GRAFURI ORIENTATE

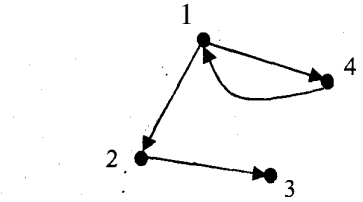
ASPECTE TEORETICE

1. Noţiunea de graf orientat

Definiţie. *Se numeşte **graf orientat** o pereche ordonată de mulţimi notată G=(V, U), unde:*
V : este o mulţime, finită şi nevidă, ale cărei elemente se numesc noduri sau vârfuri;
U : este o mulţime, de perechi ordonate de elemente distincte din V, ale cărei elemente se numesc arce.
• ***Exemplu de graf orientat:***
G=(V, U) unde: V={ 1,2,3,4}
U={{1,2}, {2,3},{1,4}}
Demonstraţie:
Perechea G este graf orientat deoarece respectă întocmai definiţia prezentată mai sus, adică:
V : este finită şi nevidă:
U : este o mulţime de perechi ordonate de elemente din V.

În continuare, vom nota submulţimea {x,y}, care reprezintă **un arc**, cu (x,y) (într-un graf orientat arcul (x,y) este diferit de arcul (y,x)). În baza celor spuse anterior, graful prezentat în exemplul de mai sus se reprezintă textual astfel:
G=(V, U) unde: V={ 1,2,3,4}
U={(1,2), (2,3), (1,4) }
În teoria grafurilor orientate se întâlnesc frecvent noţiunile:
- **extremităţile unui arc**
• fiind dat arcul u=(x,y), se numesc extremităţi ale sale nodurile x şi y;
x se numeşte extremitate iniţială;
y se numeşte extremitate finală.
- **vârfuri adiacente**
• dacă într-un graf există arcul u=(x,y) (sau u=(y,x), sau amândouă), se spune despre nodurile x şi y că sunt adiacente;
- incidenţă
• dacă ul şi u2 sunt două arce ale aceluiaşi graf, se numesc incidente dacă au o extremitate comună.
Exemplu. u1=(x,y) şi u2=(y,z) sunt incidente;
• dacă u1=(x,y) este un arc într-un graf, se spune despre el şi nodul x, sau nodul y, că sunt incidente.

Reprezentarea unui graf orientat admite două forme, şi anume:
- reprezentare textuală: aşa cum s-a reprezentat graful din exemplul anterior;
- reprezentare grafică : arcele sunt reprezentate prin săgeţi orientate, iar nodurile prin puncte.
• ***Exemplu de graf orientat reprezentat textual:***
G=(V, U) unde: V={ 1,2,3,4}
U={(1,2), (2,3), (1,4), (4,1)}
• ***Exemplu de graf orientat reprezentat grafic*** (este graful de la exemplul anterior):



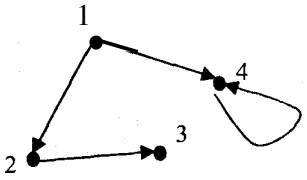
Alte definiţii

Definiţie. *Se numeşte **graf orientat** o pereche ordonată de mulţimi notată G=(V, U), unde:*
V : este o mulţime, finită şi nevidă, ale cărei elemente se numesc noduri sau vârfuri;
U : este o mulţime, de perechi ordonate de elemente din V, ale cărei elemente se numesc arce.

Această definiție diferă de prima definiție prin faptul ca acum nu se mai spune despre extremitățile unui arc ca trebuie să fie distincte. În baza acestei definiții, sunt permise şi arce de genul: u=(x,x) unde x∈ V; aceste arce se numesc **bucle**.

• ***Exemplu de graf orientat*** (reprezentat grafic):

V={1,2,3,4} U={(1,2),(2,3),(1,4), (4,4)}



Definiție. *Se numeşte **graf orientat** o pereche ordonată de mulțimi notată G=(V, U), unde:*

V : este o mulțime, finită şi nevidă, ale cărei elemente se numesc noduri sau vârfuri;

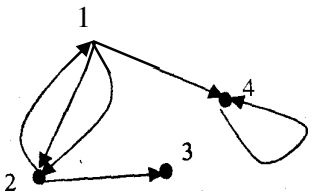
U : este o familie de perechi ordonate de elemente din V, numită familia de arce.

Această definiție diferă de cea anterioară prin faptul ca acum nu numai că se admit bucle, dar se admit şi mai multe arce identice.

• Exemplu de graf orientat (reprezentat grafic):

V={1,2 3,4}

U={(1,2), (1,2), (2,1), (1,4), (2,3), (4,4)}



Observație. Dacă într-un graf orientat **numărul arcelor identice nu depăşeşte numărul p**, atunci se numeşte **p-graf.**

2. Noțiunea de graf parțial

Definiție. Fie G=(V, U) un graf orientat. Se numeşte graf parțial, al grafului G, graful orientat $G_1=(V, U_1)$ unde $U_1 \subseteq U$.

• ***Atenție!*** Citind cu atenție definiția, de mai sus, tragem concluzia:

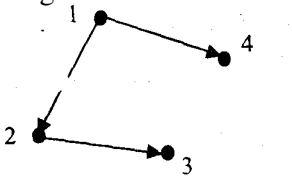
Un graf parțial, al unui graf orientat G=(V, U), are aceeaşi mulțime de vârfuri ca şi G, iar mulțimea arcelor este o submulțime a lui U sau chiar U.

• ***Exemplu***: Fie graful orientat

G=(V, U) unde: V={ 1,2,3,4}

U={(1,2), (l,4), (2,3)}

reprezentat grafic astfel:



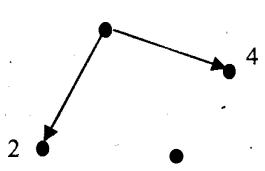
1. Un exemplu de graf parțial al grafului G este graful orientat:

$G_1=(V, U_1)$ unde: V={ 1,2,3,4}

U_1={(1,2),(1,4)}

(s-a eliminat arcul (2,3))

reprezentat grafic astfel:



3

2. Un exemplu de graf parțial al grafului G este graful orientat:

reprezentat grafic astfel:

$G_1=(V, U)$ unde: V={ 1,2,3,4}

U_1=Ø

(s-au eliminat toate arcele)

1•

•4

2 •

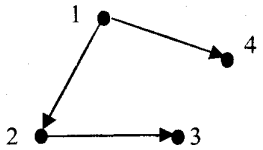
•3

Observaţie. Fie G=(V, U) un graf orientat. **Un graf parţial**, al grafului G, se **obţine păstrând vârfurile şi eliminând eventual nişte arce** (se pot elimina şi toate arcele sau chiar nici unul).

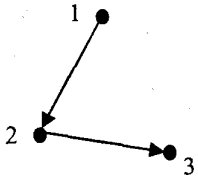
3. Noţiunea de subgraf

Definiţie. *Fie G=(V, U) un graf orientat. Se numeşte* ***subgraf****, al grafului G,* ***graful orientat*** $G_1=(V_1,U_1)$ *unde* $V_1 \subseteq V$ *iar* U_1 *conţine toate arcele din U care au extremităţile în* V_1.

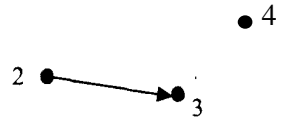
• ***Exemplu***: Fie graful orientat
G=(V, U) unde: V={ 1,2,3,4}
U={(1,2), (2,3), (1,4)}
reprezentat grafic astfel:



1. Un exemplu de **subgraf al grafului G** este graful orientat:
$G_1=(V_1, U_1)$ unde: V_1={1,2,3}
(s-a şters nodul4)
U_1={(1,2),(2,3)}
(s-a eliminat arcul (1,4))
reprezentat grafic astfel:



2. Un exemplu de **subgraf al grafului** G este graful orientat:
$G_1=(V_1, U_1)$ unde: V_1={2,3,4}
(s-a eliminat nodul 1)
U_1={(2,3)}
(s-au eliminat arcele (1,4) (1,2))
reprezentat grafic astfel:



Observaţie. Fie G=(V, U) un graf orientat. **Un subgraf,** al grafului G, **se obţine ştergând eventual anumite vârfuri şi odată cu acestea şi arcele care le admit ca extremitate** (nu se pot şterge toate vârfurile deoarece s-ar obţine un graf cu mulţimea vârfurilor vidă).

4. Gradul unui vârf

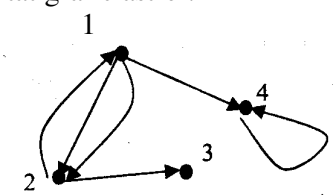
Având la bază ideea că "raportat la un vârf există arce care ies din acel vârf şi arce care intră în acel vârf", au luat naştere următoarele noţiuni:
- grad exterior
- grad interior

care vor fi prezentate in continuare.

Grad exterior

Definiţie. *Fie G=(V, U) un graf orientat si x un nod al său. Se numeşte* ***grad exterior al nodului x****, numărul arcelor de forma (x,y) (adică numărul arcelor care ies din x), notat* $\boldsymbol{d^+(x)}$.

• ***Exemplu:*** Fie graful orientat:
G=(V, U) unde: V={ 1,2,3,4}
U={(1,2), (1,2), (2,1), (1,4), (2,3)> (4,4)}
reprezentat grafic astfel:



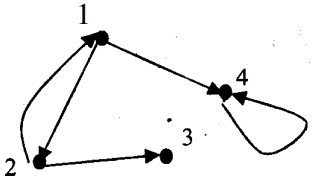
Gradul exterior al nodului 1 este $d^+(1)=3$ (în graf, sunt trei arce care ies din 1).
Gradul exterior al nodului 2 este $d^+(2)=2$ (în graf, sunt două arce care ies din 2).

Gradul exterior al nodului 3 este $d^+(3)=0$ (în graf, nu sunt arce care ies din 3).
Gradul exterior al nodului 4 este $d^+(4)=1$ (în graf, este un arc care iese din 4 (bucla)).

Observații.

1. **Mulțimea succesorilor lui x** se notează cu $\Gamma^+(x)$ şi se reprezintă astfel:
$$\Gamma^+(x)=\{y\in V|(x,y)\in U\}$$
2. **Mulțimea arcelor ce ies din x** se notează cu $\omega^+(x)$ şi se reprezintă astfel:
$$\omega^+(x)=\{(x,y)\in U|y\in V\}$$
3. **Legat de cardinalele mulțimilor** $\Gamma^+(x)$ şi $\omega^+(x)$ putem scrie:
$$|\Gamma^+(x)|=|\omega^+(x)|=d^+(x)$$

Raportat la graful prezentat în figura de mai jos,



putem scrie:

$\Gamma^+(1)=\{2,4\}$ $\omega^+(1)=\{(1,2),(1,4)\}$ $|\Gamma^+(1)|=|\omega^+(1)|=d^+(1)=2$

$\Gamma^+(2)=\{1,3\}$ $\omega^+(2)=\{(2,1),(2,3)\}$ $|\Gamma^+(2)|=|\omega^+(2)|=d^+(2)=2$

$\Gamma^+(3)=\emptyset$ $\omega^+(3)=\emptyset$ $|\Gamma^+(3)|=|\omega^+(3)|=d^+(3)=0$

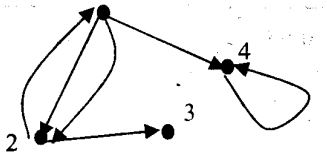
$\Gamma^+(4)=\{4\}$ $\omega^+(4)=f(4,4)\}$ $|\Gamma^+(4)|=|\omega^+(4)|=d^+(4)=1$

Grad interior

Definiție. *Fie G=(V, U) un graf orientat şi x un nod al său. Se numeşte* ***grad interior al nodului x,*** *numărul arcelor de forma (y,x) (adică numărul arcelor care intră in x), notat* $\boldsymbol{d^-(x)}$.

• ***Exemplu***: Fie graful orientat
G=(V, U) unde: V={ 1,2,3,4}
U= {(1,2), (1,2), (2,1), (1,4), (2,3), (4,4)}

reprezentat grafic astfel:



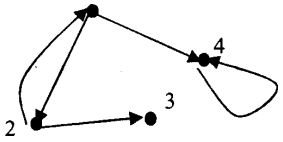
Gradul interior al nodului 1 este $d^-(1)=1$ (în graf, este un arc care intră în 1).
Gradul interior al nodului 2 este $d^-(2)=2$ (în graf, sunt două arce care intră în 2).
Gradul interior al nodului 3 este $d^-(3)=1$ (în graf, este un arc care intră în 3).
Gradul interior al nodului 4 este $d^-(4)=2$ (în graf, sunt două arce care intră în 4).

Observații.

1. **Mulțimea predecesorilor lui x** se notează cu $\Gamma^-(x)$ şi se reprezintă astfel:
$$\Gamma^-(x)=\{y\in V|(y,x)\in U\}$$
2. **Mulțimea arcelor ce intră in x** se notează cu $\omega^-(x)$ şi se reprezintă astfel:
$$\omega^-(x)=\{(y,x)\in U|x\in V\}$$
3. **Legat de cardinalele mulțimilor** $\Gamma^-(x)$ şi $\omega^-(x)$ putem scrie:
$$|\Gamma^-(x)|=|\omega^-(x)|=d^-(x)$$

Raportat la graful prezentat în figura de mai jos,

1



putem scrie:

$\Gamma^-(1)=\{2\}$ $\omega^-(1)=\{(2,1)\}$ $\left|\Gamma^-(1)\right|=\left|\omega^-(1)\right|=d^-(1)=1$

$\Gamma^-(2)=\{1\}$ $\omega^-(2)=\{(1,2)\}$ $\left|\Gamma^-(2)\right|=\left|\omega^-(2)\right|=d^-(2)=1$

$\Gamma^-(3)=\{2\}$ $\omega^-(3)=\{(2,3)\}$ $\left|\Gamma^-(3)\right|=\left|\omega^-(3)\right|=d^-(3)=1$

$\Gamma^-(4)=\{1,4\}$ $\omega^-(4)=\{(1,4),(4,4)\}$ $\left|\Gamma^-(4)\right|=\left|\omega^-(4)\right|=d^-(4)=2$

Observație. Un **nod** se numeşte **izolat** dacă:

$d^+(x)=d^-(x)=0$

Propoziție. În graful orientat G=(V, U), în care V={x_1, x_2, ...,x_n} şi sunt m arce, se verifică egalitatea :

$$\sum_{i=1}^{n} d^+(x_i)=\sum_{i=1}^{n} d^-(x_i)=m$$

5. Graf complet. Graf turneu

Definiție. *Fie G=(V, U) un graf orientat. Graful G se numeşte* ***graf complet*** *dacă* ***oricare două vârfuri distincte ale sale sunt adiacente****.*

Două vârfuri x şi y sunt adiacente dacă:

- între ele există arcul (x,y), sau

- între ele există arcul (y,x), sau

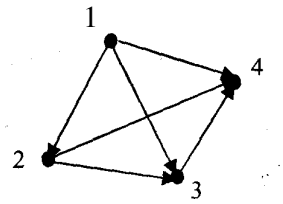
- între ele există arcele (x,y) şi (y,x).

• ***Exemplu de graf orientat complet***:

G=(V, U) unde: V={1,2,3,4}

U={(1,2),,(1,3), (1,4), (2,3), (2,4), (3,4)}

Reprezentarea sa grafică este:



Propoziție. Într-un graf complet cu n vârfuri, există între $\frac{n(n-1)}{2}$ şi n(n-1) arce. (U este privită ca o mulțime (prima definiție a grafului) nu ca o **familie**)

Demonstrație: Numărul cel mai mic de arce, când graful este complet, se obține atunci când nodurile sunt unite doar printr-un singur arc şi se determină astfel:

Pentru fiecare vârf x_i, numărul arcelor care intră şi ies este n-1, deci $d^+(x_i)+ d^-(x_i)=n-1$, pentru orice i=1..n.

Însumând toate aceste relații, obținem:

$$\sum_{i=1}^{n}\left(d^+(x_i)+d^-(x_i)\right)=\sum_{i=1}^{n}(n-1)\Rightarrow\sum_{i=1}^{n}d^+(x_i)+\sum_{i=1}^{n}d^-(x_i)=n(n-1) \qquad (1)$$

Ținând cont de faptul că:

$$\sum_{i=1}^{n} d^+(x_i)=\sum_{i=1}^{n} d^-(x_i)=m$$

relația (1) devine:

$$m+m=n(n-1)\Rightarrow m=\frac{n(n-1)}{2}$$

Numărul cel mai mare de arce, când graful este complet, se obține atunci când nodurile sunt unite prin două arce, adică pentru nodurile x şi y există arcele (x,y) şi (y,x), şi este egal cu

$$2\frac{n(n-1)}{2} = n(n-1)$$

Lema Avem $3^{\frac{n(n-1)}{2}}$ grafuri complete cu n noduri.

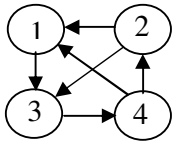
Definiție *Un* ***graf*** *orientat este* ***turneu****, dacă oricare ar fi 2 vârfuri i şi j, i≠j, între ele există un singur arc: arcul(i,j) sau arcul (j,i)*

Proprietăți:

1. Orice graf turneu este graf complet.
2. Avem $2^{\frac{n(n-1)}{2}}$ grafuri turneu cu n noduri
3. În orice graf turneu există un drum elementar care trece prin toate vârfurile grafului.

Problemă. Fiind dat un graf turneu, se cere să se afişeze un drum elementar care trece prin toate vârfurile.

```
#include<iostream.h>
#include<conio.h>
int s[20],a[20][20],n,L,i,j;
void afisare_drum()
{int i; cout<<endl;
for(i=1;i<=n;i++)
cout<<s[i]<<" ";}
void generare_drum()
{int i,j,k,p,q;
if(a[1][2]==1) {s[1]=1;s[2]=2;}
        else{s[1]=2;s[2]=1;}
L=2;
for(k=3;k<=n;k++)
        {if(a[k][s[1]]==1) p=1;
             else{i=1;q=0;
                  while(i<L&&!q)
                  if(a[s[i]][k]==1&&a[k][s[i+1]]==1) q=1;
                  else i++;
                  p=i+1;}
        for(j=L;j>=p;j--)       s[j+1]=s[j];
        s[p]=k;
        L++;}
cout<<endl;}
void main()
{
 cout<<"n=";cin>>n;
 for(i=1;i<=n-1;i++)
        for(j=i+1;j<=n;j++)
        {cout<<"exista arcul ("<<i<<","<<j<<")?[1/0]";
        do{
             cin>>a[i][j];
             } while(!(a[i][j]==0 || a[i][j]==1));
        a[j][i]=1-a[i][j];}
 generare_drum();
 afisare_drum();
getch();}
```



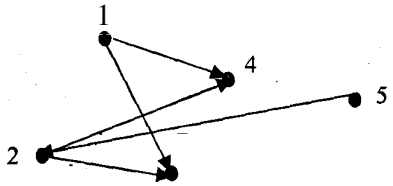

```
n=4
exista arcul (1,2)?[1/0]0
exista arcul (1,3)?[1/0]1
exista arcul (1,4)?[1/0]0
exista arcul (2,3)?[1/0]1
exista arcul (2,4)?[1/0]0
exista arcul (3,4)?[1/0]1


4 2 1 3
```

6. Conexitate

În această secțiune, vor fi prezentate noțiunile:
- lanț
- drum
- circuit
- graf conex
- componentă conexă

Lanț

Definiție. Fie G=(V, U) un graf orientat. Se numeşte lanț, în graful G, o succesiune de arce, notată $L = \lfloor u_{i_1}, u_{i_2}, ..., u_{i_k} \rfloor$ cu proprietatea ca oricare două arce consecutive au o extremitate comună (nu are importanță orientarea arcelor).
Se întâlnesc noțiunile:
- extremitățile lanțului
• fiind dat lanțul $L = \lfloor u_{i_1}, u_{i_2}, ..., u_{i_k} \rfloor$ se numesc **extremități ale sale** extremitatea arcului u_{i1} care nu este comună cu arcul u_{i2} şi extremitatea arcului u_{ik} care nu este comună cu arcul u_{ik-1}
- lungimea lanțului
• fiind dat lanțul $L = \lfloor u_{i_1}, u_{i_2}, ..., u_{i_k} \rfloor$ prin lungimea sa se înțelege numărul de arce care apar în L;
• ***Exemplu de lanț:***
cu reprezentarea grafică astfel:
Fie graful G=(V, U), unde: V={1,2,3,4,5}
U={(1,3),(1,4), (2,3), (2,4), (5,2)}=
{ u_1, u_2, u_3, u_4, u_5 }

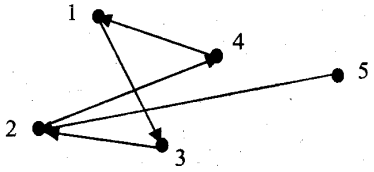


L1=[u_1 ,u_3 ,u_5] este, în graful G, lanț cu lungimea 3 şi extremitățile 1 şi 5.
L2=[u_1 ,u_2, u_4, u_5] este, în graful G, lanț cu lungimea 4 şi extremitățile 3 şi 5.

Atenție! Dacă $L = \lfloor u_{i_1}, u_{i_2}, ..., u_{i_k} \rfloor$ este lanț în graful G, atunci şi $L_1 = \lfloor u_{i_k}, u_{i_{k-1}}, ..., u_{i_1} \rfloor$ este lanț în graful G.

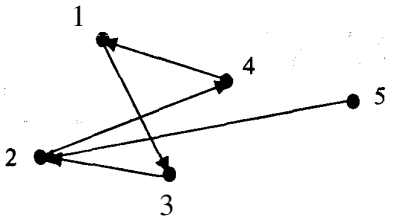
Drum

Definiție. *Fie G=(V, U) un graf orientat. Se numeşte* ***drum****, în graful G, o succesiune de noduri, notată* $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$*, cu proprietatea* $(x_{i_1}, x_{i_2}), ..., (x_{i_{k-1}}, x_{i_k}) \in U$ *(altfel spus* $(x_{i_1}, x_{i_2}), ..., (x_{i_{k-1}}, x_{i_k})$ *sunt arce).*
Se întâlnesc noțiunile:
- **extremitățile drumului**
• fiind dat drumul $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$ se numesc **extremități** ale sale nodurile x_{i1} şi x_{ik} (x_{i1} extremitate inițială şi x_{ik} extremitate finală)
- **lungimea drumului**
• fiind dat drumul $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$, prin **lungimea sa** se **înțelege numărul de arce care apar în cadrul său;**
• ***Exemplu de drum:***
Fie graful G=(V, U) unde:
V={ 1,2,3,4,5 }
U={(1,3), (4,1), (3,2), (2,4), (5,2)}
cu reprezentarea grafică astfel:



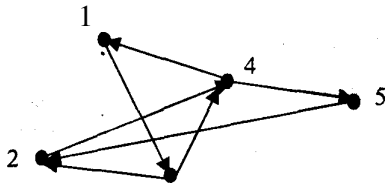
D1=(1, 3, 2) este, în graful G, drum cu lungimea 2 şi extremităţile 1 şi 2.
D2=(4, 1, 3, 2) este, în graful G, drum cu lungimea 3 şi extremităţile 4 şi 2.
Atenţie! Dacă $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$ este drum, în graful G, atunci nu neapărat şi $D1 = (x_{i_k}, x_{ik-1}, ..., x_{i_1})$ este drum, în graful G.

Definiţie. *Fie G=(V,U) un graf orientat. Se numeşte **drum elementar**, în graful G, drumul* $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$ *cu proprietatea că oricare două noduri ale sale sunt distincte (altfel spus, printr-un nod nu se trece decât o singură dată).*
• ***Exemplu***: în graful de mai jos,



drumul D1=(4, 1, 3, 2) este drum elementar.

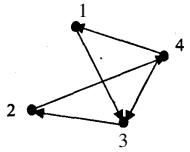
Definiţie. Fie G=(V, U) un graf orientat: Se numeşte **drum neelementar**, în graful G, drumul $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$ cu proprietatea că nodurile sale nu sunt distincte două câte două (altfel spus, prin anumite noduri s-a trecut de mai multe ori).
• ***Exemplu***: în graful de mai jos,



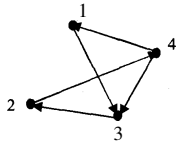
drumul D2=(4, 1, 3, 2, 4, 5, 2) este drum neelementar (prin 4 (şi 2) s-a trecut de două ori).

Circuit

Definiţie: Fie G=(V, I) un graf neorientat. Se numeşte **circuit**, în graful G, drumul $D = (x_{i_1}, x_{i_2}, ..., x_{i_k})$ cu proprietatea că $x_{i1} = x_{ik}$ şi are arcele cel compun diferite două câte două (circuitul se notează în continuare cu C).
• ***Exemplu***: În graful de mai jos,



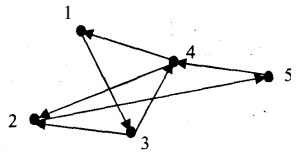
drumul C=(1, 3, 2, 4, 1) este circuit.
Definiţie. Fie G=(V, U) un graf orientat. Se numeşte **circuit elementar**, în graful G, un circuit cu proprietatea că oricare două noduri ale sale, cu excepţia primului şi a ultimului, sunt distincte.
• ***Exemplu***: în graful de mai jos,



circuitul C=(1, 3, 2, 4, 1) este circuit elementar.

Definiție. Fie G=(V, U) un graf orientat. Se numeşte **circuit neelementar**, în graful G, un circuit cu proprietatea că nodurile sale, cu excepția primului şi a ultimului, nu sunt distincte.

• ***Exemplu:*** în graful de mai jos

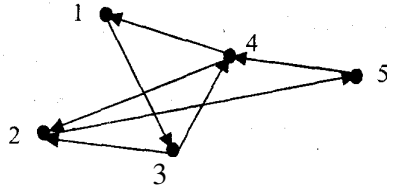


circuitul C=(1, 3, 4, 2, 5, 4, 1) este circuit neelementar (prin 4 s-a trecut de două ori).

Graf conex

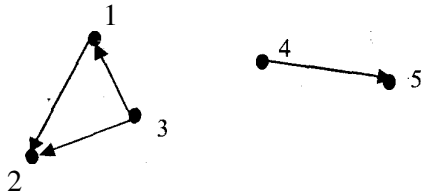
Definiție. Fie G=(V, U) un graf orientat. Graful G se numeşte **conex** dacă pentru oricare două vârfuri x şi y, x#y, **există un lanț** de extremități x şi y.

• ***Exemplu de graf care este conex:***



Graful este conex, deoarece oricare ar fi vârfurile x şi y, x#y, există un lanț in G care să le lege.

• ***Exemplu de graf care nu este conex***:



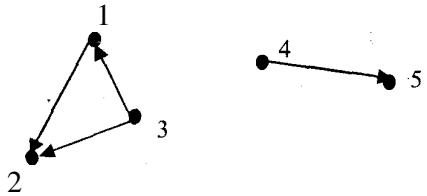
Graful nu este conex, deoarece există două vârfuri, cum ar fi 1 şi 4, pentru care nu există nici un lanț în graf care să le lege.

Componentă conexă

Definiție. Fie G=(V, U) un graf orientat. Se numeşte componentă conexă un graf orientat $G_1=(V_1, U_1)$ care verifică următoarele condiții:

- este subgraf al grafului G;
- este conex;
- nu există nici un lanț în G care să lege un nod din V, cu an nod din $V-V_1$.

• ***Exemplu:*** Fie graful G=(V, U) : V={1,2,3,4,5} şi U={(1,2), (3,1), (3,2), (4,5)}



Pentru graful de mai sus, graful G1=(V1,U1) unde: V1={4,5} şi U1={(4,5)} este componentă conexă, deoarece:

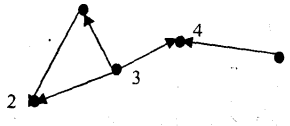
- este subgraf al grafului G;
- este conex;
- nu există nici un lanț în G care să lege un nod din V, cu un nod din $V-V_1$={1,2,3}

La fel, se poate spune şi despre graful G2=(V2,U2) unde: V2={1,2,3) şi U2={(1,2)> (3,1), (3,2)}

În concluzie, graful din figura de mai sus este format din două componente conexe.

Observație. Fie G=(V, U) un graf orientat. Graful G este conex dacă şi numai dacă este format dintr-o singură componentă conexă.

• ***Exemplu de graf conex*** (este format dintr-o singură componentă conexă):



7. Reprezentarea grafurilor orientate

Fie G=(V, U) un graf orientat, unde V={x_1, x_2,..., x_n} şi U={u_1, u_2,..., u_m} . Deoarece între mulţimea {x_1, x_2,..., x_n} şi mulţimea {1, 2,..., n} există o bijecţie, $x_i \leftrightarrow i$, putem presupune, fără a restrânge generalitatea, mai ales pentru a uşura scrierea, că V={1, 2,..., n}.
În baza celor spuse mai sus, mai departe, în loc de x_i vom scrie i, şi în loc de arcul (x_i ,x_j) vom scrie (i ,j).
Pentru a putea prelucra un graf orientat cu ajutorul unui program, trebuie mai întâi să fie reprezentat în programul respectiv.
Pentru a reprezenta un graf orientat, într-un program, există mai multe modalităţi folosind diverse structuri de date; dintre acestea în continuare vom prezenta:
- reprezentarea prin matricea de adiacenţă;
- reprezentarea prin matricea vârfuri-arce;
- reprezentarea prin matricea drumurilor;
- reprezentarea prin listele de adiacenţă;
- reprezentarea prin şirul arcelor.

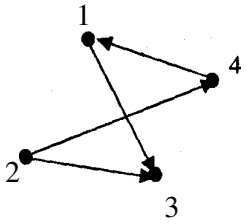
Matricea de adiacenţă

Fie G=(V; U) un graf orientat cu n vârfuri (V={ 1,2, ..., n}) şi m arce.
Matricea de adiacenţă ($A \in M_n(\{0,1\})$), asociată grafului G, este o matrice pătratică de ordin n, cu elementele:

$$a_{i,j} = \begin{cases} 1, \ daca \ (i,j) \in U \\ 0, \ daca \ (i,j) \notin U \end{cases}$$

(altfel spus, $a_{i,j}$=1, dacă există arc între i şi j şi $a_{i,j}$=0 dacă nu există arc între i şi j.

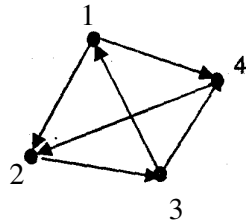
• ***Exemplul 1.***
Fie graful reprezentat ca în figura de mai jos:



Matricea de adiacenţă asociată grafului este:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

• ***Exemplul 2***.
Fie graful reprezentat ca în figura de mai jos:



Matricea de adiacenţă asociată grafului este:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

* ***Comentarii***:
l. Matricea de adiacenţă este o matrice pătratică, de ordin n, şi nu este neapărat simetrică faţă de diagonala principală, aşa cum este în cazul grafurilor neorientate.
Secvenţele de citire a matricei de adiacenţă:

```
int a[100][100];
.................
cout<<"n="; cin>>n;
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        {cout«"a[ "<<i<<","<<j<<"]="; cin»a[i][j];}
```

sau:

```
cout<<"n  m " ; cin>>n>>m;
for (i=1;i<=m;i++)
    {cout«"dati extremitatile arcului "<<i<<" "; cin>>x>>y;
```

```
    a[x][y]=1;}
.....
```

2. Matricea de adiacență are toate elementele de pe diagonala principală egale cu 0 (ne referim la definiția 1, când graful nu are bucle).

3. Numărul elementelor egale cu 1 de pe linia i este egal cu gradul exterior al vârfului i.

```
int gr_ext(int i)
{int j , s;
s=0;
for (j=1;j<=n;j++) s=s+a[i][j];
return s;}
```

4. Numărul elementelor egale cu 1 de pe coloana i este egal cu gradul interior al vârfului i.

```
int gr int(int i)
{int j, s;
s=0;
for (j=1;j<=n;j++) s=s+a[j][i];
return s;}
```

5. Dacă vârful i este un vârf izolat, pe linia i şi coloana i nu sunt elemente egale cu 1.

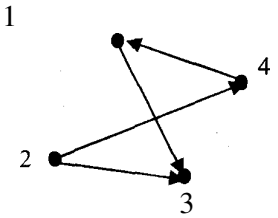
```
int vf_izolat(int i)
{return (gr_ext(i)==0) && (gr_int(i)==0);
}
```

Matricea vârfuri-arce (matricea de incidență)

Fie G=(V, U) un graf orientat cu n vârfuri (V={1,2, ..., n}) şi m arce.
Matricea vârfuri-arce ($B \in M_{nxm}(\{-1,0,1\})$), asociată grafului G, este o matrice cu n linii şi m coloane, cu elementele:

$$b_{i,j} = \begin{cases} -1, & \textit{daca i este extremitate finala pentru arcul } u_j \\ 0, & \textit{daca i nu este extremiate pentru arcul } u_j \\ 1, & \textit{daca i este extremitate initiala pentru arcul } u_j \end{cases}$$

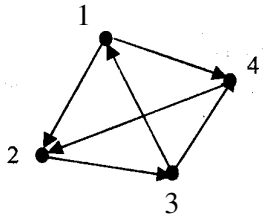
• ***Exemplul 1***. Fie graful G=(V,U) :
V={1,2,3,4},
U={(1,3),(2,3),(2,4),(4,1)}= {u_1, u_2, u_3, u_4}
reprezentat ca in figura de mai jos:



Matricea vârfuri-arce asociată grafului este:

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

• ***Exemplul 2***. Fie graful G :
V={ 1,2,3,4},
U={(1,2),(1,4),(2,3),(3,1),(3,4),(4,2)}=
{u_1, u_2, u_3, u_4,u_5, u_6},
reprezentat ca în figura de mai jos:



Matricea vârfuri-arce asociată grafului este:

$$B=\begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \end{pmatrix}=\begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 & -1 & 1 \end{pmatrix}$$

• ***Comentarii:***
1. Matricea vârfuri-arce nu este neapărat o matrice pătratică.

#***Secvențele de citire a matricei vârfuri-arce:***

```
int b[100][100];
.....................
cout<"n m" ;
cin>>n>>m;
for (i=1;i<=n;i++)
    for (j=1;j<=m;j++)
        {cout<<"b["<<i<<" ,"<<j<<"]=";
        cin>>b[i] [j];}
```

sau:

```
cout<<"n m" ;
cin>>n>>m;
for (i=l;i<=m;i++)
    { cout<<"dati extremitatile arcului "<<i<<" "; cin>>x>>y;
    b[x] [i]=1;
    b[y][i]=-1;}
.....................
```

2. Numărul elementelor egale cu l de pe linia i este egal cu gradul exterior al vârfului i:

```
int gr_ext_B( int i)
{int j, nr;
nr=0;
for (j=1;j<=m;j++)
  if (b[i][j]==1) nr=nr+1;
return nr;}
```

3. Numărul elementelor egale cu -1 de pe linia i este egal cu gradul interior al vârfului i.

```
int gr_int_B( int i)
{int j, nr;
nr=0;
for (j=1;j<=m;j++)
  if (b[i][j]==-1) nr=nr+1;
return nr;}
```

4. Dacă vârful i este un vârf izolat, pe linia i nu sunt elemente egale cu 1 sau -1.

```
int vf_izolat_B( int i)
{return (gr_ext_B(i) ==0) && (gr_int_B(i)==0);}
```

5. Pe fiecare coloană j, există un singur, element egal cu 1 şi un singur element egal cu -1.
Indicele liniei pe care se află 1 este extremitatea inițială a arcului u;
Indicele liniei pe care se află -1 este extremitatea finală a arcului u,

• ***Construirea matricei de adiacență, când se cunoaşte matricea vâfuri-arce.***
- se parcurge matricea vârfuri-arce, de la prima până la ultima coloană, cu j
- pe coloana j, se depistează indicele liniei pe care se află 1. fie acesta plus l;
- pe coloana j, se depistează indicele liniei pe care se află -I, fie acesta minus l;
- în matricea de adiacență elementul a[plus l, minus l] se face 1.

```
for (j=1 j<=m;j++)
        {for (i=1;i<=n;i++)
                if (b[i][j]==1) plusl=i;
                        else if (b[i][j]==-1) minusl=i;
        a[plusl ][minusl]=1;}
```

• ***Construirea matricei vârfuri-arce, când se cunoaşte matricea de adiacență.***
- se foloseşte variabila întreagă k, cu următorul rol:
- reprezintă numărul arcului la care s-a ajuns (la al câtelea element $a_{i,j}$=1 s-a ajuns), care este practic indicele curent al coloanei la care s-a ajuns în matricea vârfuri-arce.
- k=0;
- se parcurge matricea de adiacență, linie cu linie
  - dacă se găseşte un element $a_{i,j}$=1, atunci
    - se măreşte k cu 1;
    - in coloana k, din matricea vârfuri-arce, se trece
      pe linia i valoarea 1
      pe linia j valoarea -1

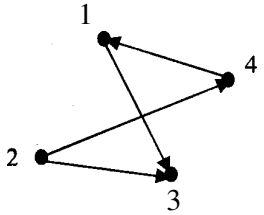
```
.....
k=0;
for (i=1;i<=n;i++)
  for (j=1 j<=n;j++)
        if (a[i][j]==1)
                {k=k+1;
                a[i][k]=1;
                a[j][k]=-l;}
.....
```

Matricea drumurilor

Fie G=(V, U) un graf orientat cu n vârfuri (V={ 1,2, ..., n}) şi m arce.
Matricea drumurilor (D∈ Mn{0,1 })), asociată grafului G, este o matrice cu n linii şi n coloane, cu elementele:

$$d_{i,j} = \begin{cases} 0, & daca\ nu\ exista\ drum\ in\ G\ de\ la\ i\ la\ \ j \\ 1, & daca\ exista\ drum\ in\ G\ de\ la\ i\ la\ \ j \end{cases}$$

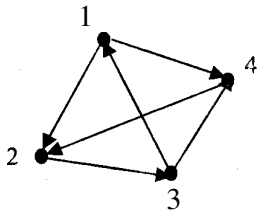
• ***Exemplul 1***.
Fie graful G=(V,U) : V={1,2,3,4}, U={(1,3),(2,3),(2,4),(4,1)} ={u_1, u_2, u_3, u_4},
reprezentat ca în figura de mai jos:



Matricea drumurilor asociată grafului este:

$$D=\begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{pmatrix}=\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

• ***Exemplul 2***.
Fie graful G : V={1,2,3,4},
U={(1,2),(1,4),(2,3),(3,1),(3,4),(4,2)}=
{u_1, u_2, u_3, u_4, u_5, u_6}
reprezentat ca în figura de mai jos:



Matricea drumurilor asociată grafului este:

$$D=\begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{pmatrix}=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

• ***Comentarii:***
1. Matricea drumurilor este o matrice pătratică.
În continuare, este prezentat în pseudocod algoritmul Roy-Warshall de determinare a matricei drumurilor plecând de la matricea de adiacență. Algoritmul constă într-un şir de transformări aplicate matricei de adiacență. Vom spune că există drum de la nodul i la nodul j, dacă găsim un nod k cu proprietatea că există drum de la i la k şi drum de la k la j. Astfel:
Un element ai,j care este 0 devine1, dacă există un nod k a.î. $a_{i,k}$=1 şi $a_{k,j}$=1. Pentru a găsi toate arcele nodului k trebuie parcurse pe rând în variabila k toate nodurile 1,2,.., n.

```
.............
pentru k=1 ... n
  pentru i=1 ... n (i#k)
          pentru j = 1 ... n (j#k)
                  dacă a[i,j]=0 si i!=k si j!=k atunci
                          a[i,j]= a[ik] * a[kj]
.....................
```

2. Dacă în matricea drumurilor d_{ii}=1, înseamnă că există în graf un circuit de extremități i.
3. Dacă în matricea drumurilor linia i şi coloana i au elementele egale cu 0, nodul i este un nod izolat.

****Programul C/C++ de construire şi afişare a matricei drumurilor***.

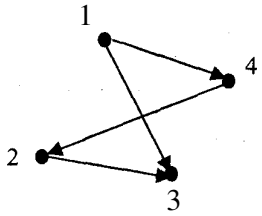
```
#include <iostrearn.h>
#include <stdio.h>
#include <conio.h>
typedef int mat[30][30];
mat a;
int        i, j, k, n, m, x, y;
void main()
{clrscr();
cout<<"n "; cin>>n;
cout<<"m="; cin>>m;                          secvența de citire a
for (i=1;i<=m;i++)                           matricei de adiacență
        {cout<<"arcul "<<i<<"";
        cout<<" x y "; cin>>x>>y; a[x] [y]=1 ;}
for (k=1;k<=n;k++)                           secvența de transformare
   for (i=1;i<=n;i++)                        a matricei de adiacență
      for (j=1;j<=n;j++)                     in matricea drumurilor
```

```
            if (a[i][j]==0)
                a[i][j]=a[i][k] *a[k][j];
cout<<"matricea drumurilor este ";
for (i=1;i<=n;i++)
      {for (j=1;j<=n;j++)
          cout<<a[i] [j] « " ";
       cout<<endl;}
getch();}
```

Liste de adiacență

Fie G=(V, U) un graf orientat, cu n vârfuri (V={ 1,2, ..., n}) şi m arce.
Reprezentarea grafului G, prin liste de adiacență, constă în:
- precizarea numărului de vârfuri n;
- pentru fiecare vârf i, se precizează lista L; a succesorilor săi, adică lista nodurilor care fac parte din mulțimea $\Gamma^+(i)$.

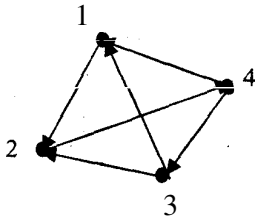
• ***Exemplul 1***. Fie graful din figura de mai jos:



Reprezentarea sa, prin liste de adiacențe, presupune:
- precizarea numărului de vârfuri n, n=4;
- precizarea listei succesorilor lui i, pentru i=1..n

Vârful i	Lista vecinilor lui i
1	3,4
2	3
3	
4	2

• ***Exemplul 2***. Fie graful din figura de mai jos:



Reprezentarea sa, prin liste de adiacențe, presupune:
- precizarea numărului de vârfuri n, n=4;
- precizarea listei vecinilor lui i, pentru i=1..n

Vârful i	Lista vecinilor lui i
1	2
2	4
3	1,2
4	1, 3

• ***Comentarii:***
Acest mod de reprezentare se poate implementa astfel:
1. Se foloseşte un **tablou bidimensional**, caracterizat astfel:
• are n +m coloane;
• $T_{1,i}$=i, pentru i=1..n;
• Pentru i=1..n $T_{2,i}$= k, dacă $T_{1,k}$ este primul nod din lista vecinilor lui i;
$T_{2,i}$ =0, dacă nodul i nu are succesori;
• Dacă $T_{i,j}$=u, adică u este un nod din lista vecinilor lui i, atunci:
$T_{2,j}$=0, dacă u este ultimul nod din lista vecinilor lui i;
$T_{2,j}$=j+1, dacă u nu este ultimul nod din lista vecinilor lui i.
• ***Exemplu*** de completare a tabloului pentru graful de la exemplul 1.
Prima etapă. Se numerotează coloanele (l..n+m), şi se trec vârfurile.

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

A doua etapă. Se trec în tabel vecinii lui 1, începând de la coloana 5.

1	2	3	4	**5**	6	7	8
1	2	3	4	**3**	**4**		
5				**6**	**0**		

$T_{2,1}$=5, pentru că primul vecin (3) al lui 1 s-a trecut la coloana 5 ($T_{1,5}$=3);
$T_{2,5}$=6, pentru că următorul vecin (4) al lui 1 s-a trecut la coloana 6 ($T_{1,}$6=4);
$T_{2,6}$=0, pentru că vecinul $T_{1,6}$ (4) al lui 1 este ultimul din listă.
A treia etapă. Se trec în tabel vecinii lui 2, începând de la coloana 7.

1	**2**	3	4	5	6	**7**	8
1	2	3	4	3	4	**3**	
5	**7**			6	0	**0**	

$T_{2,2}$=7, pentru că primul vecin (3) al lui 2 s-a trecut la coloana 7 ($T_{1,7}$=3);
$T_{2,7}$=0, pentru că vecinul $T_{1,7}$ (3) al lui 2 este ultimul din listă.
A patra etapă. Se trec în tabel vecinii lui 3, începând de la coloana 8.

1	2	**3**	4	5	6	7	8
1	2	3	4	3	4	3	
5	7	**0**		6	0	0	

$T_{2,3}$=0, pentru că 3 nu are succesori, deci lista sa este vidă
Ultima etapă. Se trec în tabel vecinii lui 4, începând de la coloana 8 (aici s-a ajuns)

1	2	3	**4**	5	6	7	8
1	2	3	4	3	4	3	**2**
5	7	0	**8**	6	0	0	**0**

$T_{2,4}$=8, pentru că primul vecin (2) al lui 4 s-a trecut la coloana 8 ($T_{1,8}$= 2);
$T_{2,8}$=0, pentru că vecinul $T_{1,8}$ (2) al lui 4 este ultimul din listă.

2. Se foloseşte un **tablou unidimensional**, cu numele **cap**, şi un **tablou bidimensional**, cu numele **L** (care reţine listele succesorilor pentru fiecare nod), caracterizate astfel:
Tabloul cap:
- are n componente;
- cap_i= c, dacă primul nod din lista vecinilor lui i este trecut în tabloul L la coloana c, adică $L_{1,c}$ este primul vecin al lui i,

şi cap_i =0, dacă nodul i nu are succesori.
Tabloul L:
- are m componente;
- dacă k este un vecin al nodului i, atunci:
  $L_{1,p}$ =k şi $L_{2,p}$=0, dacă k este ultimul vecin din listă, sau
  $L_{1,p}$=k şi $L_{2,p}$=p+1, dacă k nu este ultimul vecin din listă.
  (p este coloana la care s-a ajuns în tabloul L)

• ***Exemplu*** de completare a tablourilor cap şi L, pentru graful de la exemplul 1
Tabloul cap

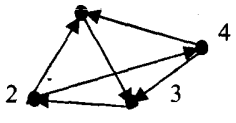
1	2	3	4
1	**3**	**0**	**4**

Tabloul L

1	2	**3**	4
3	4	3	2
2	0	0	0

3. Se foloseşte un **tablou bidimensional**, cu numele L, caracterizat astfel:
- are n linii;
- pe linia i; se trec succesorii nodului i.

• ***Exemplu*** de completare a tabloului L, pentru graful:



Tabloul L

3		
1	**4**	
2		
1	**3**	

• Implementarea în limbajul C++, a ideii prezentate mai sus, se realizează conform secvenței de program prezentată mai jos.

```
.....
int L[20][20];
int nr_vec[20];
cout<<"n="; cin>>n;
for (i=1;i<=n;i++)
        {cout<<"Dati numarul vecinilor nodului "«i; cin»nr_vec[i];
        for (j=1;j<=nr_vec[i];j++)
                        cin>>L[i][j];}
.....
```

• Construirea matricei de adiacență, când se cunoaşte L (listele vecinilor fiecărui nod).

```
.....
for (i=1;i<=n;i++)
        {for (j=1;j<=nr_vec[i];j++)
        a[i][L[i][j]]=1;}
.....
```

• Construirea tabloului L (listele vecinilor nodurilor)**, când se cunoaşte matricea de adiacentă**

```
.....
for (i=1; i<=n;i++)
        {k=0;
        for (j=1;j<=n;j++)
            if (a[i][j]==1)
                {k=k+1;
                L[i][k]=j;}
        }
.....
```

4. Se foloseşte un **tablou unidimensional**, cu numele **L**, caracterizat astfel:
- componentele sale sunt de tip **referință**;
- are **n componente**;
- L_i pointează spre **începutul listei succesorilor nodului i**.

• Construirea matricei de adiacență, când se cunoaşte L (listele vecinilor fiecărui nod).

```
.....
for (i=1;i<=n;i++)
        {c=L[i];
        while (c)
                {a[i] [c->nod]=1;
                c=c->urm;}
        }
.....
```

Şirul arcelor

Fie G=(V, U) un graf orientat, cu n vârfuri (V={ 1,2, ..., n}) şi m arce.
Reprezentarea grafului G constă în precizarea numărului **n de noduri** şi numărului **m de arce** precum şi în precizarea **extremităților pentru fiecare arc** în parte.
• *Comentarii:*
Acest mod de reprezentare se implementează astfel:
1. Se dă numărul n de noduri şi numărul m de arce, iar extremitățile fiecărui arc sunt trecute în vectorii el si e2, astfel:
- extremitățile primului arc sunt el[1] şi e2 [1];
- extremitățile celui de-al doilea arc sunt e1[2] şi e2[2];
.....

- deci, U={(el[1],e2[1]) , (el[2],e2[2]) ,..., (el [m],e2[m])}

• Secvența C++corespunzătoare este:

```
int         el[100], e2[100];
int n, m, i;
cout<<"n="; cin>>n;
cout<<"m="; cin>>m;
for (i=1;i<=m;i++)
      {cout<<"Dati extremitatile arcului cu numarul "<<i<<" ";
      cin>>el[i]>>e2[i];}
.....
```

• **Construirea matricei de adiacență, când se cunoaşte şirul muchiilor ca mai sus**.

```
.....
cout<<"n="; cin>>n;
for (i=1;i<=m;i++)
a[el[i]] [e2[i]]=1;
.....
```

• **Construirea şirului arcelor, ca mai sus, când se dă matricea de adiacență**.

```
.....
k=0;
for (i=1;i<=n;i++)
    for (j==1;j<=n;j++)
       if (a[i][j] ==1)
              { k=k+1;
              e1[k]=i;
              e2[k]=j;}
m=k;
.....
```

2. Se foloseşte an **tablou unidimensional**, cu numele u, caracterizat astfel:
- componentele sale sunt de tip record;
- are m componente;
- u_i reprezintă arcul i.

• Pentru implementare este nevoie de:

```
struct arc{int x;
           int y;};
arc u[20];
.....
```

Accesul la arcul i se face: u[i].x ..... u[i].y

• Secvența C++ corespunzătoare este:

```
cout<<"n="; cin>>n;
cout<<"m "; cin>>m;
for (i=1;i<=m;i++)
          {cout<<"'Dati extremitatile arcului cu numarul "<<i<<" ";
          cin>>u[i].x>>u[i].y;}
```

• **Construirea matricei de adiacentă, când se cunoaşte şirul arcelor ca mai sus:**

```
.....
cout<<"n='"; cin>>n;
for (i=1;i<=m;i++)
a[u[i].x][u[i].y]=1;
.....
```

• **Construirea şirului arcelor, ca mai sus, când se dă matricea de adiacență**:

```
.....
k=0;
for (i=1; i<=n; i++)
    for (j=1;j<=n;j++)
```

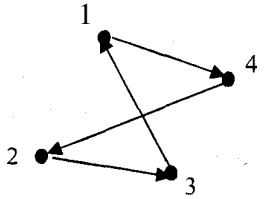
```
        if (a[i][j]==1)
                { k=k+1;
                u[k].x=i;
                u[k].y =j;}
m=k;
.....
```

8. Graf tare conex. Componente tare conexe

În această secțiune, vor fi prezentate noțiunile:
- graf tare conex
- componentă tare conexă
- algoritmul de descompunere a unui graf în componente tare conexe

Graf tare conex

Definiție. Fie G=(V, U) un graf orientat. **Graf**ul G se numeşte **tare conex**, dacă pentru oricare două vârfuri x şi y există un drum în G de la x la y şi un drum de la y la x.
• **Exemplu** de graf tare conex.

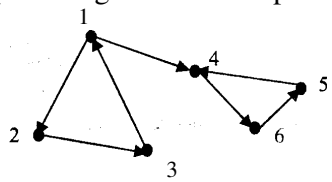


Graful este tare conex, deoarece, oricare ar fi vârfurile x şi y, există un drum in G de la x la y şi un drum de la y la x.

Componentă tare conexă

Definiție. Fie G=(V, U) un graf orientat. Se numeşte componentă tare conexă, un graf orientat $G_1 =(V_1, U_1)$ care verifică următoarele condiții:
- este **subgraf al grafului G**
- este **tare conex**;
- **oricare are fi x∈ V-V_1, subgraful lui G generat de V_1U{x} nu mai este tare conex** .

• **Exemplu.** Fie graful orientat prezentat în figura de mai jos:



Acest graf are două componente tare conexe:
- subgraful generat de nodurile 1, 2, 3;
- subgraful generat de nodurile 4, 5, 6.

Observație. Fie G=(V, U) un graf orientat: Graful G este tare conex, dacă admite o singură componentă tare conexă.

Algoritmul de descompunere a unui graf în componente tare conexe

Algoritmul procedează astfel:
- la început, nu este depistată nici o componentă tare conexă (nc=0);
- deci, nici un nod nu face parte din vreo componentă tare conexă (luate=[ ]);
- se parcurg nodurile grafului, cu i;
    - dacă i nu a fost introdus în nici o componentă tare conexă,

- se măreşte numărul componentelor tare conexe cu 1,
- se construieşte noua componentă tare conexă, astfel:
    - se intersectează predecesorii lui i cu succesorii săi, şi se reunesc cu {i}.

Pentru implementarea acestui algoritm, în limbajul C++, cu ajutorul programului prezentat mai jos, s-au folosit:

Funcţiile:

Succesori(i, S):care pune în şirul S toate nodurile j, din graf , cu proprietatea că există drum între i şi ele.

Predecesori(i, P):care pune în şirul P toate nodurile j, din graf, cu proprietatea că există drum între ele şi i.

Vectorul:

Comp :ale cărui componente, care sunt şiruri de elemente, vor reţine, la final, componentele tare conexe;

Variabilele:

d : matricea drumurilor;

luate : un şir care reţine toate nodurile care fac deja parte dintr-o componentă tare conexă;

nc : reprezintă numărul componentelor tare conexe depistate;

În program, au mai fost folosite şi alte variabile dar deoarece rolul lor reiese foarte uşor din urmărirea programului nu-l mai comentăm.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
typedef int mat[20][20];
typedef int sir[20];
mat d;
sir S, P, luate;
sir comp[50], ncomp;
int ok,nl, i, j, n, m, nc, ns, np;
void succesori(int i, sir S, int& ns)
{int j;
ns=0;
for (j=1;j<=n;j++)
        if (d[i][j]==1)
                        {ns=ns+1;
                        S[ns]=j;}
}
void predecesori(int i, sir P, int& np)
{ int j;
np=0;
for (j=1;j<=n;j++)
        if (d[j][i]==1)
                        {np=np+1;
                        P[np]=j;}
}
void intersectie(sir S, int ns, sir P, int np, sir x, int& nx)
{int ok;
int i, j;
nx=0;
for (i=1;i<=ns;i++ )
                {ok=0;
                for (j=1;j<=np;j++)
                        if (S[i]==P[j]) ok=1;
                if (ok==1)
                                {nx++;
                                x[nx]=S[i];}
                }
```

```
n=6
d[1,1]1
d[1,2]1
d[1,3]1
d[1,4]1
d[1,5]1
d[1,6]1
d[2,1]1
d[2,2]1
d[2,3]1
d[2,4]1
d[2,5]1
d[2,6]1
d[3,1]1
d[3,2]1
d[3,3]1
d[3,4]1
d[3,5]1
d[3,6]1
d[4,1]0
d[4,2]0
d[4,3]0
d[4,4]1
d[4,5]1
d[4,6]1
d[5,1]0
d[5,2]0
d[5,3]0
d[5,4]1
d[5,5]1
d[5,6]1
d[6,1]0
d[6,2]0
d[6,3]0
d[6,4]1
d[6,5]1
d[6,6]1
component tare conexa cu numarul 1
1 2 3
component tare conexa cu numarul 2
4 5 6
```

```
}
void main()
{clrscr();
cout<<"n="; cin>>n;
for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
                {cout<<"d["<<i<<","<<j<<"]";
                cin>>d[i][j];}
nc=0;
nl=0;
for (i=1;i<=n;i++)
            {ok=0;
            for (j=1;j<=nl;j++)
                if (luate[j]==i) ok=1;
            if(ok==0)
                        {nc++;
                        succesori(i,S,ns);
                        predecesori(i,P,np);
                        intersectie(S,ns,P,np,comp[nc],ncomp[nc]);
                        for (j=1;j<=ncomp[nc];j++)
                                {nl++;
                                luate[nl]=comp[nc] [j];}
                        }
            }
for (i=1;i<=nc;i++)
            {cout<<"component tare conexa cu numarul "<<i<<endl;
            for (j=1;j<=ncomp[i];j++)
            cout<<comp[i][j]<<" ";
            cout<<endl;}
getch();
}
```

9. Drumuri minime şi maxime

Noţiuni generale

În această secţiune vor fi prezentate, aşa cum sugerează şi titlul, modurile de tratare a problemelor care fac parte din următoarele două mari clase de probleme:
- probleme în care se cere determinarea drumurilor minime, dintr-un graf;
- probleme în care se cere determinarea drumurilor maxime, dintr-un graf.

Problemele de minim (maxim) se pot enunţa astfel:
1. Fiind dat graful G=(V,U), cu matricea costurilor asociată $C \in M_n(R)$, să se determine drumurile de lungime minimă (maximă) între oricare două vârfuri.
2. Fiind dat graful G=(V,U), cu matricea costurilor asociată $C \in M_n(R)$, să se determine drumurile de lungime minimă (maximă) între vârfurile i şi j.
3. Fiind dat graful G=(V,U), cu matricea costurilor asociată $C \in M_n(R)$, să se determine drumurile de lungime minimă (maximă) între vârful i şi toate celelalte vârfuri.

În cazul problemelor de minim, fiind dat graful G=(V, U) i se asociază matricea costurilor, **forma 1**, definită astfel:
$C \in M_n(R)$, unde:

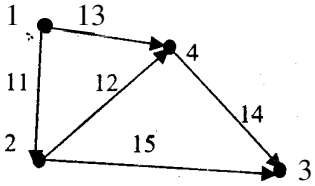
$$c_{i,j}=\begin{cases}\cos\ t,\ daca\ \text{int}re\ i\ si\ \ j\ exista\ un\ arc\ cu\ \cos tul\ \cos\ t\\ 0,\ daca\ i=j\\ \infty,\ daca\ i\neq j\ si\ \ (i,j)\notin U\end{cases}$$

În cazul problemelor de maxim, fiind dat graful G=(V, U) i se asociază matricea costurilor, **forma 2**, definită astfel:

$C\in M_n(R)$, unde:

$$c_{i,j}=\begin{cases}\cos\ t,\ daca\ \text{int}re\ i\ si\ \ j\ exista\ un\ arc\ cu\ \cos tul\ \cos\ t\\ 0,\ daca\ i=j\\ -\infty,\ daca\ i\neq j\ si\ \ (i,j)\notin U\end{cases}$$

• ***Exemplu:*** Fiind dat graful din figura de mai jos (costul fiecărui arc fiind scris pe ea)



matricea costurilor se scrie in felul următor:

Forma 1:

$$C=\begin{pmatrix}c_{11}&c_{12}&c_{13}&c_{14}\\c_{21}&c_{22}&c_{23}&c_{24}\\c_{31}&c_{32}&c_{33}&c_{34}\\c_{41}&c_{42}&c_{43}&c_{44}\end{pmatrix}=\begin{pmatrix}0&11&\infty&13\\\infty&0&15&12\\\infty&\infty&0&\infty\\\infty&\infty&14&0\end{pmatrix}$$

Forma 2:

$$C=\begin{pmatrix}c_{11}&c_{12}&c_{13}&c_{14}\\c_{21}&c_{22}&c_{23}&c_{24}\\c_{31}&c_{32}&c_{33}&c_{34}\\c_{41}&c_{42}&c_{43}&c_{44}\end{pmatrix}=\begin{pmatrix}0&11&-\infty&13\\-\infty&0&15&12\\-\infty&-\infty&0&-\infty\\-\infty&-\infty&14&0\end{pmatrix}$$

Observații.

1. Matricea costurilor forma 1 diferă de matricea costurilor forma 2 prin faptul că în loc de ∞ apare -∞.

2: În program nu se poate scrie ∞ sau -∞, de aceea recomandăm ca atunci când trebuie folosite să se definească două constante foarte mari, ca de exemplu, pentru ∞ : const p infinit = 1.e10;
pentru -∞ : const m infinit=-1.e10;

În continuare, vor fi prezentați doi algoritmi care permit rezolvarea unor probleme de minim (maxim), şi anume, vor fi prezentați:algoritmul Roy-Floyd şi algoritmul lui Dijkstra..

Algoritmul Roy-Floyd

Acest algoritm se aplică în cazul problemelor în care se dă un graf G=(V, U), care are matricea costurilor C, şi se cere să se determine **lungimea drumurilor minime**, şi în unele cazuri şi **nodurile care constituie drumurile respective**, între oricare două noduri ale grafului.

Observație. Algoritmul are la bază următoarea idee:

"Dacă drumul minim de la nodul i la nodul j trece prin nodul k, atunci şi drumul de nodul i la nodul k, precum şi de la nodul k la nodul j, este minim"

şi constă, de fapt, într-un şir de n transformări aplicate matricei costurilor C, astfel:

$T_k(C)=B$, $B\in M_n(R)$, $b_{i,j}=\text{minim}(c_{i,j},c_{i,k}+c_{k,j})$,$i,j\in\{1,..n\}$care se poate implementa astfel:

```
for k=1 ... n
        for i=1 ... n
                for j=l ... n
                        if (c[i,j]<c[i,k]+c[k,j]) c[i,j]=c[i,k]+c[k,j];
```

Descrierea detaliată a algoritmului prezentat mai sus:

Pentru fiecare pereche de noduri (i,j), unde i,j ∈ { 1,...,n}, se procedează astfel:

se parcurg cu k toate nodurile grafului, diferite de i şi j,

pentru fiecare nod k, se execută:

dacă costul drumului între nodurile i şi j este mai mic decât suma costurilor drumurilor între nodurile i şi k şi între nodurile k şi j, atunci costul drumului inițial de la i la j se va înlocui cu costul drumului i-k-j, evident, acest lucru făcându-se prin modificarea matricei costurilor.

Observație. Algoritmul prezentat în forma de mai sus, permite decât calcularea lungimii drumurilor minime între oricare două noduri ale grafului. Dacă se doreşte şi afişarea nodurilor care compun efectiv aceste drumuri, va trebui completat astfel:

Dacă lungimea drumului miinim dintre nodurile i şi j este egală cu suma dintre lungimile a 2 drumuri care trec printr-un nod intremediar k atunci nodul k face parte din drumul de lungime minimă de la i la j

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
int a[50][50],n,i,j,c,k,gasit=0,x,y;
int const p_inf=10000;
ifstream f("rf.in");
void init()       //se initializeaza matr costurilor
{f>>n;
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)if(i==j) a[i][j]=0;
                  else a[i][j]=p_inf;}
void citire()//se actualizeaza matr costurilor cu
datele din fisier
{while(f>>i>>j>>c) a[i][j]=c;
f.close();}
void transformare() //se transforma matricea
costurilor
{for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
        if(a[i][k]+a[k][j]<a[i][j])
a[i][j]=a[i][k]+a[k][j];}
void drum(int i, int j) //se det nodurile drumului
minim
```

```
4
1 2 11
1 4 13
2 4 12
2 3 15
4 3 14
```

```
{for(k=1;k<=n&&!gasit;k++)
if((i!=k&&j!=k)&&a[i][j]==a[i][k]+a[k][j])
                                {drum(i,k);
                                drum(k,j);
                                gasit=1;}
if(!gasit) cout<<j<<" ";}
void afisare(int x,int y)//afiseaza costului de drum
minim si nodurile care formeaza drumul
{if(a[x][y]<p_inf)
{cout<<"drumul minim de la nodul "<<x<<" la nodul
"<<y;
cout<<" are costul "<<a[x][y]<<endl;
cout<<x<<" ";
drum(x,y);}
else cout<<"nu exista drum";}
void main()
{cout<<"x="; cin>>x;
cout<<"y="; cin>>y;
init();
citire();
transformare();
afisare(x,y);
getch();}
```

```
x=1
y=3
drumul minim de la nodul 1 la nodul 3 are costul 26
1 2 3
x=4
y=2
nu exista drum
x=2
y=4
drumul minim de la nodul 2 la nodul 4 are costul 12
2 4
x=4
y=1
nu exista drumx=3
y=1
nu exista drumx=3
y=2
nu exista drum
```

În continuare, se prezintă programul, în C++, care implementează algoritmul comentat anterior

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
const p_inf=10000;
typedef int sir[20];
typedef int mat[20][20];
typedef sir matmul[20][20];
matmul d;
mat c, nd;
sir dr;
int i, k, j, n,m, ld,x,y,val;
void drum_de_la(int i,int j)
{ int k, gasitk,t;
if (i!= j){
```

```
n=4
m=5
x y val 1 2 11
x y val 1 4 13
x y val 2 4 12
x y val 2 3 15
x y val 4 3 14
0 11 10000 13
10000 0 15 12
10000 10000 0 10000
10000 10000 14 0
lung. drumului min de la 1 la 2 este 11
iar drumurile sunt :1 2
lung. drumului min de la 1 la 3 este 26
iar drumurile sunt :1 2 3
lung. drumului min de la 1 la 4 este 13
iar drumurile sunt :1 4
nu exista drum intre 2 si 1
lung. drumului min de la 2 la 3 este 15
iar drumurile sunt :2 3
lung. drumului min de la 2 la 4 este 12
iar drumurile sunt :2 4
nu exista drum intre 3 si 1
nu exista drum intre 3 si 2
nu exista drum intre 3 si 4
nu exista drum intre 4 si 1
nu exista drum intre 4 si 2
lung. drumului min de la 4 la 3 este 14
iar drumurile sunt :4 3
```

```
            for (k=1;k<=n;k++)
                        {gasitk=0;
                        for (t=1;t<=nd[i][j];t++)
                              if (d[i][j][t]==k) gasitk=1;
                              if (gasitk==1)
                                  { ld=ld+1;
                                  dr[ld]=k;
                                  drum_de_la(i,k);
                                  ld=ld-1;}
                        }
            }
else{ for (k=ld;k>=1;k--) cout<<dr[k]<<" ";
          cout<<endl;}
}
void afis()
{for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
                    if (c[i][j]==p_inf) cout<<"nu exista drum intre "<<i<<" si "<<j<<endl;
                    else if (i!=j)
                              { cout<<"lung. drumului min de la "<<i<<" la "<<j<<" este "<<c[i][j]<<endl;
                              cout<<"iar drumurile sunt :"<<endl;
                              ld=1;
                              dr[ld]=j;
                              drum_de_la(i, j);}
}
void reuneste(sir x,int nx, sir y, int ny, sir z, int& nz)
{ int i, j, ok;
nz=0;
for (i=1;i<=nx;i++)
                        {nz++;
                        z[nz]=x[i];}
for (j=1;j<=ny;j++)
                        {ok=0;
                        for (i=1;i<=nx;i++)
                              if (x[i]==y[i]) ok=1;
                        if (!ok)
                                  {nz++;
                                  z[nz]=y[j];}
                        }
}
void face_sirul(sir x, int nx, sir y, int& ny)
{int i;
ny=0;
for (i=1;i<=nx;i++)
                        {ny++;
                        y[ny]=x[i]; }
}
void main()
{ clrscr();
cout<<"n="; cin>>n;
for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
                    if (i==j) c[i][j]=0;
                              else c[i][j]=p_inf;
cout<<"m="; cin>>m;
```

```
for (i=1;i<=m;i++)
        {cout<<"x y val " ; cin>>x>>y>>val;
        c[x][y]=val;}
for (i=1;i<=n;i++)
        {for (j=1;j<=n;j++)
        cout<<c[i][j]<<" ";
        cout<<endl;}
for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
                if ((i!=j) && (c[i][j]<p_inf))
                                        { nd[i][j]=1;
                                        d[i][j][1]=i;}
                        else nd[i][j]=0;
for (k=1;k<=n;k++)
        for (i=1;i<=n;i++)
                for (j=1;j<=n;j++)
                        if (c[i][j]==c[i][k]+c[k][j])
                                reuneste(d[i][j],nd[i][j],d[k][j],nd[k][j],d[i][j],nd[i][j]);
                                else if (c[i][j]>c[i][k]+c[k][j])
                                    { c[i][j]=c[i][k]+c[k][j];
                                    face_sirul(d[k][j],nd[k][j],d[i][j],nd[i][j]);}
afis();
getche();}
```

Algoritmul lui Dijkstra

Acest algoritm se aplică în cazul problemelor în care se dă un graf G=(V, U), care are matricea costurilor C, şi se cere să se determine **lungimea drumurilor minime** de la un nod dat x până la fiecare nod din graf
Algortimul reține în mulțimea S nodurile care au fost deja selectate şi într-o coadă de priorități Q nodurile care nu au fost deja selectate adică Q=V-S, astfel:

- un nod y este selectat atunci când s-a determinat costul final al drumului cu costul minim de la nodul sursa x la el.
- - în coda Q prioritatea cea mai mare o are nodul pentru care costul drumului are valoarea cea mai mică dintre toate costurile de drumuri care pornesc de la nodul x la celelalte noduri neselectate încă. Pentru calcularea drumurilor de lungime minimă se întreține o mulțime D în care se memorează costul drumurilor de la nodul x la nodurile neselectate, costuri care se recalculează la fiecare extragere de nod.

Pas 1. Se inițializează mulțimea S cu mulțimea vidă, se citeşte nodul inițial x şi se atribuie mulțimii S
Pas 2. Se inițiază mulțimea D cu costurile drumurilor de la nodul x la toate celelalte noduri ale grafului
Pas 3. Cât timp coada de priorități Q nu este vidă execută:
    Pas 4. se caută printre nodurile selectate nodul y cu cel mai mic cost al drumului
    Pas 5. se adaugă nodul y la mulțimea S (se extrage din coada de priorități Q şi se declară ca nod selectat)
    Pas 6. Pentru fiecare nod neselectat execută:
        Pas 7. Se recalculează costul drumului de la nodul x la acest nod folosind ca nod intermediar nodul extras
        Pas 8. Dacă acest cost este mai mic decât cel din mulțimea D, atunci el va fi noul cost.

Se folosesc 3 vectori:

- Vectorul s pentru mulțimea nodurilor selectate
- Vectorul d conține costul drumurilor
- Vectorul t memorează drumurile găsite între nodul x şi celelalte noduri i ale grafului

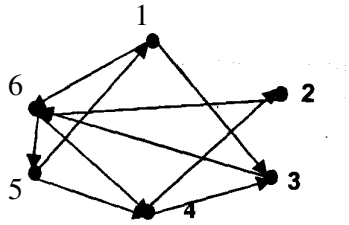
Algoritmul lui Dijkstra

Acest algoritm se aplică în cazul problemelor în care se dă un graf G=(V, U), care are matricea costurilor C, şi se cere să se determine **lungimea drumurilor minime** de la un nod dat x până la fiecare nod din graf
se selectează nodurile grafului unul câte unul în ordinea crescătoare a costului drumului de la nodul pl la ele, în mulțimea s, care inițial conține doar nodul pl.

Algoritmul foloseşte următoarele variabile:
n: reprezintă numărul de noduri ale grafului;
c: reprezintă matricea costurilor asociată grafului;
d - vect costului drumurilor
p-indică drumurile găsite între nodul pl şi celelalte noduri din graf (pt nodul i se reține nodul precedent pe unde trece drumul de la pl la i, pt pl se reține 0)
s- indică mulțimea nodurilor selectate(0 dacă i nu este selectat, 1 dacă i este selectat)

şi procedează astfel:
Pas1.- se citeşte nodul de plecare pl;
- se completează componentele vectorului d astfel
d[i]=c[pl,i], pentru i=1...n şi i≠pl,
d[pl]=0
- se completează componentele vectorului p astfel:
pentru i=1.::n, p[i]=p1, dacă i≠pl şi c[pl,i] ≠∞;
p[i]=0, altfel;
pas2. - se execută de n-1 ori următoarele:
- printre nodurile neselectate se caută cel aflat la distanța minimă față de pl şi se selectează, adăugându-l mulțimii s. Fie poz acest vârf
- pentru nodurile neselectate, j, se actualizează în d costul drumurilor de la pl la ele, utilizând ca nod intermediar nodul selectat, poz, procedând astfel:
-se compară costul existent în vectorul d, pt j, d(j), cu suma dintre costul existent în d pentru nodul selectat, poz, şi distanța de la nodul selectat, poz, la nodul pentru care se face actualizarea distanței, j:
d(poz)←a(poz,j) . În cazul în care suma este mai mică, elementul din d corespunzător nodului pentru care se face actualizarea, j, reține suma d(j) ←d(poz) +a(poz,j) şi elemetul din p corespunzător aceluaşi vârf, iar valoarea vârfului selectat p(j) ←poz (drumul trece prin acest vârf)
Pas 3. pentru fiecare vârf al grafului, cu excepția lui pl, se trasează drumul de la pl la el.

Exemplu de aplicare a algoritmului asupra grafului:



care are matricea costurilor:

$$C=\begin{pmatrix} 0 & \infty & 2 & \infty & \infty & 2 \\ \infty & 0 & \infty & \infty & \infty & 4 \\ \infty & \infty & 0 & \infty & \infty & 2 \\ \infty & 3 & 1 & 0 & \infty & \infty \\ 3 & \infty & \infty & 1 & 0 & \infty \\ \infty & \infty & \infty & 2 & 3 & 0 \end{pmatrix}$$

Rezolvare:
Pasul 1
Se stabileşte nodul de plecare: p1=1
Se completează vectorii d şi p, astfel:
- d[i]=c[pl,i], pentru i=1...n;
(în componentele vectorului d se trec elementele de pe prima linie din C)
- pentru i=1...n, p[i]=p1, dacă i≠pl şi c[pl,i] ≠∞;
p[i]=0, altfel;

	1	2	3	4	5	6
d	0	∞	2	∞	∞	2

p	0	0	1	0	0	1

Pasul 2

Dintre nodurile nealese încă, nealese={2,3,4,5,6}, se alege nodul j pentru care $d_j=\min\{d_2, d_3, d_4, d_5, d_6\}$; deci, se alege j=3, pentru ca $d_3=\min\{\infty, 2, \infty, \infty, 2\}$

	1	2	3	4	5	6
d	0		2			
p	0		1			

Se completează componentele vectorilor d şi p, pentru nodurile nealese, astfel:

2 : $\min(d_2, d_3+C_{3,2})=\min(\infty,2+\infty)=\infty$ (d_2 şi p_2 rămân nemodificate)

4 : $\min(d_4, d_3+C_{3,4})=\min(\infty,2+\infty)=\infty$ (d_4 şi p_4 rămân nemodificate)

5 : $\min(d_5, d_3+C_{3,5})=\min(\infty,2+\infty)=\infty$ (d_5 şi p_5 rămân nemodificate)

6 : $\min(d_6, d_3+C_{3,6})=\min(2,2+2)=2$ (d_5 şi p_5 rămân nemodificate)

	1	2	3	4	5	6
d	0	∞	2	∞	∞	2
p	0	0	1	0	0	1

Pasul 3

Dintre nodurile nealese încă, nealese={2,4,5,6}, se alege nodul j pentru care $d_j=\min\{d_2, d_4, d_5, d_6\}$; deci, se alege j=6, pentru că $d_6=\min\{\infty, \infty, \infty, 2\}$

	1	2	3	4	5	6
d	0		2			2
p	0		1			1

Se completează componentele vectorilor d şi p, pentru nodurile nealese, astfel:

2 : $\min(d_2, d_6+C_{6,2})=\min(\infty,2+\infty)=\infty$ (d_2 şi p_2 rămân nemodificate)

4 : $\min(d_4, d_6+C_{6,4})=\min(\infty,2+2)=4$ ($d_4=4$ şi $p_4=6$)

5 : $\min(d_5, d_6+C_{6,5})=\min(\infty,2+3)=5$ ($d_5=5$ şi $p_5=6$)

	1	2	3	4	5	6
d	0	∞	2	4	5	2
p	0	0	1	6	6	1

Pasul 4

Dintre nodurile nealese încă, nealese={2,4,5}, se alege nodul j pentru care $d_j=\min\{d_2, d_4, d_5\}$; deci, se alege j=4, pentru că dă $\min\{\infty, 4, 5\}$

	1	2	3	4	5	6
d	0		2	4		2
p	0		1	6		1

Se completează componentele vectorilor d şi p, pentru nodurile nealese, astfel:

2 : $\min(d_2, d_4+C_{4,2})=\min(\infty,4+3)=7$ ($d_2=7$ şi $p_2=4$)

5 : $\min(d_5, d_4+C_{4,5})=\min(5,4+\infty)=5$ (d_5 şi p_5 rămân nemodificate)

	1	2	3	4	5	6
d	0	7	2	4	5	2
p	0	4	1	6	6	1

Pasul 5

Dintre nodurile nealese încă, nealese={2,5}, se alege nodul j pentru care $d_j=\min\{d_2, d_5\}$; deci, se alege j=5, pentru ca $d_5=\min\{\infty, 5\}$.

	1	2	3	4	5	6
d	0		2	4	5	2
p	0		1	6	6	1

Se completează componentele vectorilor d şi p, pentru nodurile nealese, astfel:
2 : $\min(d_2, d5+C_{5,2})=\min(7,S+\infty)=7$ (d_2 şi p_2 rămân nemodificate)

	1	2	3	4	5	6
d	0	7	2	4	5	2
p	0	4	1	6	6	1

Concluzii:
Drumurile minime de la nodul 1 la celelalte noduri sunt:
2 : $p_2=4$, $p_4=6$, $p_6=1$; deci, de la 1 la 2 avem : 1, 6, 4, 2
3 : $p_3=1$; deci, de la 1 la 3 avem : 1, 3
4 : $p_4=6$, $p_6=1$; deci, de la 1 la 4 avem : 1, 6,4
5 : $p_5=6$, $p_6=1$; deci, de la 1 la 5 avem : 1, 6, 5
6.1 $p_6=1$; deci, de la 1 la 6 avem : 1, 6

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
ifstream f("d.in");
const p_inf=10000;
float a[50][50],d[50],min;
int n,i,j,pl,poz,p[50],s[50],c;
void drum (int i)
{if(p[i]!=0) drum(p[i]);
cout<<i<<" ";
}
void main()
{f>>n;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j) a[i][j]=0;
else a[i][j]=p_inf;
while(f>>i>>j>>c)
a[i][j]=c;
f.close();
cout<<"pl="; cin>>pl;
for(i=1;i<=n;i++)
{d[i]=a[pl][i];
if(i!=pl&&d[i]<p_inf) p[i]=pl;}
for(i=1;i<=n-1;i++)              {min=p_inf;
for(j=1;j<=n;j++)
if(s[j]==0)
if(d[j]<min)
{min=d[j];
poz=j;}
s[poz]=1;
for(j=1;j<=n;j++)
if(s[j]==0 && d[j]>d[poz]+a[poz][j])
```

```
6
1 6 2
1 3 2
2 6 4
3 6 2
4 2 3
```

```
{d[j]=d[poz]+a[poz][j];
p[j]=poz;}
}
for(i=1;i<=n;i++)
cout<<d[i]<<" ";
cout<<endl;
for(i=1;i<=n;i++)
if(i!=pl)
if(p[i]!=0)
{cout<<"drumul de cost minim de la nodul"<<pl<<" la nodul "<<i<<" are costul"<<d[i]<<endl;
cout<<endl;}
else
cout<<"Nu exista drum de la "<<pl<<" la "<<i<<endl;
getch();}


//algoritmul dijkstra
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
int a[50][50],n,i,j,c,d[100],s[100],p[100],x,y,min;
int const p_inf=10000;
ifstream f("rf.in");
void init()//se initiealizeaza matr costurilor
{f>>n;
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)if(i==j) a[i][j]=0;
                          else a[i][j]=p_inf;}
void citire()//se actualizeaza matr costurilor cu datele din fisier
{while(f>>i>>j>>c) a[i][j]=c;
f.close();}
void generare_drum(int x) //se genereaza drumurile
{
s[x]=1;
for(i=1;i<=n;i++)
{d[i]=a[x][i];
if(i!=x &&d[i]<p_inf) p[i]=x;}
min=p_inf;
for(i=1;i<=n;i++)
{for(j=1;j<=n;j++)
if(s[j]==0&&d[j]<min)
{min=d[j];
y=j;}
s[y]=1;
for(j=1;j<=n;j++)
if(s[i]==0&&d[j]>d[y]+a[y][j])
{d[j]=d[y]+a[y][j];
p[j]=y;}
}}
void drum(int i)
{if (p[i]!=0) drum (p[i]);
cout<<i<<" ";}
void afisare(int x)
{for(i=1;i<=n;i++)
if(i!=x)
```

```
4
1 2 11
1 4 13
2 4 12
2 3 15
```

```
x=1
drumul cu costul minim de la nodul 1 la nodul 2 are costul 11
1 2
drumul cu costul minim de la nodul 1 la nodul 3 are costul 26
1 2 3
drumul cu costul minim de la nodul 1 la nodul 4 are costul 13
1 4
```

```
if(p[i]!=0)
{cout<<"drumul cu costul minim de la nodul "<<x;
cout<<" la nodul "<<i<<" are costul "<<d[i]<<endl;
drum(i);
cout<<endl;}
else cout<<" nu exista drum de la "<<x<<" la "<<i<<endl;}
void main()
{cout<<"x="; cin>>x;
init();
citire();
generare_drum(x);
afisare(x);
getch();}
```