Drumuri Minime şi Maxime în grafuri Orientate

A. Aspecte teoretice

Definiţie. Se numeşte **graf orientat** o pereche **ordonată de mulţimi** notate **G = (V, U)**, unde :

V : este o mulţime ,**finintă** şi **nevidă,** ale cărei elemente se numesc **vârfuri** sau **noduri** ;

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) }

Graful **G** este un **graf orientat** deoarece respectă definiţia prezentată mai sus, adică :

V este **finită** şi **nevidă ;**

U este o mulţime de **perechi ordonate** de elemente din **V**.

Observaţie !

Într-un graf orientat **arcul (x, y)** este diferit de **arcul (y, x)** .

Terminologie fregventă în teoria grafurilor:

- **extremităţile unui arc**
  - fiind dat un arc **u = (x, y)**, se numesc extremităţi ale sale **nodurile x** şi **y:**
    - **x** se numeşte **extremitate initială;**
    - **y** se numeşte **extremitate finală.**
- **vârfuri adiacente**
  - dacă intr-un graf există arcul **u = (x, y)** (sau **(y ,x)**, sau amândoua), se spune despre **nodurile x** şi **y** ca sunt **adiacente.**
- **incidenta**
  - dacă **u1** şi **u2** sunt două arce ale aceluiaş 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.

B. Metode de reprezentare a grafului orientat

Sunt admise două metode de reprezentare :

- **reprezentare textuală :** asa cum au fost prezentate până acum
- **reprezentare grafica : arcele** sunt reprezentate prin **săgeţi orientate** iar **nodurile** prin **puncte.**

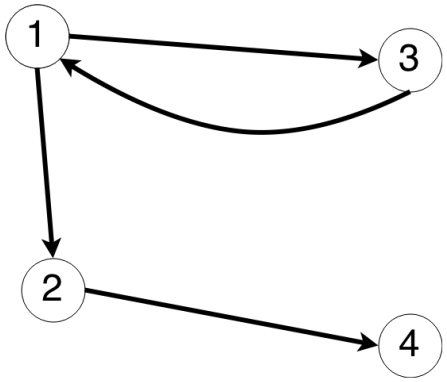
Exemplu de graf orientat reprezentat **textual:**

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)**:**



Observatie:

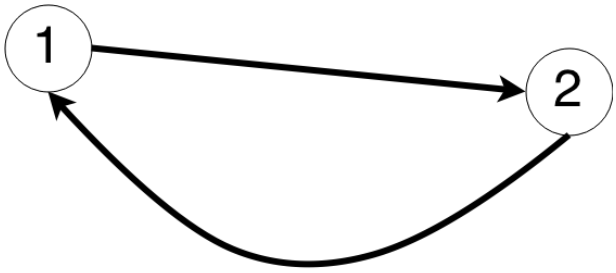
Considerând graful :

G = (V, U) unde :

V = { 1, 2 }

U = { (1, 2); (2, 1) }

reprezentarea sa grafică poate fi :



În continuarea lucrării se va folosi următoarea notație pentru a reprezenta nodurile care au arce de la unul la altul în ambele direcţii, adică în felul următor :

C. Noţiunea de graf parţial

Definiţie. Fie **G=(V, U)** un **graf orientat**.Se numeşte **graf parţial** al grafului **G graful orientat G1 = (V, U1)** unde **U1 ⊆ U.**

Citind cu atenţie definiţia tragem concluzia ca un **graf parţial** al unui graf **G** este un graf care are aceeaşi mulţime de vârfuri ca şi **G** , iar mulţimea arcelor este o submulţime a lui **U** sau chiar **U**.

Exemplu :

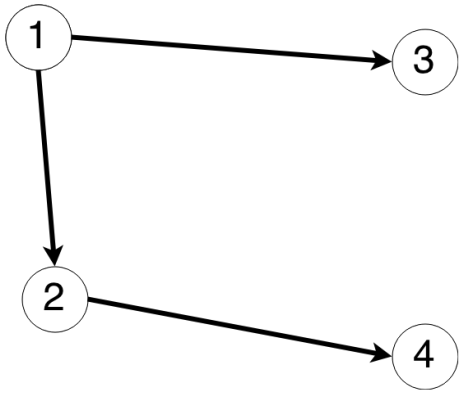
Fie graful orientat :

G = (V, U)

V = {1, 2, 3, 4}

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

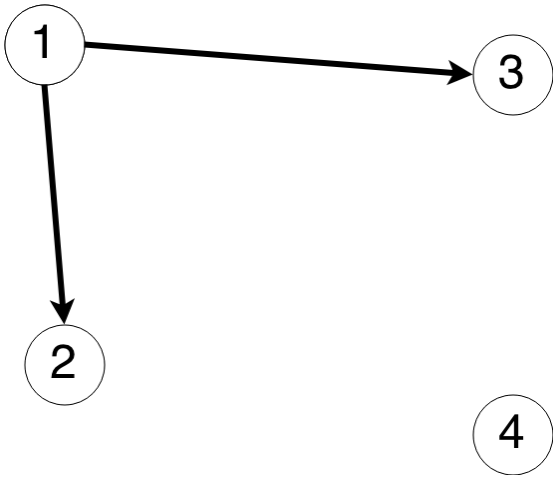
reprezentat **grafic** astfel :



1. Un exemplu de **grafic parţial al lui G** este graful orientat :

G = (V, U)
V = {1, 2, 3, 4}
U = {(1, 2); (1, 3)} (s-a eliminat arcul (2, 4))

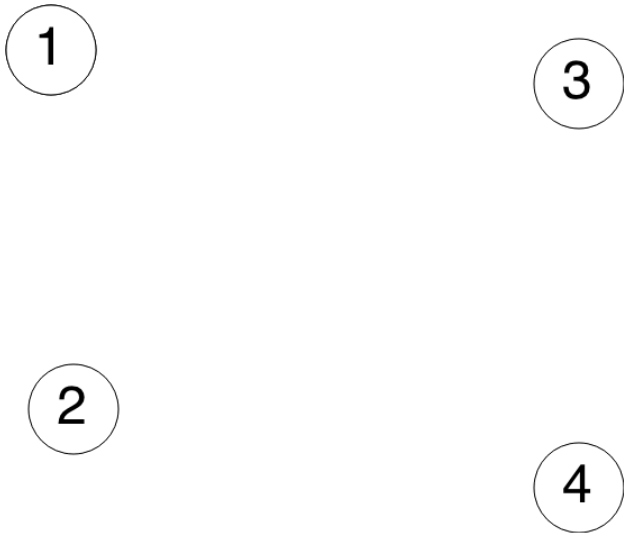
reprezentat **grafic** astfel :



2. Un exemplu de **grafic parţial al lui G** este graful orientat :

G = (V, U)
V = {1, 2, 3, 4}
U = ⊘ (s-au eliminat toate arcele)

reprezentat **grafic** astfel :



Observatie !
Fie **G = (V, U)** un **graf orientat**.**Un graf parţian** , al grafului **G**, **se poate obţine pastrând vârfurile şi eliminând eventual niste arce** (se pot elimina și **toate arcele** sau chiar **nici unu**).

D. Noţiunea de subgraf

Definiţie. Fie **G=(V, U)** un **graf orientat**.Se numeşte **subgraf** al grafului **G graful orientat G1 = (V1, U1)** unde **V1** $\subseteq$ **V** , iar **U1** conţine toate arcele din **U** care au extremităţile în **V1**.

Exemplu :

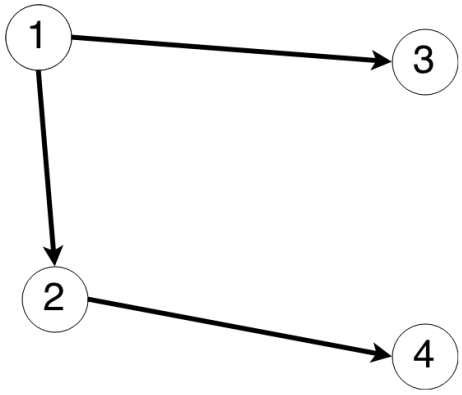
Fie graful orientat :

G = (V, U)

V = {1, 2, 3, 4}

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

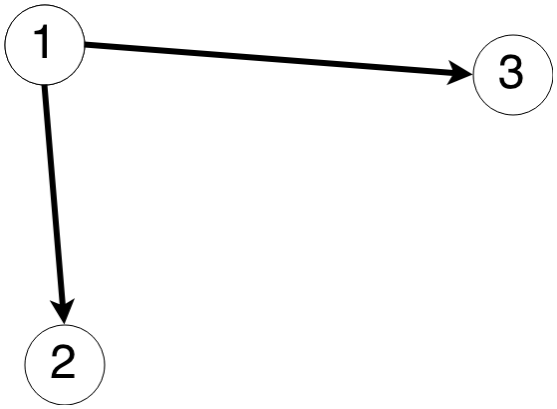
reprezentat **grafic** astfel :



3. Un exemplu de **subgraf al lui G** este graful orientat :

G = (V, U)
V = {1, 2, 3 } (s-a eliminat nodul 4)
U = {(1, 2); (2, 3)} (s-a eliminat arcul (2, 4))

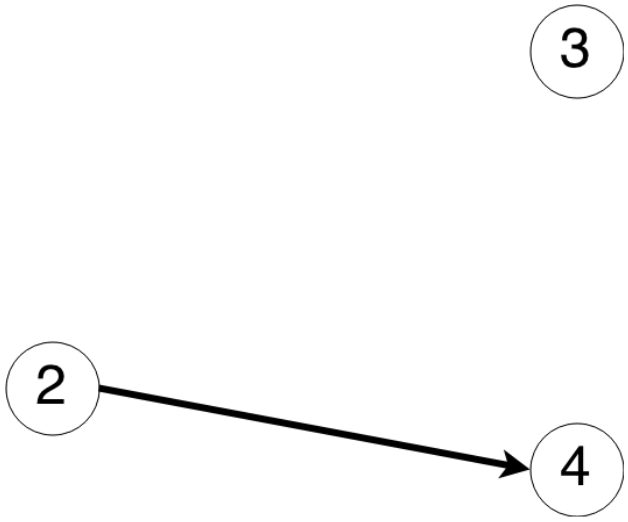
reprezentat **grafic** astfel :



4. Un exemplu de **graf parţial al lui G** este graful orientat :

G = (V, U)
V = {2, 3, 4} (s-a eliminat nodul 1)
U = {(2, 4) } (s-au eliminat arcele (1,3) şi (1,2))

reprezentat **grafic** astfel :



Observaţie !
Fie **G = (V, U)** un **graf orientat**.**Un subgraf** , al grafului **G**, **se poate obţine ştergând eventual anumite noduri 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 varfurilor vida)

E. Gradul unui nod

Ţinând cont de faptul că "**raportat la un vârf există arce care ies din acel vârf şi arce care intră în acel vârf** ", iau naştere următoarele noţiuni :

- **grad interior**
- **grad exterior**

Definiţie. Fie **G=(V, U)** un graf orientat şi **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 $d^+(x)$.

Exemplu :

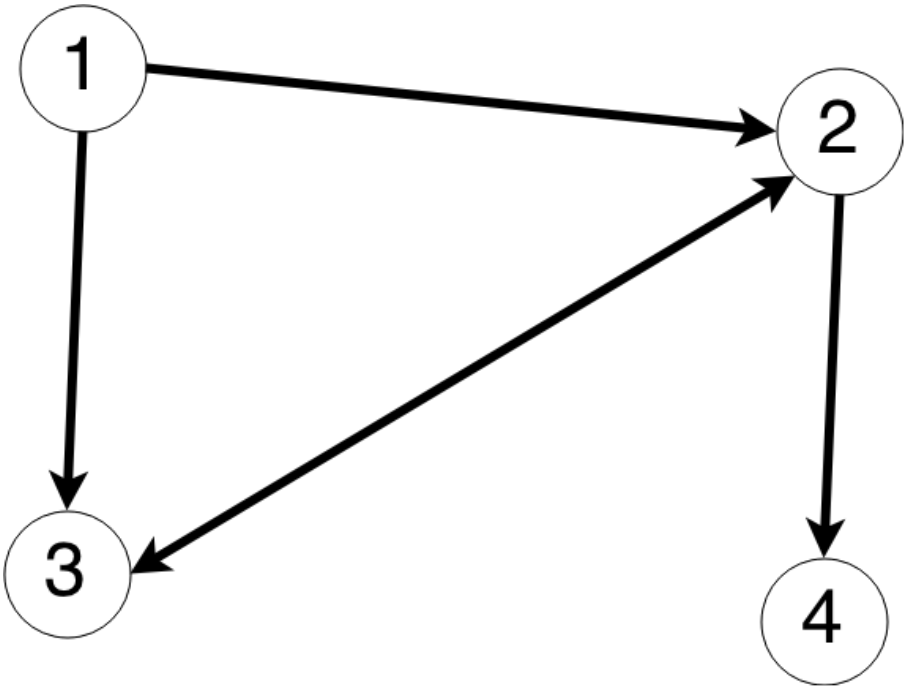
Fie grafiul :

G = (V, U) unde:

V = {1, 2, 3, 4}

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

Reprezentat astfel :



Gradul exterior al **nodului 1** este $d^+(1) = 2$ **(** în graf, sunt **trei** arce care **ies** din nodu 1)
Gradul exterior al **nodului 2** este $d^+(2) = 1$ **(** în graf, sunt **doua** arce care **ies** din nodu 2)
Gradul exterior al **nodului 3** este $d^+(3) = 0$ **(** în graf, sunt **zero** arce care **ies** din nodu 3)
Gradul exterior al **nodului 4** este $d^+(4) = 0$ **(** în graf, sunt **zero** arce care **ies** din nodu 4)

Observatii

1) **Mulţimea succesorilor nodului x** se notează cu $\Gamma^+(x)$ şi se reprezintă astfel :
$$\Gamma^+(x) = \{ y \in V \mid (x, y) \in U\}$$

2) **Mulţimea arcelor ce ies din nodul x** se notează cu $\omega^+(x)$ şi se reprezintă astfel :
$$\omega^+(x) = \{ (x, y) \in U \mid y \in V \}$$

3) **Legat de cardinalele mulţimilor** $\Gamma^+(x)$ **şi** $\omega^+(x)$ putem scrie :
$$| \Gamma^+(x) | = | \omega^+(x) | = d^+(x)$$

Raportat la acelaşi graf putem scrie următoarele:

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

$\Gamma^+(2) = \{4\}$	$\omega^+(2) = \{(2,4)\}$	$\lvert\Gamma^+(2)\rvert = \lvert\omega^+(2)\rvert = d^+(2) = 1$
$\Gamma^+(3) = \phi$	$\omega^+(3) = \phi$	$\lvert\Gamma^+(3)\rvert = \lvert\omega^+(3)\rvert = d^+(3) = 0$
$\Gamma^+(4) = \phi$	$\omega^+(4) = \phi$	$\lvert\Gamma^+(4)\rvert = \lvert\omega^+(4)\rvert = d^+(4) = 0$

Definiţie. 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 intra din x), notat $d^-(x)$.

Exemplu :

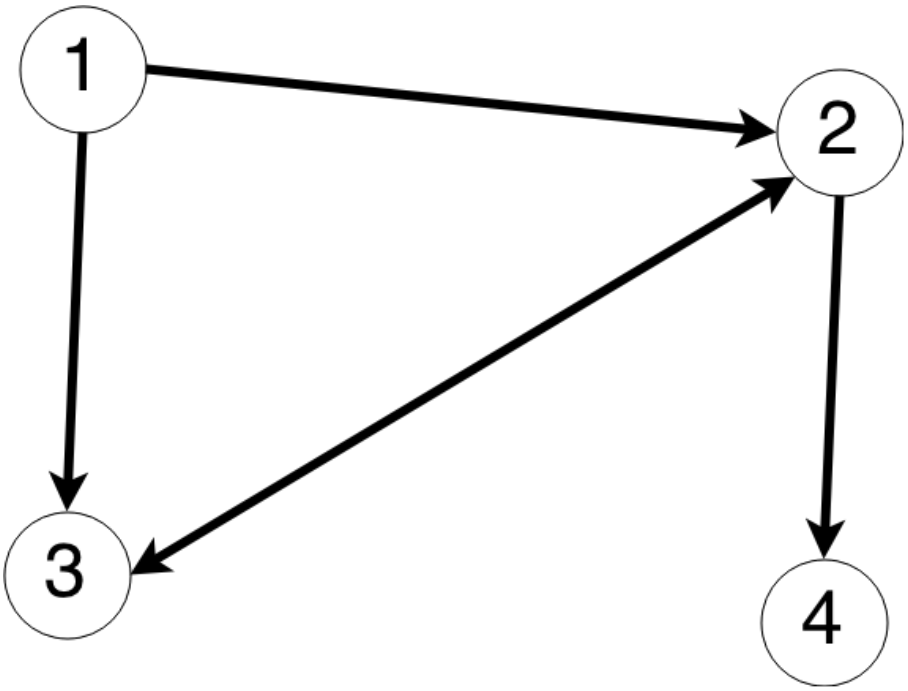
Fie graful :

G = (V, U) unde:

V = {1, 2, 3, 4}

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

Reprezentat astfel :



Gradul exterior al **nodului 1** este $d^-(1) = 0$ **(**în graf, sunt **zero** arce care **intra** din nodu 1)

Gradul exterior al **nodului 2** este $d^-(2) = 2$ **(**în graf, sunt **doua** arce care **intra** din nodu2)

Gradul exterior al **nodului 3** este $d^-(3) = 2$ **(**în graf, sunt **doua** arce care **intra** din nodu 3)
Gradul exterior al **nodului 4** este $d^-(4) = 1$ **(**în graf, este **un** arc care **intra** din nodu 4)

Observatii.

2) **Mulţimea predecesorilor nodului x** se notează cu $\Gamma^-(x)$ şi se reprezintă astfel :
$\Gamma^-(x) = \{ y \in V \mid (y, x) \in U\}$

2) **Mulţimea arcelor ce intra din nodul x** se notează cu $\omega^-(x)$ şi se reprezintă astfel :
$\omega^-(x) = \{ (y, x) \in U \mid y \in V \}$

3) **Legat de cardinalele mulţimilor** $\Gamma^-(x)$ **şi** $\omega^-(x)$ putem scrie :
$|\Gamma^-(x)| = |\omega^-(x)| = d^-(x)$

Raportat la acelaşi graf putem scrie următoarele:

$\Gamma^-(1) = \phi$	$\omega^-(1) = \phi$	$\lvert\Gamma^-(1)\rvert = \lvert\omega^-(1)\rvert = d^-(1) = 0$
$\Gamma^-(2) = \{1, 3\}$	$\omega^-(2) = \{ (1,2); (3, 2) \}$	$\lvert\Gamma^+(2)\rvert = \lvert\omega^-(2)\rvert = d^-(2) = 2$
$\Gamma^-(3) = \{1, 2\}$	$\omega^-(3) = \{ (1, 3); (2, 3) \}$	$\lvert\Gamma^-(3)\rvert = \lvert\omega^-(3)\rvert = d^-(3) = 2$
$\Gamma^-(4) = \{2\}$	$\omega^-(4) = \{2, 4\}$	$\lvert\Gamma^-(4)\rvert = \lvert\omega^-(4)\rvert = d^-(4) = 1$

F. Conexitate

În această secțiune vor fi prezentate următoarele noţiuni :

- Drum
- Drum elementar
- Drum neelementar

Drum

Definiţie.

Fie **G=(V, U)** un **graf orientat**. Se numeşte **drum** , în graful **G**, o succesiune de noduri notată $D = (x_1, x_2, x_3, \ldots, x_k)$ cu proprietatea $(x_1, x_2), (x_2, x_3), \ldots (x_{k-1}, x_k) \in U$ (altfel spus $(x_1, x_2), (x_2, x_3), \ldots (x_{k-1}, x_k)$ sunt arce).

Se întalnesc noţiunile:

- **extremităţile drumului**
  - fiind dat drumul $D = (x_1, x_2, x_3, \ldots, x_k)$ se numesc

extremităţi ale sale nodurile x_1 **şi** x_k (x_1 **- extremitate iniţială**; x_k **- extremitate finală**);

- **lungimea drumului**
  - fiind dat drumul $D = (x_1, x_2, x_3, \ldots, x_k)$, prin **lungimea sa** se înţelege **numarul de arce care apar în cadrul său**;

Exemplu:

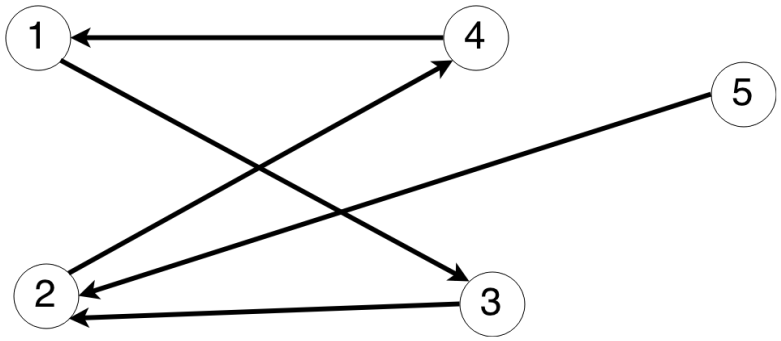
Fie graful :

G = (V, U) unde:

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

U = {(1, 3); (4, 1); (3, 2); (2, 4); (5, 2)}

Reprezentat 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 $D = (x_1, x_2, x_3, \ldots, x_k)$ este drum în graful **G**, atunci nu neapărat şi $D1 = (x_k, x_{k-1}, x_{k-2}, \ldots, x_1)$ este drum în graful **G.**

<u>Drum elementar</u>

Definiţie.

Fie **G=(V, U)** un **graf orientat**. Se numeşte **drum elementar** , în graful **G**, drumul $D = (x_1, x_2, x_3, \ldots, x_k)$ cu proprietatea că **oricare două noduri ale sale sunt distincte** (altfel spus, **printr-un nod nu se trece decât o singură dată**).

Exemplu:

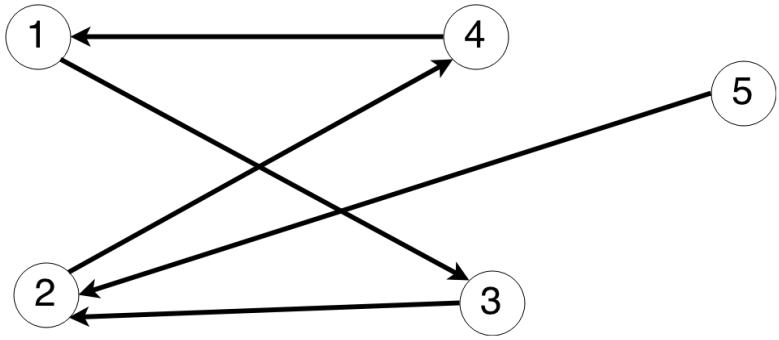
Fie graful :

G = (V, U) unde:

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

U = {(1, 3); (4, 1); (3, 2); (2, 4); (5, 2)}

Reprezentat astfel :



drumul **D = (4, 1, 3, 2)** este, în graful **G, drum elementar** cu lungimea **3** şi extremităţile **4** şi **2 .**

<u>Drum neelementar</u>

Definiţie.

Fie **G=(V, U)** un **graf orientat**. Se numeşte **drum elementar** , în graful **G**, drumul $D = (x_1, x_2, x_3, \ldots, x_k)$ cu proprietatea că **nodurile sale nu sunt distincte două câte două**(altfel spus, **printr-un nod s-a trecut de mai multe ori**).

Exemplu:

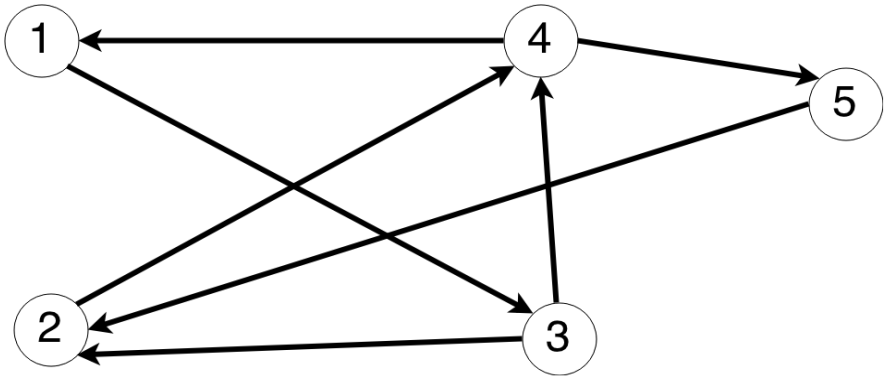
Fie graful :

G = (V, U) unde:

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

U = {(1, 3); (4, 1); (3, 2); (2, 4); (3, 4); (4, 1); (4, 5); (5, 2) }

Reprezentat astfel :



drumul **D2 = (4, 1, 3, 2, 4, 5, 2)** este, în graful **G, drum neelementar** cu lungimea **6** şi extremităţile **4** şi **2 .**

G. Metode de stocare digitală a grafurilor orientate

Pentru a putea prelucra un graf orientat cu ajutorul unui program, trebuie mai întai sa fie reprezentat în program.
Există mai multe modalităţi de a reprezenta un graf orientat într-un program:

- reprezentare prin **matrice de adiacenţă;**
- reprezentare prin matrice **vârfuri-arce;**
- reprezentare prin **matrice a drumurilor;**
- reprezentare prin **lista de adiacenţă;**
- reprezentare prin **şirul arcelor ;**

În continuare vom prezenta cum se poate reprezenta un graf orientat prin **matricea de adiacenţă.**
Pentru a reprezenta un graf în memoria unui program o sa ne folosim următoarele proprietăi ale unui graf orientat:

- **arcul (x, y) este diferit de arcul (y, x)**
- $\forall (x, y) \in U \mid x, y \in V$

Fie **G = (V, U)** un graf orientat cu n vârfuri (**V = {1, 2, 3 … n }**) şi **m** arce .

Definiţie

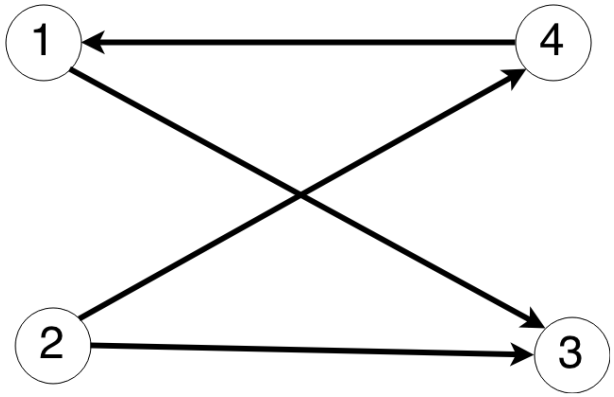
Matricea de adiacenţă ($A \in M_n(\{0, 1\})$), asociată grafului **G**,este **o matrice pătratică de ordin n**, cu elementele:

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

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

Exemplu 1 :

Fie graful **G** reprezentat ca în figura de mai jos :

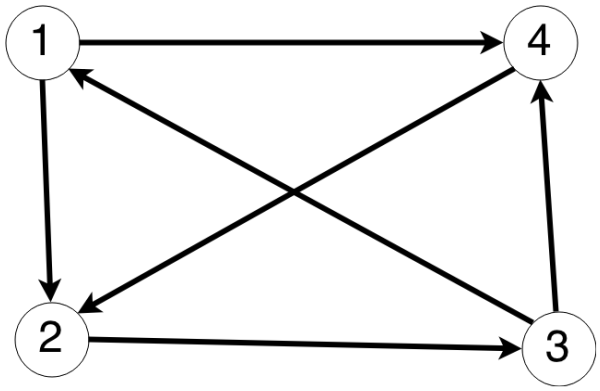


Matricea de adiacenţă asociată grafului este :

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Exemplu 2 :

Fie graful **G** reprezentat ca în figura de mai jos :



Matricea de adiacenţă asociată grafului este :

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Comentarii :

Matricea de adiacenţă este o matrice **pătratică**, de ordin n, şi **nu este neapărat simetrică** faţa de diagonala principală, aşa cum este cazul grafurilor neorientate.

Secvenţe de citire a matricei de adiacenţă:

```
program prrogram_1;
var a :array[1..100, 1..100] of 0..1;
```

```
   i, j, n :byte;

begin
write('n=');
readln(n); { numarul de noduri }

{ initlializam matriea cu 0 }
fillchar(a, sizeof(a), 0);

for i := 1 to n do
   for j := 1  to n do
      begin
      { citim valoarea (0 sau 1) a fiecarui element din matricea de
adiacenta }
      write('a[', i, ', ', j, ']=');
      readln(a[i, j]);
      end;
……
```

sau

```
program prrogram_2;
var a :array[1..100, 1..100] of 0..1;
   i, j, n, m, x, y :byte;

begin
write('n=');
readln(n); { numarul de noduri }

write('m=');
readln(m); { numarul de muchii }

{ initlializam matriea cu 0 }
fillchar(a, sizeof(a), 0);

for i := 1 to m do
   begin
   write('x=');
   readln(x); { extremitatea initiala a arcului (x, y) }

   write('y=');
   readln(y); { extremitatea finala a arcului (x, y) }
```

```
    a[x, y] := 1;
    end;
……..
```

Comentarii :

- **Matricea de adiacenţă** are toate elementele de pe diagonala principală egale cu **0** .
- Numărul elementelor egale cu 1 de pe linia **i este egal cu gradul exterior al vârfului i.**
    - ```
      function gr_ext(i :byte):byte;
      { subprogramu returneaza gradul exterior al nodului <i> }
      var j, s :byte;
      begin
      s := 0;

      for j := 1 to n do
         s := s + a[i, j];

      gr_ext := s;
      end;
      ```
- Numărul elementelor egale cu 1 de pe coloana **i este egal cu gradul interior al vârfului i.**
    - ```
      function gr_int(i :byte):byte;
      { subprogramu returneaza gradul interior al nodului <i> }
      var j, s :byte;
      begin
      s := 0;

      for j := 1 to n do
         s := s + a[j, i];

      gr_int:= s;
      end;
      ```
- Dacă vârful **i** este un **vârf izolat , pe linia i şi pe coloana i nu sunt elemente egale cu 1.**
    - ```
      function vf_izolat(i :byte):boolean;
      { subprogramu returneaza TRUE dacă nodul <i> este izolat şi
        FALSE în caz contrar }
      begin
      vf_izolat := (gr_ext(i) = 0) AND (gr_int(i) = 0);
      end;
      ```
      ```

G. Metode de parcurgere a grafurilor orientate

Prin parcurgerea sau traversarea unui graf orientat se urmăreste examinarea nodurilor sale, plecând dintr-un nod dat **x** , astfel încât fiecare nod, la care se poate ajunge din x pe muchii adiacente două câte două, sa fie marcat o singură dată.

Există două metode principale de parcurgere a grafurilor :

- **în lăţime**
- **în adâncime**

Metoda parcurgeri în lăţime

Această metodă foloseşte o structură de tip coadă. Se porneşte de la nodul **x** se găsesc toate nodurile **y** $\in V$ **a. î.** $\exists$ **(x, y)** $\in$ **U,** apoi se ia fiecare nod y găsit şi se

repetă procesul pentru acesta .

```
program program_3;
var a :array[1..100, 1..100] of 0..1; { matricea de adiacenta }
   c :array[0..100] of byte; { coada }
   viz :array[1..100] of boolean; { vectorul cu noduri vizitate }
   i, n, m, x, p :byte;
   i_c, n_c :integer;


procedure citire_matrice_adiacenta;
{ citim  matricea de adiacenta a grafului }
var i, x, y :integer;
begin
write('n=');
readln(n); { numarul de noduri }

write('m=');
readln(m); { numarul de muchii }

{ initlializam matriea cu 0 }
fillchar(a, sizeof(a), 0);
```

```
writeln('In continuare introduceti extremitatile arcelor :');
for i := 1 to m do
   begin
   write('x=');
   readln(x); { extremitatea initiala a arcului (x, y) }

   write('y=');
   readln(y); { extremitatea finala a arcului (x, y) }

   a[x, y] := 1;
   writeln;
   end;
end;


procedure init_coada;
{ initializam coada }
begin
n_c := -1;
i_c := 0;
end;


procedure push_coada(x :byte);
{ adaugam elementul <x> în coada }
begin
inc(n_c);
c[n_c] := x;
end;


function este_vida_coada:boolean;
{ returnam TRUE dacă coada este vida, şi FALSE în caz contrar }
begin
este_vida_coada := (n_c < i_c );
end;


function pop_coada:byte;
{ returnam primul element din coada şi il eliminam
  acest subprogram se utilizeaza doar cand coada nu
  este vida }
begin
```

```
pop_coada := c[i_c]; { returnam elementul }
inc(i_c); { eliminam elementul }
end;


procedure init_viz;
{ initializam vectorul cu nodurile vizitate cu FALSE, nici
  un nod nu a fost viziat }
begin
fillchar(viz, sizeof(viz), FALSE);
end;


procedure vizitat(i :byte);
{ marcam nodul <i> ca fiind vizitat }
begin
viz[i] := TRUE;
end;


function a_fost_vizitat(i :byte):boolean;
{ returneaza TRUE dacă nodul a fost viziat, în caz contrar
  returneaza FALSE }
begin
a_fost_vizitat := viz[i];
end;


procedure prelucrare_nod(x :byte);
{ prelugram nodul, în cazu de fata o sa il afisam doar }
begin
writeln('Prelugram nodul :', x);
end;


begin
init_coada; { initializam coada }

citire_matrice_adiacenta; { citim toate elementele grafului }

write('De la ce nod doriti sa pornim : ');
readln(x); { citim nodul de pornire }
```

```
push_coada(x); { adaugam <x> în coada }

while  not este_vida_coada() do { cat timp mai avem noduri de prelucrat în coada }
   begin

   p := pop_coada; { scoatem un element din coada }

   prelucrare_nod(p); { prelucram nodul }

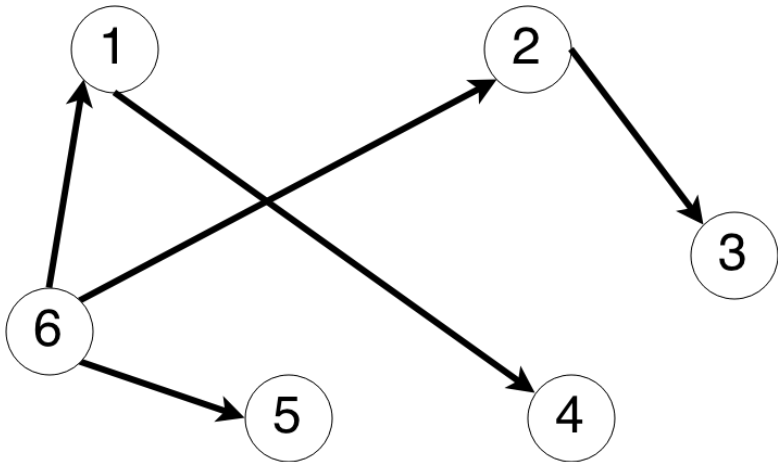
   vizitat(p); { noram nodul ca fiind vizitat }

   { cautam toate nodurile la care putem ajunge prin nodul x }
   for i := 1 to n do
      if (a[p, i] = 1) AND not a_fost_vizitat(i) then { dacă exista arc de la <p> la
                                                        <i> şi dacă nu am vizitat nodul <i> }
         push_coada(i); { adauga nodul în coada }

   end;

writeln;
end.
```

Dacă avem graful reprezentat grafic astfel :



Pornind de la nodul **6** vom descoperi nodurile în ordinea următoare :

6, 1, 2, 5, 4, 1

Pornind de la nodul **1** vom descoperi nodurile în ordinea următore :

1, 4

Observatie :

Dacă există noduri izolate sau dacă nodul are mai multe componente conexe, metodele de de parcurgere nu vor descoperi toate nodurile, doar pe cele la care poate ajunge.

Metoda parcurgeri în adâncime

Această metodă foloseşte o structură de tip stiva. Se porneşte de la nodul **x** se găseşte primul nod **y** $\in V$ **a. î.** $\exists$ **(x, y)** $\in$ **U,** iar pentru acest **y** se repetă procesul, când ajungem la un nod care nu are vecini nevizitaţi, coborâm în stivă şi cautăm un alt vecin nevizitat, până stiva devine vidă .

```
program program_4;
var a :array[1..100, 1..100] of 0..1; { matricea de adiacenta }
   s :array[0..100] of byte; { stiva }
   viz :array[1..100] of boolean; { vectorul cu noduri vizitate }
   i, n, m, x, p :byte;
   i_s, n_s :integer;
   ok :boolean;


procedure citire_matrice_adiacenta;
{ citim  matricea de adiacenta a grafului }
var i, x, y :integer;
begin
write('n=');
readln(n); { numarul de noduri }

write('m=');
readln(m); { numarul de muchii }

{ initlializam matriea cu 0 }
fillchar(a, sizeof(a), 0);

writeln('In continuare introduceti extremitatile arcelor :');
for i := 1 to m do
   begin
   write('x=');
   readln(x); { extremitatea initiala a arcului (x, y) }

   write('y=');
   readln(y); { extremitatea finala a arcului (x, y) }

   a[x, y] := 1;
   writeln;
   end;
```

```
end;


procedure init_stiva;
{ initializam stiva }
begin
n_s := 0;
i_s := 0;
end;


procedure push_stiva(x :byte);
{ adaugam elementul <x> în stiva }
begin
inc(n_s);
s[n_s] := x;
end;


function este_vida_stiva:boolean;
{ returnam TRUE dacă stiva este vida, şi FALSE în caz contrar }
begin
este_vida_stiva := (n_s = i_s );
end;


procedure pop_stiva;
{ eliminam un element din stiva }
begin
dec(n_s); { eliminam elementul }
end;

function acces_stiva:byte;
{ accesam elementul din varful stivei }
begin
acces_stiva := s[n_s];
end;


procedure init_viz;
{ initializam vectorul cu nodurile vizitate cu FALSE, nici
  un nod nu a fost viziat }
begin
```

```
fillchar(viz, sizeof(viz), FALSE);
end;


procedure vizitat(i :byte);
{ marcam nodul <i> ca fiind vizitat }
begin
viz[i] := TRUE;
end;


function a_fost_vizitat(i :byte):boolean;
{ returneaza TRUE dacă nodul a fost viziat, în caz contrar
  returneaza FALSE }
begin
a_fost_vizitat := viz[i];
end;


procedure prelucrare_nod(x :byte);
{ prelugram nodul, în cazu de fata o sa il afisam doar }
begin
writeln('Prelugram nodul :', x);
end;


begin
init_stiva; { initializam stiva }

citire_matrice_adiacenta; { citim toate elementele grafului }

write('De la ce nod doriti sa pornim : ');
readln(x); { citim nodul de pornire }

push_stiva(x); { adaugam <x> în stiva }

while  not este_vida_stiva() do { cat timp mai avem noduri de prelucrat în stiva }
   begin

   p := acces_stiva; { accesam elementul din varful stivei }

   if not a_fost_vizitat(p) then { dacă elementul nu a fost vizitat }
      begin
```

```
    prelucrare_nod(p); { prelucreazal }
    vizitat(p); { noteazal ca fiind vizitat }
    end;

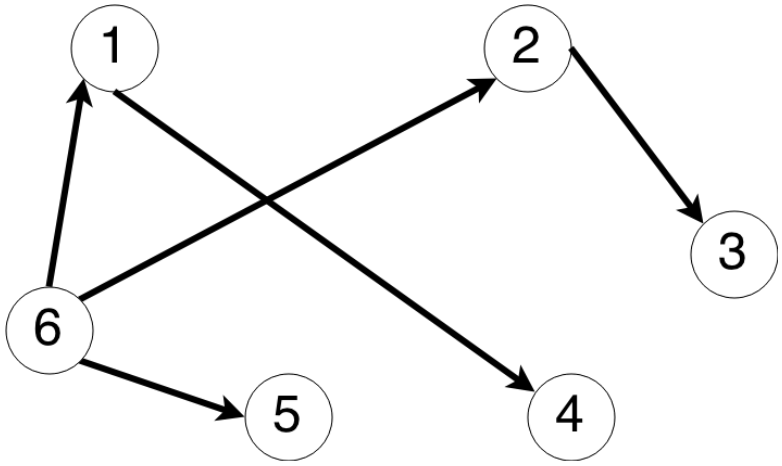
  { cautam un nod de la care putem ajunge de la p, care nu a fost vizitat }
  ok := false; { presupunem ca nu exista nici un nod }
  for i := 1 to n do
    if (a[p, i] = 1) AND ( not a_fost_vizitat(i) ) then
      begin
      push_stiva(i);
      ok := true;
      break;
      end;

  if not ok then
    pop_stiva();


  end;

 writeln;
 end.
```

Dacă avem graful reprezentat grafic astfel :



Pornind de la nodul **6** vom descoperi nodurile în ordinea următore :

6, 1, 4, 2, 3, 5

Pornind de la nodul **1** vom descoperi nodurile în ordinea următoare :

1, 4

Observatie :

Dacă există noduri izolate sau dacă nodul are mai multe componente conexe, metodele de de parcurgere nu vor descoperi toate nodurile, doar pe cele la care poate ajunge.

Comentarii :

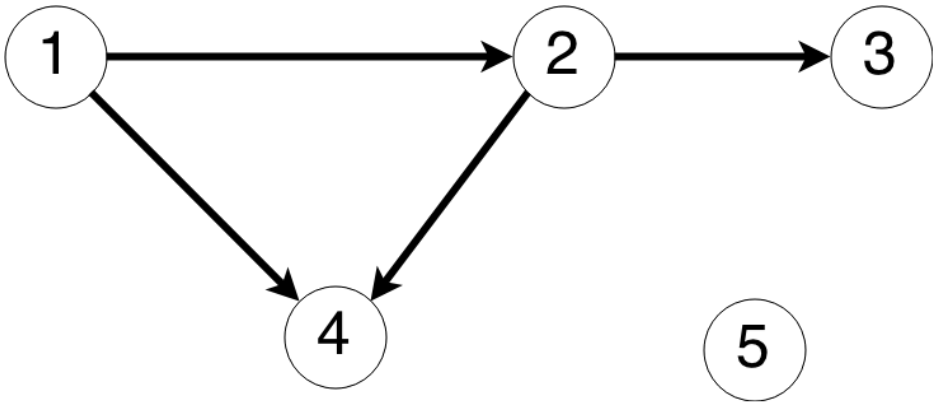
Aceste metode nu parcurg tot graful, dacă avem de exemplu

G = (V, U)

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

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

Reprezentat grafic astfel :



Dacă îl parcurgem în lăţime pornind de la nodul **1** vom descoperi nodurile în ordinea următoare : **1, 2, 4, 3**

Dacă îl parcurgem în adâncime pornind de la nodul **1** vom descoperi nodurile în ordinea următoare : **1, 2, 3, 4**

Nodul **5** nu va fi vizitat deoarece nu exista un nod **x ∈ V a. î. (x, 5) ∈ U.**

H. Drumuri minime şi maxime

În această categorie vom trata problemele următoare:

- determinarea **drumurilor minime** dintr-un graf;
- determinarea **drumurilor** maxime dintr-un graf.

Cerinţele care necesită determinarea drumurilor **minime (maxime)** pot varia :

- Se poate cere determinarea **drumurilor de lungime minimă (maximă) între oricare doua vârfuri din graful G = (V, U) ştiinduse matricea cu costuri asociată grafului C** $\in M_n(\Re)$

- Se poate cere determinarea **drumului de lungime minimă (maximă) dintre un nod i şi un nod j în graful G = (V, U) ştiinduse matricea cu costuri asociată grafului C** $\in M_n(\Re)$

- Se poate cere determinarea **drumului de lungime minimă(maximă) dintre un nod i şi restul nodurilor dintr-un graf G = (V, U) ştiinduse matricea cu costuri asociată grafului C** $\in M_n(\Re)$

În cazul **problemelor de minim** , fiind dat graful **G = (V, U)** i se asociază **matricea costurilor, forma 1**, definită astfel :

C $\in M_n(\Re)$**, unde :**

$$c_{i,j} = \begin{cases} cost\ t\ daca\ intre\ i \quad si \quad j\ exista\ un\ arc\ cu\ costul\ cost\ t \\ 0 \quad daca\ i = j \\ inft \quad daca\ i \neq j\ \ si\ (i,j) \notin U \end{cases}$$

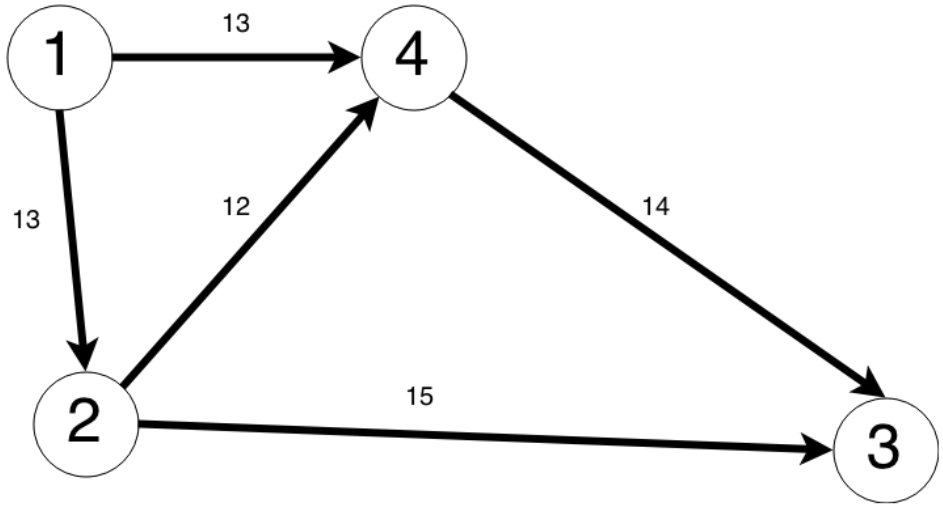
În cazul **problemelor de minimmaxim** , fiind dat graful **G = (V, U)** i se asociază **matricea costurilor, forma 2**, definită astfel :

C $\in M_n(\Re)$**, unde :**

$$c_{i,j} = \begin{cases} cost\ t\ daca\ intre\ i \quad si \quad j\ exista\ un\ arc\ cu\ costul\ cost\ t \\ 0 \quad daca\ i = j \\ -\infty \quad daca\ i \neq j\ \ si\ (i,j) \notin U \end{cases}$$

Exemplu :

Fie graful **G** reprezentat ca în figura de mai jos (costul fiecărui arc fiind scris pe el):



Matricea costurilor de adiacenţa ar arăta în felul următor :

Forma 1 :

$$\begin{bmatrix} 0 & 11 & \infty & 13 \\ \infty & 0 & 15 & 12 \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 14 & 0 \end{bmatrix}$$

Forma 2:

$$\begin{bmatrix} 0 & 11 & -\infty & 13 \\ -\infty & 0 & 15 & 12 \\ -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & 14 & 0 \end{bmatrix}$$

Observaţii :

1. Matricea costurilor **forma 1** diferă de matricea costurilor **forma 2** prin faptul că în loc de ∞ are - ∞.
2. În program, nu se poate folosi conceptul de ∞ sau - ∞, de aceea o sa folosim în loc două constante simetrice cu valori foarte mari *const p_infinit = 1.e10;*
   const m_infinit = -1.e10;

Vom prezenta doi algoritmi care permit determinarea drumurilor minime (maxime) într-un graf:

- **Roy-Floyd**
- **Dijkstra**
- **Metodă brută**

Algoritmul Roy-Floyd

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

Algoritmul se bazează pe următoarea idee :

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

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

În esentă, vom vedea dacă costul unui drum **i - j** este mai mic decât suma costurilor drumurilor **i - k** şi **k - j** , dacă **i - j < i - k + k - j,** ştim că drumul minim de la **i** la **j** trece prin **k.**

În acest stadiu algoritmul nu determină şi drumul minim, doar costul acestui drum minim, pentru a putea vedea şi drumul efectiv de la **i** la **j** folosim o matrice **D** care va salva pentru fiecare element

$d_{i,j}$ = *drumul de la i la j*

după mai multe calcule drumul va fi cel care va avea costul minim .

Matricea drumurilor se iniţializează în felul următor :

- dacă $c_{i,j} < \infty$ atunci exista arc între **i şi j** ,adică $c_{i,j} = \{i, j\}$
- dacă $c_{i,j} = \infty$ atunci nu există arc între **şi j** ,adică $c_{i,j} = \{\phi\}$

Matricea drumurilor se iniţializează după citirea matricii costurilor (de unde vedem ce arce sunt în graf şi cu ce cost), iniţial matricea drumurilor va avea doar arcele ca şi drumuri, ulterior aceste drumuri vor fi modificate în funcţie de costul minim .

Transformările sunt următoarele :

- dacă $c_{i,j} < c_{i,k} + c_{k,j}$ atunci $c_{i,j}$ *devine* $c_{i,k} + c_{k,j}$ şi $d_{i,j} = concat(d_{i,k}, d_{k,j})$
- dacă $c_{i,j} > c_{i,k} + c_{k,j}$ atunci $c_{i,j}$ rămane acelaşi, şi $d_{i,j}$ rămane acelaşi

În exemplu o să citim matricea costurilor dintr-un fişier *costuri.txt* şi vom codifica -∞ cu -1, pentru a simplifica lucrurile, fişierul va avea pe prima linie numărul de noduri (linii şi coloane a matricii) şi pe următoarele linii matricea costurilor.

Exemplu a conţinutului unui fişier :

4

0 4 4 10
-1 0 -1 4
-1 -1 0 3
-1 -1 -1 0

În continuare vă voi prezenta o variantă a algoritmului **Roy-Floyd :**

```
program algoritmul_roy_floyd;
uses crt;
const p_infinit = 1.e10;
      m_infinit = -1.e10;
      n_max = 100;
type
   vector=array[1..n_max] of integer;
   drum = record { va retine  un drum }
      v :vector; { nodurile drumului }
      n :integer; { cate noduri are drumul }
      end;

var drumuri :array[1..n_max, 1..n_max] of drum; { matricea care va retine drumul de la un
                                                                    element la latul }
   costuri :array[1..n_max, 1..n_max] of real; { matricea costurilor }
   i, j, n :integer;
   f :text;

procedure citire_mat_costuri;
{ citim matricea de costuri, de unde aflam şi ce muchii exista în graf
  citirea se face dintr-un fisier 'costuri.txt' care are pe prima linie
  numarul de noduri apoi pe fiecare linie următoare o linie din matricea costurilor }
var i, j :integer;
begin
assign(f, 'costuri.txt');
reset(f);

readln(f, n);

for i := 1 to n do
   begin
   for j := 1 to n do
      begin
      read(f, costuri[i, j]);
      if costuri[i, j] = -1 then
```

```
            costuri[i, j] := p_infinit;
        end;
    writeln;
    end;
close(f);
end;

procedure init_drum(var x :drum);
{ initializam un drum }
begin
x.n := 0;
end;

procedure init_mat_drumuri;
{ initializam matricea de drumuri, pentru arcele existente }
begin
for i := 1 to n do
    for j := 1 to n do
        begin
        if (i <> j) AND (costuri[i, j] < p_infinit) then
            begin
            drumuri[i, j].n := 2;
            drumuri[i, j].v[1] := i;
            drumuri[i, j].v[2] := j;
            end
        else
            init_drum(drumuri[i, j]);
        end;
end;


function concat_drum(x, y :drum):drum;
{ alipim doua drumuri, nodul din mijloc se scrie o singura data }
var i :integer;
    aux :drum;
begin
init_drum(aux); { initializam drumul }
if (x.v[x.n] = y.v[1]) AND (x.n > 0) AND (y.n > 0) then { dacă exista macar un nod în ambele
drumuri }
    begin
    for i := 1 to x.n do
        begin
        aux.n := aux.n + 1;
```

```
        aux.v[aux.n] := x.v[i];
        end;
    for i := 2 to y.n do
        begin
        aux.n := aux.n + 1;
        aux.v[aux.n] := y.v[i];
        end;
    end;

concat_drum := aux;
end;


procedure afis;
{ afisam drumurile Sicosturie }
var i, j, k :integer;
    aux :drum;
begin
for i := 1 to n do
    for j := 1 to n do
        if i <> j then { un drum de la un nod la el insusi nu are sens }
            if (costuri[i, j] < p_infinit) then { dacă exista drum de la i -> j }
                begin
                { afisam costul şi drumul cel mai scurt }
                writeln('Intre nodul ', i, ' ,', j, ' are costul ', costuri[i, j]:4:2);
                writeln('Drumul este :');
                aux := drumuri[i, j];
                write(' ':5);
                for k := 1 to aux.n do
                    write(aux.v[k], ' ');
                writeln;
                end
            else
                writeln('Intre nodul ', i, ', ', j, ' nu exista drum !');
end;

procedure roy_floyd;
{ algoritmul principal }
var i, j, k :integer;
begin
for i := 1 to n do { pentru fiecare nod în graf }
    for j := 1 to n do { pentru pereche de noduri i, j }
        for k := 1 to n do { verificam pentru fiecare nod în graf }
```

```
        if (k <> i) AND ( k <> j) then { dacă nodul k este diferit de nodurile i şi j }
           if ( costuri[i, j] > costuri[i, k] + costuri[k, j]) then { şi dacă drumul i - k - j este mai
                                                                      scurt decat drumul i - j }
              begin
              costuri[i, j] := costuri[i, k] + costuri[k, j]; { modifică costul cu noul cost, mai mic}
              drumuri[i, j] := concat_drum(drumuri[i, k], drumuri[k, j]); { alipeste drumurile i -
                                                                  k şi k - j şi salveazale în locul drumului i - j }
              end;
end;

begin
citire_mat_costuri; { citim costurile }
init_mat_drumuri; { initializam drumurile }

clrscr; { curatam ecranul }

roy_floyd;

afis; { afisam rezultatul }

readln;
end.
```

Pentru a caută drumul maxim dintre toate nodurile trebuie făcute nişte modificări minore algoritmului:

- În timpul citirii matricii costurilor trebuie să iniţializăm matricea costurilor cu -∞ (*m_infinit*) în loc de +∞ (*p_infinit*)

```
          ……..
for i := 1 to n do
   begin
   for j := 1 to n do
      begin
      read(f, costuri[i, j]);
      if costuri[i, j] = -1 then
         costuri[i, j] := m_infinit;
      end;
   writeln;
   end;
          ……..
```

- Mai trebuie modificată şi comparaţia facută între costul actual al drumului **i - j** şi

suma costurilor drumului **i - k** şi **k - j**

```
……..
for i := 1 to n do { pentru fiecare nod în graf }
   for j := 1 to n do { pentru pereche de noduri i, j }
      for k := 1 to n do { verificam pentru fiecare nod în graf }
            if (k <> i) AND ( k <> j) AND ( costuri[i, k] > m_infinit)
             AND ( costuri[k, j] > m_infinit) then { dacă nodul k este d    iferit de
        nodurile i
                                                    si j, şi dacă nu avem drumuri inexistente,
                                                    adică
                                                    costuri = m_infinit  }


          if ( costuri[i, j] < costuri[i, k] + costuri[k, j]) then { şi dacă drumul i - k - j este mai
                                                                  lung decat drumul i - j }
             begin
             costuri[i, j] := costuri[i, k] + costuri[k, j]; { modifică costul cu noul cost, mai mic}
             drumuri[i, j] := concat_drum(drumuri[i, k], drumuri[k, j]); { alipeste drumurile i -
                                                    k şi k - j şi salvează-le în locul drumului i - j }
             end;
          ……..
```

- Şi iniţializarea matricii drumurilor trebuie modificată

```
……..
if (i <> j) AND (costuri[i, j] > m_infinit) then
      begin
      drumuri[i, j].n := 2;
      drumuri[i, j].v[1] := i;
      drumuri[i, j].v[2] := j;
      end
      ……..
```

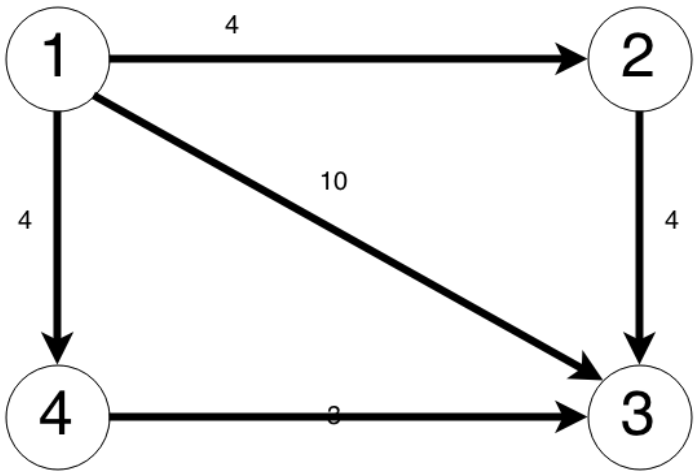
- Un ultim lucru care trebuie modificat e procedura de afişare

```
      ……..
    if i <> j then { un drum de la un nod la el însusi nu are sens }
     if (costuri[i, j] < m_infinit) then { dacă exista drum de la i -> j }
        begin
```

```
{ afişm costul şi drumul cel mai scurt }
writeln('Intre nodul ', i, ' ,', j, ' are costul ', costuri[i, j]:4:2);
writeln('Drumul este :');
……..
```

Exemplu :

Pentru graful :



care are matricea costurilor :

```
4
0 4 4 10
-1 0 -1 4
-1 -1 0 3
-1 -1 -1 0
```

Rezultatul ar trebui sa fie :

```
Între nodul 1 ,2 are costul 4.00
Drumul este :
   1 2
Între nodul 1 ,3 are costul 4.00
Drumul este :
   1 3
Între nodul 1 ,4 are costul 7.00
Drumul este :
   1 3 4
Între nodul 2, 1 nu există drum !
Între nodul 2, 3 nu există drum !
Între nodul 2 ,4 are costul 4.00
Drumul este :
```

2 4
Între nodul 3, 1 nu există drum !
Între nodul 3, 2 nu există drum !
Între nodul 3 ,4 are costul 3.00
Drumul este :
3 4
Între nodul 4, 1 nu există drum !
Intre nodul 4, 2 nu exista drum !
Intre nodul 4, 3 nu exista drum !

Bineînţeles cautarea drumului cel mai lung va duce la trecerea prin mai multe noduri de mai multe ori (asta folosind algoritmul lui Roy-Floyd modifica) deoarece nu există încă un algoritm optimizat pentru găsirea drumurilor maxime .

Algoritmul Dijkstra

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

Programul foloseşte următoarele variabile :

- ***n*** **:** reprezintă **numărul de noduri** al grafului
- ***c*** **:** reprezintă matricea costurilor asociată grafului
- ***pornire*** **:** reprezintă **nodul de plecare**
- ***costuri_drumuri*** **:** vector care conţine costul drumului de la nodul de pornire la oricare alt nod
- ***drum*** **:** vector care va fi populat astfel :
  - ***drum[i]*** reprezintă **nodul care îl precede pe *i* în drumul minim**
- **vizitate :** vector care va ţine evidenţa nodurilor care au fost vizitate

Algoritmul procedează astfel :

- se citeşte matricea costurilor
- vectorul ***vizitate*** se initializează cu ***false***
- se citeşte nodul de pornire şi se noteaza ***vizitate[pornire] = true***
- se completează vectorul ***drum*** astfel :
  - pentru **i = 1 ... n,** ***drum[i] = pornire, c[pornire, i] ≠ ∞ şi i ≠ pornire;***
  - ***drum[i] = 0,*** *altfel;*
- se completează vectorul ***costuri_drumuri*** astfel :
  - ***costuri_drumuri[i] = c[pornire, i],*** pentru ***i = 1 ... n dacă i ≠ pornire***

- ***costuri_drumuri[pornire] = 0***

- cât timp există noduri nevizitate executăm următoarele:
  - se caută nodul ***k*** dintre cele nevizitate, a carui drum de la ***pornire*** la ***k*** este minim, adica ***costuri_drumuri[k] minim*** şi ***vizitate[k] = false***
  - se calculează drumurile de la nodul de pornire la restul nodurilor prin nodul ***k*** în felul următor :
    - calculăm costul minim dintre ***costuri_drumuri[i]*** (costul pe care îl ştim momentan de la ***pornire*** la ***i***) şi ***costuri_drumuri[k]*** (costul pe care îl ştim momentan de la ***pornire*** la ***k) + c[k, i]*** (adică costul arcului ***(k, i)***), cu alte cuvinte, vedem care drum e mai scurt, cel pe care îl avem momentan de la ***pornire*** la ***i*** sau cel de la ***pornire*** la ***k*** şi apoi de la ***k*** la ***i***
      - dacă costul de la ***pornire*** la ***i*** este cel mai mic, atunci nimic nu se întamplă
      - dacă costul de la ***pornire*** la ***k*** este cel mai mic, atunci ***costuri_drumuri[i] = costuri_drumuri[k] + c[k, i]*** şi ***drum[i] = k*** (precedentul nodului ***i*** în drumul minim ***pornire - i*** este ***k***)

În continuare o să vă prezint o versiune a algoritmului lui **Dijkstra** :

```
program algoritmul_dijkstra_min;
uses crt;

const p_infinit = 1.e10;
     m_infinit = -1.e10;
     max_noduri = 1000;

type matrice=array[1..max_noduri, 1..max_noduri] of real;
    vector_i=array[1..max_noduri] of integer;
    vector=array[1..max_noduri] of real;

var c :matrice; { matricea costurilor }
   costuri_drum : vector; { costul drumului de la nodu de pornire la <i> }
   drum : vector_i; { drumul de la nodu de pornire la oricare nod }
   vizitate :array[1..max_noduri] of boolean; { tinem cont de ce noduri am vizitat }
   n, pornire :integer;
   f :text;

procedure citire_matrice_costuri;
var i, j :integer;
begin
```

```
assign(f, 'costuri.txt');
reset(f);

readln(f, n);

for i := 1 to n do
   begin
   for j := 1 to n do
      begin
      read(f, c[i, j]);
      if c[i, j] = -1 then { intifint în text e codificat ca şi <-1> }
         c[i, j] := p_infinit;
      end;
   readln(f);
   end;
close(f);
end;

procedure init_vizitate;
{ initializam vectorul cu noduri vizitate }
var i :integer;
begin
for i := 1 to n do
   vizitate[i] := False;
end;

procedure vizitat(x :integer);
{ notam nodul <x> ca fiind vizitat }
begin
vizitate[x] := True;
end;

function exista_neviz:boolean;
{ returnam True dacă mai exista noduri nevizitate }
var i :integer;
   aux :boolean;
begin
aux := false; { presupunem ca nu mai exista noduri de vizitat }
for i := 1 to n do
   if vizitate[i] = False then { dacă nodul <i> nu a fost vizitat }
      begin
      aux := true; { atunci mai exista noduri de vizitat }
      break; { oprin cautarea }
```

```
        end;
exista_neviz := aux;
end;

function este_neviz(x :integer):boolean;
{ returnam True dacă nodul <x> este nevizitat, false în faz contrar }
begin
este_neviz := not vizitate[x];
end;


procedure init_drum_costuri;
{ initlializam costurile fiecarui drum }
var i : integer;
begin
for i := 1 to n do
   if (i <> pornire)  then { dacă i <> pornire  }
      costuri_drum[i] := c[pornire, i] { drumul e constituit doar dintr-un arc şi costul este cher
costul arcului }
   else
      costuri_drum[i] := 0; { nu exista arc, deci initializam cu 0 }

end;

procedure init_drum;
{ initlializam vectorul drumurilor de la nodu de pornire la restu }
var i :integer;
begin
for i := 1 to n do
   if (i <> pornire) AND (c[pornire, i] < p_infinit) then { dacă exista arc de la nodu de pornire
la i }
      drum[i] := pornire { fiind doar un arc, extremitatea initiala e chear nodul de pornire }
   else
      drum[i] := 0 ;
end;

procedure dijkstra;
var i, k :integer;
   min :real;
begin
while exista_neviz() do { cat mai exista noduri nevizitate }
   begin
   min := p_infinit;
```

```
   k := 0;

   for i := 1 to n do  { cautam nodul la care putem ajunge cel mai usor }
      if este_neviz(i) AND (costuri_drum[i] < min )then
         begin
         min := costuri_drum[i];
         k := i;
         end;

   vizitat(k);

   for i := 1 to n do
      if este_neviz(i) AND (c[k, i] < p_infinit) then { dacă exista arc  de la k la i }
         if costuri_drum[i] > costuri_drum[k] + c[k, i] then { dacă drumul de la nodu pe care il
avem de la nodu de
                                                pornire pana la <i> este mai lung decat drumul prin
nodul
                                                <k> atunci drumul minim trece prin k }
            begin
            costuri_drum[i] :=  costuri_drum[k] + c[k, i]; { costul minim e cel al drumului pana
la <k> + de la <k> la <i>}
            drum[i] := k; { la nodul <i> se ajunge prin nodul <k> }
            end;
   end;
end;

procedure afisare_drum(i :integer);
{ afisam drumul de la nodu de pornire pana la nodul <i> }
begin
if i <> 0 then
   begin
   afisare_drum(drum[i]);
   write(i, ' ');
   end;
end;

procedure afis;
var i :integer;
begin
for i := 1 to n do
   if i <> pornire then
      if drum[i] <> 0 then  { am gasit un drum de la <pornire> la <i> }
         begin
```

```
            writeln('Drumul de la ', pornire, ' la ', i, ' are lungimea :', costuri_drum[i]:4:2);
            writeln('Si este compus din ');
            write(' ':7);
            afisare_drum(i);
            writeln;
            end
        else
            writeln('Nu exista drum de la ', pornire, ' la ', i);
end;
begin
citire_matrice_costuri; { citim matricea de costuri }

write('Drum de pornire :');
readln(pornire);

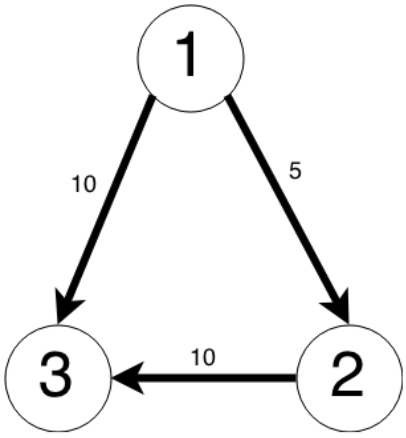
vizitat(pornire);
init_drum;
init_drum_costuri;

dijkstra;
afis;
end.
```

Algoritmul lui **Dijkstra** nu poate fi modificat pentru a căuta drumurile maxime, el va găsi drumurile maxime aproximative.

Exemplu :

Fie graful reprezentat grafic :



Dacă dorim sa căutăm cel mai scurt drum de la nodul 1 la restul nodurilor, rezultatul ar fi următorul :

```
Nodul de pornire :1
Drumul de la 1 la 2 are lungimea :10.00
Şi este compus din
     1 2
Drumul de la 1 la 3 are lungimea :5.00
Şi este compus din
     1 3
```

Nu se cunoşte încă un algoritm eficient pentru căutarea drumurilor maxime !

Dar o implementare ar fi :

```
program algoritmul_dijkstra_minim;
uses crt;

const p_infinit = 1.e10;
     m_infinit = -1.e10;
     max_noduri = 1000;

type matrice=array[1..max_noduri, 1..max_noduri] of real;
    vector_i=array[1..max_noduri] of integer;
    vector=array[1..max_noduri] of real;

var c, c2 :matrice; { matricea costurilor }
   costuri_drum : vector; { costul drumului de la nodu de pornire la <i> }
   drum : vector_i; { drumul de la nodu de pornire la oricare nod }
   vizitate :array[1..max_noduri] of boolean; { tinem cont de ce noduri am vizitat }
   n, pornire :integer;
   f :text;

procedure citire_matrice_costuri;
var i, j :integer;
begin
assign(f, 'costuri.txt');
reset(f);

readln(f, n);

for i := 1 to n do
   begin
   for j := 1 to n do
      begin
      read(f, c[i, j]);
```

```
      if c[i, j] = -1 then { intifint in text e codificat ca si <-1> }
         c[i, j] := p_infinit;
      end;
   readln(f);
   end;
close(f);
end;

procedure init_vizitate;
{ initializam vectorul cu noduri vizitate }
var i :integer;
begin
for i := 1 to n do
   vizitate[i] := False;
end;

procedure vizitat(x :integer);
{ notam nodul <x> ca fiind vizitat }
begin
vizitate[x] := True;
end;

function exista_neviz:boolean;
{ returnam True daca mai exista noduri nevizitate }
var i :integer;
   aux :boolean;
begin
aux := false; { presupunem ca nu mai exista noduri de vizitat }
for i := 1 to n do
   if vizitate[i] = False then { daca nodul <i> nu a fost vizitat }
      begin
      aux := true; { atunci mai exista noduri de vizitat }
      break; { oprin cautarea }
      end;
exista_neviz := aux;
end;

function este_neviz(x :integer):boolean;
{ returnam True daca nodul <x> este nevizitat, false in faz contrar }
begin
este_neviz := not vizitate[x];
end;
```

```
procedure init_drum_costuri;
{ initlializam costurile fiecarui drum }
var i : integer;
begin
for i := 1 to n do
   if (i <> pornire)  then { daca i nu este nodul de pornire }
      costuri_drum[i] := c[pornire, i] { drumul e constituit doar dintr-un arc si costul
este cher costul arcului }
   else
      costuri_drum[i] := 0; { nu exista arc, deci initializam cu 0 }

end;

procedure init_drum;
{ initlializam vectorul drumurilor de la nodu de pornire la restu }
var i :integer;
begin
for i := 1 to n do
   if (i <> pornire) AND (c[pornire, i] < p_infinit) then { daca exista arc de la nodu de
pornire la i }
      drum[i] := pornire { fiind doar un arc, extremitatea initiala e chear nodul de
pornire }
   else
      drum[i] := 0 ;
end;

procedure dijkstra;
var i, k, c1 :integer;
   min :real;
begin
c1 := 0;
while exista_neviz() and (c1 < n) do { cat mai exista noduri nevizitate }
(* for j := 1 to n-2 do *)
   begin
   c1 := c1 + 1;
   min := p_infinit;
   k := 0;

   for i := 1 to n do  { cautam nodul la care putem ajunge cel mai usor }
      if este_neviz(i) AND (costuri_drum[i] < min )then
         begin
         min := costuri_drum[i];
```

```
                k := i;
                end;

            vizitat(k);

            for i := 1 to n do
               if este_neviz(i) AND (c[k, i] <> p_infinit) then { daca exista arc  de la k la i }

                  if (costuri_drum[i] > costuri_drum[k] + c[k, i]) OR (costuri_drum[i] =
p_infinit) then { daca drumul de la nodu pe care il avem de la nodu de
                                                pornire pana la <i> este mai lung decat
drumul prin nodul
                                                <k> atunci drumul minim trece prin k }
                     begin
                     costuri_drum[i] :=  costuri_drum[k] + c[k, i]; { costul minim e cel al
drumului pana la <k> + de la <k> la <i>}
                     drum[i] := k; { la nodul <i> se ajunge prin nodul <k> }
                     end;
               end;
            end;

            procedure afisare_drum(i :integer);
            { afisam drumul de la nodu de pornire pana la nodul <i> }
            begin
            if i <> 0 then
               begin
               afisare_drum(drum[i]);
               write(i, ' ');
               end;
            end;

            function calc_cost(k :integer):real;
            begin

            if k <> 0 then
               begin
               calc_cost := c2[drum[k], k] + calc_cost(drum[k]);
               end
            else
               calc_cost := 0;
            end;

            procedure afis;
```

```
var i :integer;
begin
for i := 1 to n do
   if i <> pornire then
      if drum[i] <> 0 then  { am gasit un drum de la <pornire> la <i> }
         begin
         writeln('Drumul de la ', pornire, ' la ', i, ' are lungimea :', calc_cost(i):4:2);
         writeln('Si este compus din ');
         write(' ':7);
         afisare_drum(i);
         writeln;
         end
      else
         writeln('Nu exista drum de la ', pornire, ' la ', i);
end;

procedure re_init_costuri;
var i, j :integer;
begin
c2 := c;
for i := 1 to n do
   for j := 1 to n do
      if (c[i, j] < p_infinit) AND (c[i, j] > 0) then
         c[i, j] :=  1 /  c[i, j];
end;




begin
citire_matrice_costuri; { citim matricea de costuri }
re_init_costuri;

write('Nodul de pornire :');
readln(pornire);

vizitat(pornire);

init_drum;
init_drum_costuri;

dijkstra;
```

```
afis;
end.
```

Metodă Brută

O metodă de aflare a drumurilor maxime este cea brută, folosind un algoritm BackTracking, care generează toate drumurile posibile de la un nod **x** la un nod **y** si pastrează doar drumul cu costul maxim .

O implementare poate fi :

```
program btk_maxim;
type vector = array[1..1000] of integer;
   drum = record
   v :vector;
   n :integer;
   cost :real;
   end;

var aux, sol :drum;
   f :text;
   c, r, first, last, max, n, i, j, m :integer;
   drumuri :array[1..100, 1..100] of drum;
   costuri :array[1..100, 1..100] of real;
   stiva :array[1..100] of integer;

procedure citire_costuri;
var i, j :integer;
begin
assign(f, 'costuri.txt');
reset(f);

readln(f, n);

for i := 1 to n do
   begin
   for j := 1 to n do
      read(f, costuri[i, j]);
   readln(f);
   end;
```

```
close(f);
end;

procedure init(k :integer);
begin
stiva[k] := 0;
end;

procedure init_drum(dr :drum);
begin
dr.n := 0;
dr.cost := 0;
end;


function continua(k :integer):boolean;
var i :integer;
begin
continua := (stiva[1] = first) AND (k <= m ) ;

for i := 2 to k do
   if costuri[stiva[i-1], stiva[i]] = -1 then
      begin
      continua := false;
      end;

for i := 1 to k-1 do
   for j := i+1 to k do
      if stiva[i] = stiva[j]  then
         begin
         continua := false;
         end;
end;

function exista(k :integer):boolean;
begin
exista := stiva[k] < n;
end;

function solutie(k :integer):boolean;
begin
solutie := stiva[k] = last;
end;
```

```
procedure optimizare(k :integer);
var i :integer;
   cost :real;
begin
cost := 0;
for i := 2 to k do
   begin
   cost := costuri[stiva[i-1], stiva[i]] + cost;
   end;

if sol.cost < cost then
   begin
   init_drum(sol);

   for i := 1 to k do
      sol.v[i] := stiva[i];
   sol.n := k;
   sol.cost := cost;
   end;
end;

procedure bkt(k :integer);
begin
init(k);

while exista(k) do
   begin
   inc(stiva[k]);

   if continua(k) then
      if solutie(k) then
         optimizare(k)
      else
         bkt(k+1);
   end;
end;

begin
citire_costuri;

{ aflam numarul de muchii }
m := 1;
```

```
        for i := 1 to n do
           for j := 1 to n do
              if (costuri[i, j] <> -1) AND ( i <> j) then
                 inc(m);

        first := 1;
        last := 1;

        while first <= n do
           begin
           last := 1;
           while last <= n do
              begin
              writeln(first, ' - ', last);
              writeln;
              if first  <>  last then
                 begin
                 writeln('Cautam drumul maxim dintre ', first, ' si ', last );

                 init_drum(sol);

                 bkt(1);

                 if sol.n > 0 then
                    begin
                    writeln('Drumul maxim de la ', first, ' la ', last , ' are costul :',
sol.cost:4:2);
                    writeln('Drumul este compus din :');
                    for r := 1 to sol.n do
                       write(sol.v[r], ' ');
                    writeln;
                    end
                 else
                    writeln('Nu exista drum de la ', first , ' la ',  last);

                 drumuri[i, j] := sol;
                 end
              else
                 writeln('Nu am intrat');

              last := last + 1;
              end;
           first := first + 1;
```

```
        end;

    end.
```

Program auxiliar :

Deoarece programele din această lucrare citesc matricea de adiacenţă şi cea a costurilor dintr-in fisier, programul următor va citi arcele şi costul acestora şi va genera şi salva într-un fisier matricea costurilor automat :

```
program algoritm_creeare_matrice_costuri;
var i, j, n, m, x, y :integer;
   v :array[1..1000, 1..1000] of real;
   f :text;
   s :string;

begin
write('Numarul de noduri :');
readln(n);

write('Numarul de arce :');
readln(m);

{ initializam matricea costurilor }
for i := 1 to n do
   for j := 1 to n do
      if i = j then
         v[i, j] := 0
      else
         v[i, j] := -1;


{ citim muchiile }
for i := 1 to m do
   begin
   write('x= ');
   readln(x);

   write('y= ');
   readln(y);
```

```
    write('Costul arcului ', x, ' ', y,' :');
    readln(v[x, y]);
    end;

{ afisam matricea costurilor }

writeln;
writeln;
for i := 1 to n do
    begin
    for j := 1 to n do
        write(v[i, j]:4:2, ' ');
    writeln;
    end;
writeln;
writeln;
repeat
    write('Doriti sa salvam matricea costurilor intr-un fisier?(y/n) ');
    readln(s);
until (s = 'y') OR (s = 'n');

if s = 'y' then
    begin
    write('Numele fisierului in care doriti sa salvam : ');
    readln(s);

    assign(f, s);
    rewrite(f);

    writeln(f, n);

    for i := 1 to n do
        begin
        for j := 1 to n do
            write(f, v[i, j]:4:2, ' ');
        writeln(f);
        end;

    close(f);
    end;
end.
```

I. Program Complex :

În continuare vă voi prezenta un program care toţi algoritmii prezentaţi pană acum :

```
program program_complex;
uses crt;
const n_max = 1000;
      p_inf = 1.e10;
      m_inf = -1.e10;
type
   vector_int = array[1..n_max] of integer;
   vector_bool = array[1..n_max] of boolean;
   matrice_int = array[1..n_max, 1..n_max] of integer; { vector cu elementei integer }
   matrice_real = array[1..n_max, 1..n_max] of real; { vector cu elemente reale }
   matrice_bool = array[1..n_max, 1..n_max] of boolean; { vector cu elemente de tip boolean }

   drum = record { o structura care retine un drum }
      v :vector_int; { elementele drumului }
      n : integer; { numarul de elemente ale drumului }
      end;

   stiva = object
      vector :vector_int; { vector de stocare }
      n :integer; { numar de elemente }
      procedure init; { initializam stiva }
      procedure push(x :integer); { adaugam un element in stiva }
      procedure pop; { eliminam un element din stiva }
      function acces:integer; { accesam elementul din varful stivei }
      function vida:boolean; { verificam daca stiva este sau nu vida }
   end;

   coada = object
      vector :vector_int; { vector de stocare }
      n, i :integer; { numar de elemente }
      procedure init; { initializam coada }
      procedure push(x :integer); { adaugam un element in coada }
      procedure pop; { scoatem un element din coada }
      function acces:integer; { accesam primul element din coada }
      function vida:boolean; { verificam daca coada este vida sau nu }
```

```
    end;

    vizitate = object
        vector :vector_bool;
        n :integer;
        procedure init(x :integer); { initializam vectorul de noduri vizitate }
        procedure vizitat(x :integer); { notam nodul <x> ca fiind vizitat }
        function exista_neviz: boolean; { vedem daca mai avem noduri nevizitate }
        function este_neviz(x :integer): boolean; { vedem daca nodul <x> este nevizitat }
    end;

    matrice_drum = array[1..n_max, 1..n_max] of drum;

    graf = object
        nume, fisier, cauta_roy, cauta_dj :string; { numele grafului, si numele fisierului de unde a
fost citit, cauta - reprezinta ce s-a cautat, drumul
                                    minim sau maxim }
        n :integer; { n - numarul de noduri al grafului }
        citit, roy_floyd, dijkstra :boolean; { retine daca graful a fost sau nu citit }
        mat_adiacenta :matrice_bool; { matricea de adiacenta }
        mat_drum :matrice_drum; { matricea de drumuri, pentru roy_floyd }
        mat_costuri, mat_costuri_roy, mat_costuri_dj :matrice_real; { matricea de costuri }
        matrice_drum_dj :matrice_int; { fiecare linie din matrice va fi un vector de drumuri
pentru algoritmul dikkstra }
        coada :coada; { coada }
        stiva :stiva; { stiva }
        vizitate :vizitate; { vector cu noduri vizitate }

        procedure init;
        procedure initializare(x :integer); { procedure de initializare a grafului dupa ce stim
numarul de noduri }
        procedure initializare_fisier(x :string); { initlializam numele fisierului }
        procedure afisare; { afisam lista arcelor }
        procedure citire_mat_adiacenta; { procedure de citire a matrici de adiacenta din fisier }
        procedure citire_mat_costuri; { procedura care citeste matricea de costuri }
        procedure config_mat_costuri(x :real); { initializam matricea costurilor pentru a cauta
drumurile maxime sau minime }
        procedure parcurgere_in_latime(x :integer); { parcurgem graful in latime pornind de la
nodul <x> si afisam nodurile gasite }
        procedure parcurgere_in_adancime(x :integer); { parcurgem graful in adancime pornind
de la nodul <x> si afisam nodurile gasite }
        procedure set_nume(s :string); { setam un nume }
        procedure set_file(s :string); { setam un fisier }
```

```
      procedure afis_mat_adiacenta; { afisam matricea de adiacenta }
      procedure afis_mat_costuri; { afisam matricea costurilor }
      procedure afis_lista_noduri; { afisam lista nodurilor }
      procedure afis_lista_noduri_cost; { afisam lista nodurilor si a costurilor }
      procedure roy_floyd_min; { aflam drumurile minime folosind algoritmul roy_floyd }
      procedure roy_floyd_max; { aflam drumurile minime folosind algoritmul roy_floyd }
      procedure dijkstra_min(x :integer); { aflam drumurile minime del a <x> la restul
nodurilor folosind algorimtul dijkstra}
      procedure afis_drum_roy(s :string);  { afisam drumurile maxime }
      procedure afis_drum_dj(x :integer); {afisam drumurile minime de la un nod anume la alte
noduri }
      procedure drum_min(a, b :integer); { afisam drumul minim dintre nodul <a> si nodul
<b> }
      procedure drum_max(a, b :integer); { afisam drumul maxim dintre nodul <a> si nodul
<b> }
      procedure bkt(a, b :integer); { calculam drumul maxim de la a, la b }
   end;

var gf :graf;
{=============================================================
============================}
{ Metode drumuri }
procedure afis_dr(x, i :integer);
begin
if i <> 0 then
   begin
   afis_dr(x, gf.matrice_drum_dj[x, i]);
   write(i, ' ');
   end;
end;
procedure afis_mat_drum;
var i, j, k :integer;
begin
for i := 1 to gf.n do
   for j := 1 to gf.n do
      begin
      writeln('Drum : ', i, ' - ', j);
      for k := 1 to gf.mat_drum[i, j].n do
         write(gf.mat_drum[i, j].v[k], ' ');
      writeln;
      end;
end;
function inf_or_nr(x :real):string;
```

```
begin
if x = p_inf then
   inf_or_nr := 'infinit'
else
   if x = m_inf then
      inf_or_nr := '-infinit'
   else
      str(x:4:2, inf_or_nr);
end;

procedure init_drum(var x :drum);
{ initializam un drum }
begin
x.n := 0;
end;

procedure init_mat_drumuri_min;
{ initializam matricea de drumuri, pentru arcele existente }
var i, j :integer;
begin
for i := 1 to gf.n do
   for j := 1 to gf.n do
      begin
      if (i <> j) AND (gf.mat_costuri[i, j] < p_inf) then
         begin
         gf.mat_drum[i, j].n := 2;
         gf.mat_drum[i, j].v[1] := i;
         gf.mat_drum[i, j].v[2] := j;
         end
      else
         init_drum(gf.mat_drum[i, j]);

      end;
end;


procedure init_mat_drumuri_max;
{ initializam matricea de drumuri, pentru arcele existente }
var i, j :integer;
begin
for i := 1 to gf.n do
   for j := 1 to gf.n do
      begin
```

```
        if (i <> j) AND (gf.mat_costuri[i, j] > m_inf) then
           begin
           gf.mat_drum[i, j].n := 2;
           gf.mat_drum[i, j].v[1] := i;
           gf.mat_drum[i, j].v[2] := j;
           end
        else
           init_drum(gf.mat_drum[i, j]);
        end;
end;

function concat_drum(x, y :drum):drum;
{ alipim doua drumuri, nodul din mijloc se scrie o singura data }
var i :integer;
   aux :drum;
begin
init_drum(aux); { initializam drumul }
if (x.v[x.n] = y.v[1]) AND (x.n > 0) AND (y.n > 0) then { daca exista macar un nod in ambele
drumuri }
   begin
   for i := 1 to x.n do
      begin
      aux.n := aux.n + 1;
      aux.v[aux.n] := x.v[i];
      end;
   for i := 2 to y.n do
      begin
      aux.n := aux.n + 1;
      aux.v[aux.n] := y.v[i];
      end;
   end;

concat_drum := aux;
end;

procedure init_vector_drum(x :integer);
var i :integer;
begin
for i := 1 to gf.n do
   begin
   if (x <> i) AND (gf.mat_costuri[x, i] < p_inf) then
      gf.matrice_drum_dj[x, i] := x
   else
```

```
      gf.matrice_drum_dj[x, i] := 0;
   writeln;
   end;
end;

{=============================================================
============================}
{ Metode obiect vizitate }
procedure vizitate.init(x :integer);
{ initializam vectorul de vizitate cu false }
var i :integer;
begin
n := x;

for i := 1 to n do
   vector[i] := False;
end;

procedure vizitate.vizitat(x :integer);
{ notam nodul <x> ca fiind vizitat }
begin
vector[x] := true;
end;

function vizitate.exista_neviz:boolean;
{ verificam daca mai exista sau nu noduri nevizitate }
var i :integer;
begin
exista_neviz := false;
for i := 1 to n do
   if vector[i] = false then
      begin
      exista_neviz := true;
      break;
      end;
end;

function vizitate.este_neviz(x :integer):boolean;
{ verificam daca nodul <x> a fost sau nu vizitat }
begin
este_neviz := not vector[x];
end;
```

```
{============================================================
============================}
{ Metode obiect stiva }
procedure stiva.init;
{ initializam stiva }
begin
n := 0;
end;

procedure stiva.push(x :integer);
{ adaugam un element in stiva }
begin
n := n + 1;
vector[n] := x;
end;

procedure stiva.pop;
{ eliminam un vector din stiva }
begin
n := n - 1;

if vida() then { daca stiva devine vida }
   init(); { o initializam }
end;

function stiva.acces:integer;
{ returnam elementul din capul stivei }
begin
acces := vector[n];
end;

function stiva.vida:boolean;
{ returnam <True> daca stiva e goala si false in caz contrar }
begin
vida := ( n <= 0 );
end;
{============================================================
============================}
{ Metode obiect coada }
procedure coada.init;
{ initializam coada }
begin
i := 1;
```

```
n := 0;
end;

procedure coada.push(x :integer);
{ adaugam un element in coada }
begin
n := n + 1;
vector[n] := x;
end;

procedure coada.pop;
{ eliminam un vector din coada }
begin
i := i + 1;

if vida() then { daca stiva este vida }
   init(); { o initializam }
end;

function coada.acces:integer;
begin
acces := vector[i];
end;

function coada.vida:boolean;
begin
vida := (i > n);
end;
{============================================================
============================}
{ Metode obiect Graf }

procedure graf.initializare(x :integer);
{ initlializam toate componentele in functie de n }
var i, j :integer;
begin
n := x;

{ initlializam matricea de adiacenta cu False }
for i := 1 to n do
   for j := 1 to n do
      mat_adiacenta[i, j] := False;
```

```
end;

procedure graf.initializare_fisier(x :string);
begin
fisier := x;
end;

procedure graf.citire_mat_costuri;
{ citim matricea de costuri si initializam si matricea de adiacenta, si n o data cu asta }
var f :text;
   x, i, j :integer;
begin
assign(f, fisier);
reset(f);

readln(f, x);

{ initlializam graful cu numarul de noduri <x> }
initializare(x);

for i := 1 to x do
   begin
   for j := 1 to x do
      begin
      read(f, mat_costuri[i, j]);
      if mat_costuri[i, j] > 0 then
         mat_adiacenta[i, j] := True;
      end;
   readln(f);
   end;
close(f);
n := x;
citit := true; { marcam graful ca fiind citit }
end;

procedure graf.config_mat_costuri(x :real);
{ configuram matricea costurilor, adica inlocuim <-1> cu x, care va fi <p_inf> sau <m_inf> }
var i, j :integer;
begin
for i := 1 to n do
   for j := 1 to n do
      if (mat_costuri[i, j] = -1) OR (mat_costuri[i, j] = p_inf) OR (mat_costuri[i, j] = m_inf)
then
```

```
            mat_costuri[i, j] := x;
end;

procedure graf.citire_mat_adiacenta;
var f :text;
   aux, i, j, x :integer;

begin
assign(f, fisier);
reset(f);

readln(f, x);

{ initlializam elementele matrici in fuctie de <x> noduri }
initializare(x);

for i := 1 to x do
   begin
   for j := 1 to x do
      begin
      read(f, aux);
      if aux = 1 then
         mat_adiacenta[i, j] := true;
      end;
   readln(f);
   end;
close(f);

{ initliazizam matricea costurilor cu 1 }

for i := 1 to n do
   for j := 1 to n do
      if i = j then
         mat_costuri[i, j] := 0
      else
         if mat_adiacenta[i, j] then
            mat_costuri[i, j] := 1
         else
            mat_costuri[i, j] := -1;

citit := true; { marcam graful ca fiind citit }
end;
```

```
procedure graf.afisare;
var i, j :integer;
begin
writeln('Graful cu numele :', nume);
writeln('Are urmatoarele arce :');
for i := 1 to n do
   for j := 1 to n do
      if mat_adiacenta[i, j] then
         writeln(i, ' - ', j);
end;

procedure graf.init;
{ procedura care initializeaza un graf }
begin
nume := 'No Name';
fisier := '';
n := 0;
cauta_roy := 'none';
cauta_dj := 'none';
citit := false;
end;

procedure graf.set_nume(s :string);
begin
nume := s;
end;

procedure graf.parcurgere_in_latime(x :integer);
var i, p :integer;
   procedure prelucrare(nod :integer);
   { prelucram nodul <nod>, momentan doar in afisam }
   begin
   write(nod, ' ');
   end;

begin

{ initializam coada }
coada.init;

{ initializam vectorul vizitate }
vizitate.init(n);
```

```
{ adaugam nodul <x> in coada }
coada.push(x);

writeln('Graful cu numele <', nume, '>, parcurs in latime de la nodul : ',x);
writeln('Ii gasim urmatoarele noduri :');

while not coada.vida do
   begin

   p := coada.acces;
   if vizitate.este_neviz(p) then { nodul din varful stivei trebuie procesat }
      begin
      prelucrare(p);
      vizitate.vizitat(p);
      end
   else
      coada.pop;

   for i := 1 to n do
      if mat_adiacenta[p, i] AND vizitate.este_neviz(i) then
         coada.push(i);
   end;

writeln;
end;

procedure graf.parcurgere_in_adancime(x :integer);
var i, p :integer;
   ok :boolean;
   procedure prelucrare(nod :integer);
   { prelucram nodul <nod>, momentan doar in afisam }
   begin
   write(nod, ' ');
   end;
begin

{ initializam stiva }
stiva.init;

{ initializam vectorul vizitate }
vizitate.init(n);

{ adaugam nodul <x> in stiva }
```

```
stiva.push(x);
(* vizitate.vizitat(stiva.acces); *)


writeln('Graful cu numele <', nume, '>, parcurs adancime de la nodul : ', x);
writeln('Ii gasim urmatoarele noduri :');


while not stiva.vida do
   begin

   p := stiva.acces; { salvam elementul din capul stivei }

   if vizitate.este_neviz(p) then { nodul din varful stivei trebuie procesat }
      begin
      prelucrare(p);
      vizitate.vizitat(p);
      end;

   ok := false; { presupunem ca nu am gasti nici un alt nod nevizitat pornind de la <p> }

   for i := 1 to n do
      if mat_adiacenta[p, i] AND vizitate.este_neviz(i) then
         begin
         stiva.push(i); { adaugam nodul gasit in stiva }
         ok := true; { memoram faptul ca am gasit un nod de la <p> }
         break;
         end;


   if not ok then  { daca nu am gasit nici un nod, eliminam nodul din stiva }
      stiva.pop;
   end;

writeln;
end;

procedure graf.set_file(s :string);
{ setam un fisier de citire }
begin
fisier := s;
end;
```

```
procedure graf.afis_mat_adiacenta;
{ afisam matricea de adiacenta }
var i, j :integer;
begin
for i := 1 to n do
   begin
   for j := 1 to n do
      if mat_adiacenta[i, j] then
         write(' * ')
      else
         write(' x ');
   writeln;
   end;
end;

procedure graf.afis_mat_costuri;
{ afisam matricea costurilor }
var i, j :integer;
begin
for i := 1 to n do
   begin
   for j := 1 to n do
      write(' ',inf_or_nr(mat_costuri[i, j]):8, ' ');
   writeln;
   end;
end;

procedure graf.afis_lista_noduri;
var i, j :integer;
begin
writeln('Graful cu numele ', nume, ' are urmatoarele noduri :');
for i := 1 to n do
   for j := 1 to n do
      if mat_adiacenta[i, j] then
         writeln(i, ' - ', j);
end;

procedure graf.afis_lista_noduri_cost;
var i, j :integer;
begin
writeln('Graful cu numele ', nume, ' are urmatoarele noduri :');
for i := 1 to n do
   for j := 1 to n do
```

```
      if mat_adiacenta[i, j] then
         writeln(i, ' - ', j, '  : ', mat_costuri[i, j]:4:2);
end;

procedure graf.roy_floyd_min;
var i, j, k :integer;
   a, b, c :string;
begin
{ initializam matricea de costuri }

config_mat_costuri(p_inf);

mat_costuri_roy := mat_costuri;


{ initializam matricea de drumuri }
init_mat_drumuri_min;

writeln(' Vom afisa si modul de lucru al algoritmului :');
writeln;
writeln;

for i := 1 to n do
   for j := 1 to n do
      for k := 1 to n do
         if (k <> i) AND (k <> j) then
            begin

            a := inf_or_nr(mat_costuri_roy[i, j]);
            b := inf_or_nr(mat_costuri_roy[i, k]);
            c := inf_or_nr(mat_costuri_roy[k, j]);
            writeln('Vedem daca drumul direct de la ', i, ' la ', j, ' cu costul : ', a);
            writeln('Este mai mic decat suma drumului de la ', i, ' la ', k, ' cu costul :', b);
            writeln('cu drumul de la ', k, ' la ', j, ' cu costul :', c);

            if (mat_costuri_roy[i, j] > mat_costuri_roy[i, k] + mat_costuri_roy[k, j]) then
               begin
               writeln('Costul fiind mai mic, modificam matricea drumurilor si a costurilir !');
               mat_costuri_roy[i, j] := mat_costuri_roy[i, k] + mat_costuri_roy[k, j];
               mat_drum[i, j] := concat_drum(mat_drum[i, k], mat_drum[k, j]);
               end;
            writeln;
            end;
```

```
writeln;
writeln;
writeln('Afisam drumurile minime obtinute folosind algoritmul lui Roy-Floyd :');
afis_drum_roy('min');
cauta_roy := 'min';
end;

procedure graf.roy_floyd_max;
var i, j, k :integer;
   a, b, c :string;
begin
{ initializam matricea de costuri }
config_mat_costuri(m_inf);

mat_costuri_roy := mat_costuri;

{ initializam matricea de drumuri }
init_mat_drumuri_max;

writeln(' Vom afisa si modul de lucru al algoritmului :');
writeln;
writeln;

for i := 1 to n do
   for j := 1 to n do
      for k := 1 to n do
         begin
         if (k <> i) AND (k <> j) AND (i <> j) AND (mat_costuri_roy[i, k] > m_inf) AND
(mat_costuri_roy[k, j] > m_inf) then
            begin
            a := inf_or_nr(mat_costuri_roy[i, j]);
            b := inf_or_nr(mat_costuri_roy[i, k]);
            c := inf_or_nr(mat_costuri_roy[k, j]);
            writeln('Vedem daca drumul direct de la ', i, ' la ', j, ' cu costul : ', a);
            writeln('Este mai mare decat suma drumului de la ', i, ' la ', k, ' cu costul :', b);
            writeln('cu drumul de la ', k, ' la ', j, ' cu costul :', c);
            if (mat_costuri_roy[i, j] < mat_costuri_roy[i, k] + mat_costuri_roy[k, j]) then
               begin
               writeln('Costul fiind mai maxim, modificam matricea drumurilor si a costurilir !');
               mat_costuri_roy[i, j] := mat_costuri_roy[i, k] + mat_costuri_roy[k, j];
               mat_drum[i, j] := concat_drum(mat_drum[i, k], mat_drum[k, j]);
               end;
```

```
            writeln;
            end;
         end;


writeln;
writeln;
writeln('Afisam drumurile maxime obtinute folosind algoritmul lui Roy-Floyd :');
afis_drum_roy('max');
cauta_roy := 'max';
end;

procedure graf.dijkstra_min(x :integer);
var p, i, count  :integer;
   min :real;
   s :string;
begin

count := 0;

config_mat_costuri(p_inf);

mat_costuri_dj := mat_costuri; { duplicam matricea costurilor }

vizitate.init(n); { initializam vecotrul cu elemente vizitate }

vizitate.vizitat(x); { notam nodul de pornire x ca fiind vizitat }

init_vector_drum(x); { initializam vectorul drumrilor pentru nodul x }

while vizitate.exista_neviz() AND ( count < n) do
   begin
   min := p_inf; { pe care il stim momentan }
   p := 0;

   for i := 1 to n do { cautam nodul la care putem ajunge cel mai usor de la nodul de pornire }
      if vizitate.este_neviz(i) AND (mat_costuri_dj[x, i] < min ) then
         begin
         min := mat_costuri_dj[x ,i];
         p := i;
         end;

   vizitate.vizitat(p);
```

```
   for i := 1 to n do
      if vizitate.este_neviz(i) AND mat_adiacenta[p, i] then
         if mat_costuri_dj[x, i] > mat_costuri_dj[x, p] + mat_costuri[p, i] then
            begin
            mat_costuri_dj[x, i] := mat_costuri_dj[x, p] + mat_costuri[p, i];
            matrice_drum_dj[x, i] := p;
            end;
   count := count + 1;
   end;
writeln(' Drumurile minime de la nodul ', x, ' la restul nodurilor din graf :');
afis_drum_dj(x);
str(x, s);
cauta_dj := cauta_dj + s;
end;

procedure graf.afis_drum_roy(s :string);
var i, j, k :integer;
   aux :drum;
begin
if s = 'min' then
   begin
   for i := 1 to n do
      for j := 1 to n do
         if i <> j then
            if (mat_costuri_roy[i, j] <> p_inf) then
               begin
               writeln('Intre nodurile ', i, ' si  ', j, ' drumul are costul : ', mat_costuri_roy[i,
j]:4:2);
               writeln('Drumul este :');
               aux := mat_drum[i, j];
               write(' ':5);
               for k := 1 to aux.n do
                  write(aux.v[k], ' ');
               writeln;
               writeln;
               end
            else
               writeln('Intre nodurile ', i, ' si ', j, ' nu exista drum !');
   end
else
   begin
   for i := 1 to n do
```

```
      for j := 1 to n do
         if i <> j then
            if (mat_costuri_roy[i, j] <> m_inf) then
               begin
               writeln('Intre nodurile ', i, ' si  ', j, ' drumul are costul : ',  mat_costuri_roy[i,
j]:4:2);
               writeln('Drumul este :');
               aux := mat_drum[i, j];
               write(' ':5);
               for k := 1 to aux.n do
                  write(aux.v[k], ' ');
               writeln;
               writeln;
               end
            else
               writeln('Intre nodurile ', i, ' si ', j, ' nu exista drum !');
   end;
end;

procedure graf.afis_drum_dj(x :integer);
var i :integer;
begin
for i := 1 to n do
   if i <> x then
      if (matrice_drum_dj[x, i] <> 0) then
         begin
         writeln('Drumul de la ', x, ' la ', i ,' are costul :', mat_costuri_dj[x, i]:4:2);
         writeln('Si este compus din :');
         write(' ':7);
         afis_dr(x, i);
         writeln;
         end
      else
         writeln('Nu exista drum de la ', x, ' la ', i, ' !');
end;

procedure graf.drum_min(a, b :integer);
var i :integer;
   s :string;
begin
str(a, s);
if cauta_roy = 'min' then
   begin
```

```
    writeln('Drumul minim a fost folosit deja folosind algoritmul roy-floyd');
    writeln('Drumul minim este :');
    if mat_costuri_roy[a, b] < p_inf then
      for i := 1 to mat_drum[a, b].n do
        write(mat_drum[a, b].v[i], ' ')
    else
      writeln(' Din pacate nu se poate ajunge de la ', a, ' la ', b);
    writeln;
    end
  else
    if pos(s, cauta_dj) <> 0 then
      begin
      writeln('Drumul minim a fost folosit deja folosind algoritmul dijkstra ');
      if (mat_costuri_dj[a, b] <> 0) AND (mat_costuri_dj[a, b] < p_inf) then
        begin
        writeln('Drumul de la ', a, ' la ', b ,' are costul :', mat_costuri_dj[a, b]:4:2);
        writeln('Si este compus din :');
        write(' ':7);
        afis_dr(a, b);
        end
      else
        writeln('Nu exista drum de la ', a, ' la ', b, ' !');
      end
    else
      begin
      writeln('Drumul minim nu a fost deja calculat asa ca il vom calcula folosind aglgoritmul
dikstra ');

      dijkstra_min(a);

      writeln;
      writeln('Am terminat de calculat drumul minim !');
      writeln;

      if (mat_costuri_dj[a, b] <> 0) AND (mat_costuri_dj[a, b] < p_inf) then
        begin
        writeln('Drumul de la ', a, ' la ', b ,' are costul :', mat_costuri_dj[a, b]:4:2);
        writeln('Si este compus din :');
        write(' ':7);
        afis_dr(a, b);
        end
      else
        writeln('Nu exista drum de la ', a, ' la ', b, ' !');
```

```
        end;
writeln;
end;

procedure graf.drum_max(a, b :integer);
begin
writeln(' Cautam drumul maxim folosind algortimul metodei brute ');
bkt(a, b);
writeln;
end;

procedure graf.bkt(a, b :integer);
var i, nr :integer;
   sol, sti :vector_int;
   max :real;
   procedure initi_s(k :integer);
   begin
   sti[k] := 0;
   end;

   function solutie(k :integer):boolean;
   begin
   solutie := sti[k] = b;
   end;

   function exista(k :integer):boolean;
   begin
   exista := sti[k] < n;
   end;

   function continua(k :integer):boolean;
   var i, j :integer;
   begin
   if sti[1] <> a then
      continua := false
   else
      begin
      for i := 2 to k do
         if not mat_adiacenta[sti[i-1], sti[i]] then
            begin
            continua := false;
            break;
            end;
```

```
    for i := 1 to k-1 do
      for j := i+1 to k do
        if sti[i] = sti[j] then
          begin
          continua := false;
          break;
          end;
    end;

  procedure optimizare(k :integer);
  var i :integer;
    cost :real;
  begin
  cost := 0;
  for i := 2 to k do
    cost := cost + mat_costuri[i-1, i];

  if max < cost then
    begin
    for i := 1 to k do
      sol[i] := sti[i];
    nr := k;
    max := cost;
    end;
  end;

  procedure back(k :integer);
  begin
  initi_s(k);
  while exista(k) do
    begin
    inc(sti[k]);

    if continua(k) then
      if solutie(k) then
        optimizare(k)
      else
        back(k+1);
    end;
  end;
begin
```

```
max := 0;
nr := 0;

{ cautam solutiile  }
back(1);

writeln('Incepem cautarea drumului maxim de la ', a, ' la ', b);
writeln;
if max > 0 then
   begin
   writeln('Drumul de la ', a, ' la ', b, ' are costul :', max:4:2);
   writeln('Drumul este compus din :');
   for i := 1 to nr do
      write(sol[i], ' ');
   writeln;
   end
else
   writeln('Nu exista drum de la ', a, ' la ', b, ' !');

end;

{============================================================
============================}
{ metode ajutatoare }
function to_int(s :string):integer;
{ transformam stringul <s> in integer si il validam }
var cod, n :integer;
begin
val(s, n, cod);

if (cod = 0) and(n <= n_max) and(n <= gf.n) then
   to_int := n
else
   to_int := 0;
end;
function strip(s :string):string;
{ eliminam spatiile albe de la inceputu si finalul stringului si in intoarcem scris cu litere mici }
begin
if length(s) > 0 then
   begin
   { eliminam spatiile de la inceputul textului }
   while s[1] = ' ' do
      delete(s, 1, 1);
```

```
    { eliminam spatiile de la sfarstitul textului }
    while s[length(s)] = ' ' do
       delete(s, length(s), 1);
    end;
strip := lowercase(s);
end;

function elm_spc(s :string):string;
{ eliminam spatiile din <s> }
begin
while pos(' ',s) <> 0 do
   delete(s, pos(' ', s), 1);
elm_spc := s;
end;

function elm_space(s :string):string;
{ eliminam toate spatiile albe dintr-un string si il convertim in litere mici  }
begin
s := elm_spc(s); { eliminam spatiile }
elm_space := lowercase(s); { convertim in litera mica }
end;

procedure split_args(s :string; var a, b :string; var ok :boolean);
{ programul imparte comanda in 2 <a> si <b> in functie de paranteze, si foloseste variabila
boolean
  pentru a vedea daca parsarea s-a facut corect }
var first, last :integer;
begin
s := elm_spc(s); { eliminam spatiile din s }
ok := true;
first := pos('(', s);
last := pos(')', s);
a := '';
b := '';

if (first = 0) or (last = 0) then
   ok := false;

a := copy(s, 1, first-1);
b := copy(s, first+1, last - first - 1);

end;
```

```
procedure split_args2(s :string; var a, b :string; var ok :boolean);
{ programul imparte comanda in 2 <a> si <b> in functie de paranteze, si foloseste variabila
boolean
  pentru a vedea daca parsarea s-a facut corect }
var comma :integer;
begin
s := elm_spc(s); { eliminam spatiile din s }
ok := true;
comma := pos(',', s);
a := '';
b := '';

if comma  = 0 then
   ok := false;

a := copy(s, 1, comma-1);
b := copy(s, comma+1, length(s));
end;


function check_close(s :string):boolean;
{ verificam daca utilizatorul a folosit comanda <close> }
begin
if (elm_space(s) = 'close') or (elm_space(s) = 'exit') or (elm_space(s) = 'bye') then
   check_close := true
else
   check_close := false;
end;


function check_info(s :string):boolean;
{ verificam daca utilizatorul a tastat comanda <info> }
begin
if (elm_space(s) = 'info') then
   check_info := true
else
   check_info := false;
end;

function check_clear(s :string):boolean;
{ verificam daca utilizatorul a tastat comanda <clear> }
begin
```

```
if elm_space(s) = 'clear' then
   check_clear := true
else
   check_clear := false;
end;

function check_set_file(s :string):boolean;
{ verificam daca utilizatorul doreste sa seteze un fisier de citire }
var a, b :string;
   ok :boolean;
begin
split_args(s, a, b, ok);

check_set_file :=  ok and (elm_space(a) = 'set_file');
end;

function check_set_nume(s :string):boolean;
{ verificam daca utilizatorul doreste sa seteze un nume }
var a, b :string;
   ok :boolean;
begin
split_args(s, a, b, ok);

check_set_nume :=  ok and (elm_space(a) = 'set_nume');
end;

function check_help(s :string):boolean;
{ verificam daca utilizatorul doreste sa seteze un nume }
begin
s := elm_spc(s);

s := copy(s, 1, 4);

check_help := elm_space(s) = 'help';
end;

function check_citire_adj(s :string):boolean;
begin
s := elm_space(s);

check_citire_adj := s = 'citire_mat_adj';
end;
```

```
function check_citire_cost(s :string):boolean;
begin
s := elm_space(s);

check_citire_cost := elm_space(s) = 'citire_mat_cost';
end;

function check_parcurge_latime(s :string):boolean;
var a, b :string;
   ok :boolean;
   n :integer;
begin
split_args(s, a, b, ok);

n := to_int(b);


check_parcurge_latime := (elm_space(a) = 'parcurge_latime') and ok and (n > 0);
end;

function check_parcurge_adancime(s :string):boolean;
var a, b :string;
   ok :boolean;
   n :integer;
begin
split_args(s, a, b, ok);

n := to_int(b);

check_parcurge_adancime := (elm_space(a) = 'parcurge_adancime') and ok and (n > 0);
end;

function check_afisare_muchii(inp :string):boolean;
begin
check_afisare_muchii := elm_space(inp) = 'afisare_muchii';
end;

function check_afisare_muchii_cost(inp :string):boolean;
begin
check_afisare_muchii_cost := elm_space(inp) = 'afisare_muchii_cost';
end;

function check_afis_mat_adj(inp :string):boolean;
```

```
begin
check_afis_mat_adj := elm_space(inp) = 'afis_mat_adj';
end;

function check_afis_mat_cost(inp :string):boolean;
begin
check_afis_mat_cost := elm_space(inp) = 'afis_mat_cost';
end;


function check_roy_floyd_min(inp :string):boolean;
begin
check_roy_floyd_min := elm_space(inp) = 'roy_floyd_min';
end;

function check_roy_floyd_max(inp :string):boolean;
begin
check_roy_floyd_max := elm_space(inp) = 'roy_floyd_max';
end;


function check_dijkstra_min(inp :string):boolean;
var a, b :string;
   n :integer;
   ok :boolean;
begin
split_args(inp, a, b, ok);

n := to_int(b);

check_dijkstra_min :=  ok AND (elm_space(a) = 'dijkstra_min') AND ( n > 0 );
end;

function check_drum_min(inp :string):boolean;
var a, b :string;
   ok :boolean;
begin
split_args(inp, a, b, ok);

check_drum_min := ok AND (a = 'drum_min');
end;

function check_drum_max(inp :string):boolean;
```

```
var a, b :string;
   ok :boolean;
begin
split_args(inp, a, b, ok);

check_drum_max := ok AND (a = 'drum_max');
end;

function check_credit(inp :string):boolean;
begin
check_credit := elm_space(inp) = 'credit';
end;

function check_blank(inp :string):boolean;
begin
check_blank := elm_space(inp) = '';
end;

function check_bkt(inp :string):boolean;
begin
check_bkt := elm_space(inp) = 'bkt';
end;
{============================================================
============================}
{ Metode de afisare }
procedure help(s :string);
{ afisam help in functie de input }
var a, b :string;
   ok :boolean;
begin
s := elm_space(s);
split_args(s, a, b, ok);

writeln;
case b of
   '' :begin
      writeln('Aveti la dispozitie urmatoarele comenzi :');
      writeln('   set_nume(arg)');
      writeln('   set_file(arg)');
      writeln('   citire_mat_cost');
      writeln('   citire_mat_adj');
      writeln('   parcurge_adancime(arg)');
      writeln('   parcurge_latime(arg)');
```

```
    writeln('  afisare_muchii');
    writeln('  afisare_muchii_cost');
    writeln('  afis_mat_adj');
    writeln('  afis_mat_cost');
    writeln('  roy_floyd_max');
    writeln('  roy_floyd_min');
    writeln('  dijkstra_min(arg)');
    writeln('  drum_max(arg, arg)');
    writeln('  drum_min(arg, arg)');
    writeln('  credit');
    writeln('  bkt');
    writeln('');
    writeln('  Pentru mai multe informatii legate de fiecare comanda puteti accesa !');
    writeln('  <help(commanda)> .');
    writeln('  ');
    writeln('  Exemplu de utilizare:');
    writeln('     help(set_nume)');
    writeln('     help(drum_max)');
    writeln('     help(parcurge_latime)');
    writeln('');
    end;
  'set_nume':begin
    writeln('set_nume(arg)');
    writeln('  Cu ajutorul acestei comenzi puteti seta numele grafului.');
    writeln('  Acest nume va fi folosit daca doriti sa salvati graful. ');
    writeln('  Exemplu de utilizare:');
    writeln('     set_nume(GrafulMeu)');
    writeln('  set_nume(arg)');
    writeln('');
    end;
  'set_file':begin
    writeln('set_file(arg)');
    writeln('  Cu ajutorul acestei comenzi puteti seta un fisier sursa al');
    writeln('  grafului.Din acest fisier se va citit, matricea de adiacenta,');
    writeln('  sau matricea costurilor grafului.');
    writeln('  Exemplu de utilizare:');
    writeln('     set_file(graf.txt)');
    writeln('     set_file(costuri.in)');
    end;
  'citire_mat_cost' :begin
    writeln('  citire_mat_cost');
    writeln('  Cu ajutorul acestei comenzi puteti citi matricea costurilor din fiseirul');
    writeln('  setat cu ajutorul comenzi <set_file(arg)> !');
```

```
      writeln('  Exemplu de utilizare:');
      writeln('      citire_mat_cost');
      end;
    'citire_mat_adj' :begin
      writeln('  citire_mat_adj');
      writeln('  Cu ajutorul acestei comenzi puteti citi matricea de adiacenta din fiseirul');
      writeln('  setat cu ajutorul comenzi <set_file(arg)> !');
      writeln('  Exemplu de utilizare:');
      writeln('      citire_mat_adj');
      end;
    'parcurge_adancime':begin
      writeln('  parcurgere_adancime(arg)');
      writeln('  Cu ajutorul acestei comenzi puteti parcurge graful in adancime,' );
      writeln('  graful, pornind de la nodul specificat ca si argument .');
      writeln('  Exemplu de utilizare:');
      writeln('      parcurge_adancime(2)');
      writeln('      parcurge_adancime(33)');
      end;
    'parcurge_latime':begin
      writeln('  parcurgere_latime(arg)');
      writeln('  Cu ajutorul acestei comenzi puteti parcurge graful in latime,' );
      writeln('  graful, pornind de la nodul specificat ca si argument .');
      writeln('  Exemplu de utilizare:');
      writeln('      parcurge_latime(2)');
      writeln('      parcurge_latime(33)');
      end;
    'afisare_muchii':begin
      writeln('  afisare_muchii');
      writeln('  Cu ajutorul acestei comenzi putem afisa muchiile grafului stocat momentan .');
      writeln('  Exemplu de utilizare:');
      writeln('      afisare_muchii');
      end;
    'afisare_muchii_cost':begin
      writeln('  afisare_muchii_cost');
      writeln('  Cu ajutorul acestei comenzi putem afisa muchiile grafului stocat momentan si
');
      writeln('  costul asociat fiecarei muchii .');
      writeln('  Exemplu de utilizare:');
      writeln('      afisare_muchii');
      end;
    'afis_mat_cost':begin
      writeln('  afis_mat_cost');
```

```
      writeln('  Cu ajutorul acestei comenzi putem afisa matricea costurilor asociata grafului.
');
      writeln('  Exemplu de utilizare:');
      writeln('     afis_mat_adj');
      end;
   'afis_mat_adj':begin
      writeln('  afis_mat_adj');
      writeln('  Cu ajutorul acestei comenzi putem afisa matreia de adiacenta a graului .');
      writeln('  Exemplu de utilizare:');
      writeln('     afis_mat_adj');
      end;
   'roy_floyd_max':begin
      writeln('  roy_floyd_max');
      writeln('  Cu ajutorul acestei comenzi cautam drumurile maxiem de la orce nod din graf
la restul nodurilor');
      writeln('  cu ajutorul algoritmului lui Roy-Floyd.');
      writeln('  Exemplu de utilizare:');
      writeln('     roy_floyd_max');
      end;
   'roy_floyd_min': begin
      writeln('  roy_floyd_min');
      writeln('  Cu ajutorul acestei comenzi cautam drumurile minime de la orce nod din graf
la restul nodurilor');
      writeln('  cu ajutorul algoritmului lui Roy-Floyd.');
      writeln('  Exemplu de utilizare:');
      writeln('     roy_floyd_min');
      end;
   'dijkstra_max':begin
      writeln('  dijkstra_max(arg)');
      writeln('  Cu ajutorul acestei comenzi aflam drumurile maxime de la nodul primit ca si
argument ');
      writeln('  si restul nodurilor din graf .');
      writeln('  Exemplu de utilizare:');
      writeln('     dijkstra_max(1)');
      writeln('     dijkstra_max(4)');
      end;
   'dijkstra_min':begin
      writeln('  dijkstra_min(arg)');
      writeln('  Cu ajutorul acestei comenzi aflam drumurile minime de la nodul primit ca si
argument ');
      writeln('  si restul nodurilor din graf .');
      writeln('  Exemplu de utilizare:');
      writeln('     dijkstra_min(1)');
```

```
      writeln('      dijkstra_min(4)');
      end;
   'drum_max':begin
      writeln('   drum_max(arg, arg)');
      writeln('   Cu ajutorl acestei comenzi vom afisa drumul maxim de la nodul primit ca prim
argument ');
      writeln('   pana la nodul primit ca al doilea argument. Programul va verifica daca
cautarea nu a fost deja ');
      writeln('   facuta. Daca nu a fost facuta, va cauta drumul folosit algoritmul bkt ( metoda
bruta).');
      writeln('   Exemplu de utilizare:');
      writeln('      drum_max(1, 2);');
      writeln('      drum_max(3, 6);');
      end;
   'drum_min':begin
      writeln('   drum_min(arg, arg)');
      writeln('   Cu ajutorl acestei comenzi vom afisa drumul minin de la nodul primit ca prim
argument ');
      writeln('   pana la nodul primit ca al doilea argument. Programul va verifica daca
cautarea nu a fost deja ');
      writeln('   facuta. Daca nu a fost facuta, va cauta drumul folosit algoritmul Dijkstra .');
      writeln('   Exemplu de utilizare:');
      writeln('      drum_min(1, 2);');
      writeln('      drum_min(3, 6);');
      end;
   'bkt':begin
      writeln('   bkt');
      writeln('   Cu ajutorul acestei comenzi afisam drumurile maxime de la orce nod spre restul
nodurilor ');
      writeln('   folosind metoda bruta ( backtraking ).');
      writeln('   Exemplu de utilizare:');
      writeln('      bkt');
      end;
   'credit':begin
      writeln('   credit');
      writeln('   Aceasta comanda va afisa informatii utile despre autorul programului, utilizare,
menire si licenta ');
      end;
   end;
writeln;
end;
procedure info;
begin
```

```
writeln;
writeln('      Drumuri Minime si Maxime in Grafuri orientate      ');
writeln('   ');
writeln('   ');
writeln('   Accesati comanda <help> pentru mai multe informatii !');
writeln('   ');
writeln('   ');
writeln;
end;

procedure clear;
{ curatam ecranul }
begin
clrscr;
end;

procedure set_file(s :string);
{ setam un fisier de citire }
var a, b :string;
   ok :boolean;
begin
split_args(s, a, b, ok);

if ok then
   begin
   writeln('Am setat fisierul sursa ca fiind :', b);
   gf.set_file(b);
   end;
end;

procedure set_nume(s :string);
{ setam un numele grafului }
var a, b :string;
   ok :boolean;
begin
split_args(s, a, b, ok);

if ok then
   begin
   writeln('Am setat numele grafului ca fiind :', b);
   gf.set_nume(b);
   end;
end;
```

```
procedure citire_mat_adiacenta;
begin
writeln;

if gf.fisier <> '' then
   begin
   writeln('Citim matricea de adiacenta din fisierul : ', gf.fisier);
   gf.citire_mat_adiacenta;
   gf.citit := true;
   writeln('Citirea a fost facuta cu succes !');
   end
else
   begin
   writeln('ERROARE:');
   writeln('Nu ati selectat un fisier de unde putem citit matricea de adiacenta !');
   writeln('Folositi comanda <set_file(arg)> pentru a seta unul .');
   writeln('Pentru mai multe informatii folosit comanda <help> .');
   end;
writeln;
end;

procedure citire_mat_costuri;
begin
writeln;

if gf.fisier <> '' then
   begin
   writeln('Citim matricea costurilor din fisierul : ', gf.fisier);
   gf.citire_mat_costuri;
   gf.citit := true;
   writeln('Citirea a fost facuta cu succes !');
   end
else
   begin
   writeln('ERROARE:');
   writeln('Nu ati selectat un fisier de unde putem citit matricea costurilor !');
   writeln('Folositi comanda <set_file(arg)> pentru a seta unul .');
   writeln('Pentru mai multe informatii folosit comanda <help> .');
   end;
writeln;
end;
```

```
procedure parcurge_latime(s :string);
{ parcurgem in latime graful }
var a, b :string;
   ok :boolean;
   i :integer;
begin
split_args(s, a, b, ok); { impartim argumentele }

i := to_int(b); { transformam argumentu in integer }

writeln;
if i  > 0 then
   begin
   writeln;
   gf.parcurgere_in_latime(i);
   writeln;
   writeln('Am terminat de parcurs !');
   end
else
   begin
   writeln('ERROARE:');
   writeln(' A aparut o eroare incercand sa parcurgem graful in latime. ');
   writeln(' Folotisi comanda <help> pentru a vede mai multe informatii .');
   end;
writeln;
end;

procedure parcurge_adancime(s :string);
{ parcurgem in latime graful }
var a, b :string;
   ok :boolean;
   i :integer;
begin
split_args(s, a, b, ok); { impartim argumentele }

i := to_int(b); { transformam argumentu in integer }

writeln;
if i  > 0 then
   begin
   writeln;
   gf.parcurgere_in_adancime(i);
   writeln;
```

```
   writeln('Am terminat de parcurs !');
   end
else
   begin
   writeln('ERROARE:');
   writeln(' A aparut o eroare incercand sa parcurgem graful in adancime. ');
   writeln(' Folotisi comanda <help> pentru a vede mai multe informatii .');
   end;
writeln;
end;


procedure afisare_muchii(inp :string);
begin
if (gf.n > 0)  then
   gf.afis_lista_noduri
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosnutlati comanda <help> ')
   end;
end;

procedure afisare_muchii_cost(inp :string);
begin
if (gf.n > 0)  then
   gf.afis_lista_noduri_cost
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosnutlati comanda <help> ')
   end;
end;


procedure afis_mat_adj(inp :string);
begin
if gf.citit then
   gf.afis_mat_adiacenta
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
```

```
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure afis_mat_cost(inp :string);
begin
if gf.citit then
   gf.afis_mat_costuri
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure roy_floyd_min(inp :string);
var a, b :string;
   ok :boolean;
begin
split_args(inp, a, b, ok);


if gf.citit then
   begin
   gf.roy_floyd_min;
   writeln;
   writeln(' Am terminat de aflat drumurile minime . . . ');
   end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure roy_floyd_max(inp :string);
var a, b :string;
   ok :boolean;
begin
split_args(inp, a, b, ok);
```

```
if gf.n > 0 then
   begin
   gf.roy_floyd_max;
   writeln;
   writeln(' Am terminat de aflat drumurile maxime. . . ');
   end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure dijkstra_min(inp :string);
var a, b :string;
   ok :boolean;
   n :integer;
begin
split_args(inp, a, b, ok);

n := to_int(b);

if gf.n > 0 then
   if n > 0 then
      begin
      gf.dijkstra_min(n);
      writeln(' Am terminat de aflat drumurile minime de la nodul ', n, ' la restul nodurilor . . .
');
      end
   else
      begin
      writeln(' ERROARE:');
      writeln(' Parametrul transmis functiei este incorect, folositi comadna <help> ');
      writeln(' pentru mai multe informatii .');
      end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
```

```
end;

procedure drum_max(inp :string);
var a, b, c, d :string;
   x, y :integer;
   ok1, ok2 :boolean;
begin
split_args(inp, a, b, ok1);
split_args2(b, c, d, ok2);

x := to_int(c);
y := to_int(d);

if gf.citit then
   begin
   if ok2 AND (x > 0) AND (x <= gf.n) AND (y > 0) AND (y <= gf.n) then
      begin
      writeln('Cautam drumul maxim de la ', x, ' la ', y, ' :');
      gf.drum_max(x, y);
      end
   else
      begin
      writeln(' ERROARE:');
      writeln(' Nu ati transmis argumente corect sau sunt invalide , cosultati comanda <help>
');
      writeln(' pentru mai multe informatii .');
      end;
   end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure drum_min(inp :string);
var a, b, c, d :string;
   x, y :integer;
   ok1, ok2 :boolean;
begin
split_args(inp, a, b, ok1);
split_args2(b, c, d, ok2);
```

```
x := to_int(c);
y := to_int(d);

if gf.citit then
   begin
   if ok2 AND (x > 0) AND (x <= gf.n) AND (y > 0) AND (y <= gf.n) then
      begin
      writeln('Cautam drumul minim de la ', x, ' la ', y, ' :');
      gf.drum_min(x, y);
      end
   else
      begin
      writeln(' ERROARE:');
      writeln(' Nu ati transmis argumente corect sau sunt invalide , cosultati comanda <help>
');
      writeln(' pentru mai multe informatii .');
      end;
   end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;

procedure credit(inp :string);
begin
writeln('');
writeln('   Drumuri Minime si Maxime in grafuri Orientate ');
writeln('');
writeln(' Versiunea: 3.0.1');
writeln('');
writeln(' Autor : Micu Matei-Marius');
writeln(' Email : matei10@yahoo.com');
writeln(' Gmail : micumatei@gmail.com');
writeln(' GitHub : matei10');
writeln('');
writeln(' Licenta : MIT ');
writeln('   O copie a licente MIT ar trebui sa fie distribuita o data cu programul ');
writeln('');
writeln(' Descriere :');
```

```
writeln('   Programul aduna o serie de algoritmi intr-un singur loc pentru o utilizare mai
usoara ');
writeln('');
writeln(' Algoritmi :');
writeln('   - parcurgerea in adancime a unui graf orientat');
writeln('   - parcurgerea in latime a unui graf orientat ');
writeln('   - algoritmul Roy-Floyd');
writeln('   - algoritmul Dijkstra');
writeln('   - o serie de algoritmi auxiliari pentru cititre si afisare a matricilor de adiacenta si
cea a costurilor !');
writeln('');
writeln('');
end;

procedure bkt(inp :string);
var i, j :integer;
begin
if gf.citit then
   begin
   writeln('Cautam drumurile maxime folosind forta bruta ');
   writeln;
   for i := 1 to gf.n do
      for j := 1 to gf.n do
         if i <> j then
            begin
            writeln(' Cautam drumul intre ', i, ' si ', j, ' :');
            gf.bkt(i, j);
            end;
   end
else
   begin
   writeln(' ERROARE:');
   writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');
   writeln(' pentru mai multe informatii .');
   end;
end;


{============================================================
============================}
{ Metode de managereriere }
procedure start;
{ ascultam pentru inputul utilizatorului }
```

```
var inp :string;
   ok :boolean;
begin
write('>> ');
readln(inp);

while not check_close(inp) do
   begin

   ok := false; { presupunem ca s-a introdus o comanda gresita }

   { verificam daca utilizatorul cere informatii }
   if check_info(inp) then
      begin
      info;
      ok := true;
      end;

   { verificam daca userul doreste sa curatam ecranul }
   if check_clear(inp) then
      begin
      clear;
      ok := true;
      end;

   { verificam daca userul doreste sa seteze un fisier de citire }
   if check_set_file(inp) then
      begin
      set_file(inp);
      ok := true;
      end;

   { verificam daca userul doreste ajutor }
   if check_help(inp) then
      begin
      help(inp);
      ok := true;
      end;

   { verificam daca userul vrea sa citeasca matricea de adiacenta }
   if check_citire_adj(inp) then
      begin
      citire_mat_adiacenta;
```

```
   ok := true;
   end;

{ verificam daca userul vrea sa citeasca matricea de costuri }
if check_citire_cost(inp) then
   begin
   citire_mat_costuri;
   ok := true;
   end;

{ verificam daca userul vrea sa parcurga graful in latime }
if check_parcurge_latime(inp) then
   begin
   parcurge_latime(inp);
   ok := true;
   end;

{ verificam daca userul vrea sa parcurga graful in adancime }
if check_parcurge_adancime(inp) then
   begin
   parcurge_adancime(inp);
   ok := true;
   end;

{ verificam daca userul vrea sa afisam muchiile grafului }
if check_afisare_muchii(inp) then
   begin
   afisare_muchii(inp);
   ok := true;
   end;


{ verificam daca userul vrea sa afisam muchiile grafului si costul asociat  }
if check_afisare_muchii_cost(inp) then
   begin
   afisare_muchii_cost(inp);
   ok := true;
   end;

{ verificam daca userul doreste sa afisam matricea de adiacenta }
if check_afis_mat_adj(inp) then
   begin
   afis_mat_adj(inp);
```

```
    ok := true;
    end;

  { verificam daca userul doreste sa afisam matricea costurilor asociate }
  if check_afis_mat_cost(inp) then
    begin
    afis_mat_cost(inp);
    ok := true;
    end;

  { verificam daca userul doreste sa foloseasca algoritmul roy-floyd pentru drumuri minime }
  if check_roy_floyd_min(inp) then
    begin
    roy_floyd_min(inp);
    ok := true;
    end;

  { verificam daca userul doreste sa foloseasca algoritmul roy-floyd pentru drumuri maxime}
  if check_roy_floyd_max(inp) then
    begin
    roy_floyd_max(inp);
    ok := true;
    end;

  { verificam daca userul doreste sa afle durmurile minime de la un nod anume folosid
algoritmul dijkstra }
  if check_dijkstra_min(inp) then
    begin
    dijkstra_min(inp);
    ok := true;
    end;

  { verificam daca utilizatorul doreste sa afisam drumul minim  dintre doua noduri }
  if check_drum_min(inp) then
    begin
    drum_min(inp);
    ok := true;
    end;

  { verificam daca utilizatorul doreste sa afisam drumul maxi dintre doua noduri }
  if check_drum_max(inp) then
    begin
    drum_max(inp);
```

```
      ok := true;
      end;

   { daca se doreste afisarea creditelor }
   if check_credit(inp) then
      begin
      credit(inp);
      ok := true;
      end;

   { daca s-a introdus o linie goala }
   if check_blank(inp) then
      begin
      ok := true;
      end;

   { verificam daca utilizatorul doreste sa cautam drumurime maxime de la oricare doua
noduri ale grafului }
   if check_bkt(inp) then
      begin
      bkt(inp);
      ok := true;
      end;

   { daca nu s-a introdus o linie corecta }
   if not ok then
      begin
      writeln;
      writeln(' Comanca introdusa nu a fost recunoscuta ');
      writeln(' Folositi comanda <help> pentru mai multe informatii ');
      writeln;
      end;

   write('>> ');
   readln(inp);
   end;
end;

{============================================================
============================}

{ Program Principal }
begin
```

```
gf.init; { initializam obiectul graf }

info; { display info }


start; { ascultam pentru mesajele utilizatorilor }
writeln;
end.
```

Cuprins

Aspecte teoretice 1
B. Metode de reprezentare a grafului orientat 2
C. Noţiunea de graf parţial 3
D. Noţiunea de subgraf 5
E. Gradul unui nod 7
G. Metode de parcurgere a grafurilor orientate 13
H. Drumuri minime şi maxime 25
Algoritmul Roy-Floyd 28
Algoritmul Dijkstra 35
Metodă Brută 46
Program auxiliar 50
I. Program Complex 52

Bibliografie

Informatica (Manual pentru clasa a XI-a) -
Mioara Gheorghe, Monica Tătărâm, Corina Achinca, Constanţa Năstase

Pascal Teorie Şi Aplicaţii (clasa a XI-a) -
Cristian Udrea, Claudia Elena Udrea, Dan Cristian Ţacu, Diana Nicoleta Udrea

Grafice realizate folosind www.draw.io

Siteuri utilizare:

- **http://stackoverflow.com/**
- **https://github.com/**
- **www.draw.io**
- **https://docs.google.com/**

Toata lucrarea şi programele pot fi găsite pe :

- **https://github.com/matei10/atestat-2015**

Autor : Micu Matei-Marius
Email : matei10@yahoo.com
Gmail : micumatei@gmail.com
GitHub : https://github.com/matei10
StackOverflow : http://stackoverflow.com/users/4311994/matei