byte);
var a:byte;
begin
  dro[nr]:=n;
  if n<>dokt
    then
      begin
        a:=0;
        repeat
          next(n,a);
        until ((w[n].dr-k[n][a])=w[a].dr)or(a>ilw);
      if (a<=ilw)
        then
          begin
            znajdz(a,nr+1)
          end
      end;
end;

begin                                   {początek programu głównego}
  clrscr;
  wczytaj;
  write('Wierzcholkow jest ',ilw,' od ktorego mam zaczac? ');
  readln(odkt);
  write('Wierzcholkow jest ',ilw,' do ktorego mam isc? ');
  readln(dokt);
  dijskra(odkt,dokt);
  znajdz(odkt,1);
  i:=0;
  repeat
    inc(i);
    write(dro[i],' ');
  until dro[i+1]=0;
  readkey;
end.                                    {koniec programu głównego}
```

Irena Wasilewin

DYREKTOR

mgr Marian Sajna

VI Liceum Ogólnokształcące
im. Jana i Jędrzeja Śniadeckich
85-014 Bydgoszcz, ul. Staszica 4
tel./fax 327 03 71, 322 37 41
NIP 554-11-01-330 REGON 000220575