

Drumuri Minime și Maxime în grafuri Orientate

A. Aspecte teoretice

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

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

U : este o mulțime 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 definiția prezentată mai sus, adică :

V este **finită** și **nevidă** ;

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

Observație !

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

Terminologie fregventă în teoria grafurilor:

- **extremitățile unui arc**
 - fiind dat un arc $u = (x, y)$, se numesc extremități ale sale **nodurile x și y** :
 - x se numește **extremitate initială**;
 - y se numește **extremitate finală**.
- **vârfuri adiacente**
 - dacă într-un graf există arcul $u = (x, y)$ (sau (y, x) , sau amândoua), se spune despre **nodurile x și y** ca sunt **adiacente**.
- **incidenta**
 - dacă u_1 și u_2 sunt două arce ale aceluiaș graf, se numesc **incidente** dacă **au o extremitate comună**.

Exemplu :

$u_1 = (x, y)$ și $u_2 = (y, z)$ sunt incidente

- dacă $u1 = (x, y)$ este un arc într-un graf, se spune despre el și **nodul x**, sau **nodul y**, că sunt incidente.

B. Metode de reprezentare a grafului orientat

Sunt admise două metode de reprezentare :

- ❑ **reprezentare textuală** : așa cum au fost prezentate până acum
- ❑ **reprezentare grafică** : **arcele** sunt reprezentate prin **săgeți orientate** iar **nodurile** prin **puncte**.

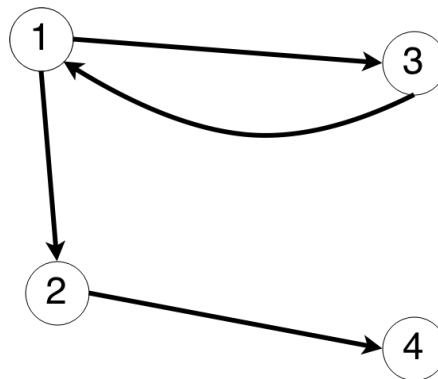
Exemplu de graf orientat reprezentat **textual**:

$G = (V, U)$ 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:

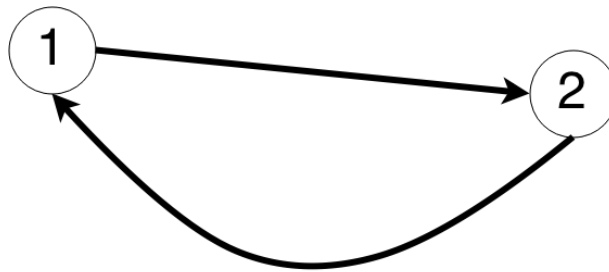
Considerând graful :

$G = (V, U)$ 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 direcții, adică în felul următor :



C. Noțiunea de graf parțial

Definiție. Fie $G=(V, U)$ un **graf orientat**. Se numește **graf parțial** al grafului G **graful orientat** $G1 = (V, U1)$ unde $U1 \subseteq U$.

Citind cu atenție definiția tragem concluzia ca un **graf parțial** al unui graf G este un graf care are aceeași mulțime de vârfuri ca și G , iar mulțimea arcelor este o submulțime a lui U sau chiar U .

Exemplu :

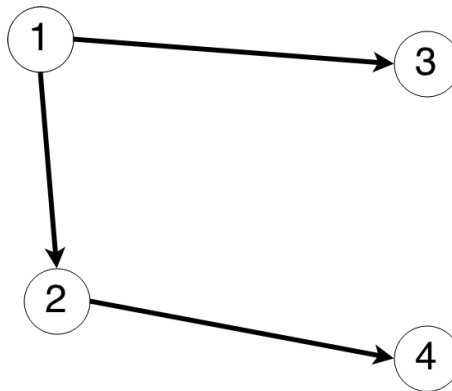
Fie graful orientat :

$$G = (V, U)$$

$$V = \{1, 2, 3, 4\}$$

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

reprezentat **grafic** astfel :



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

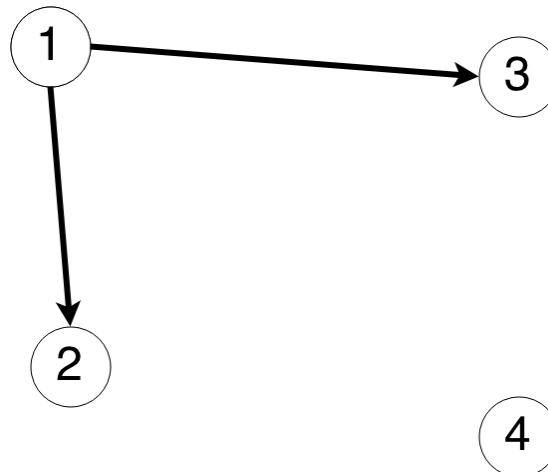
$$G = (V, U)$$

$$V = \{1, 2, 3, 4\}$$

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

(s-a eliminat arcul (2, 4))

reprezentat **grafic** astfel :



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

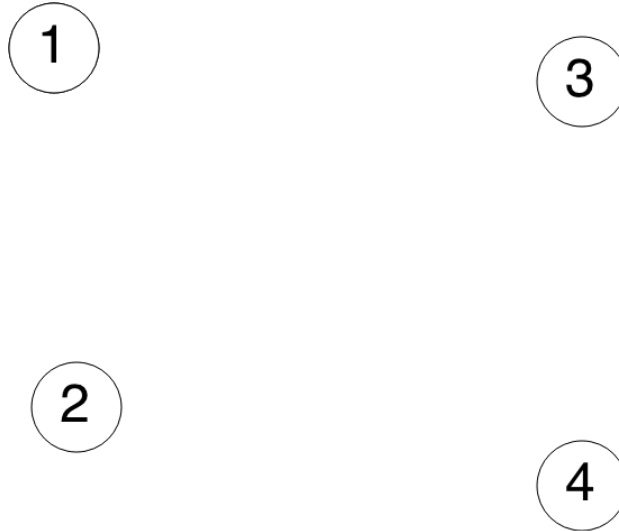
$$G = (V, U)$$

$$V = \{1, 2, 3, 4\}$$

$$U = \emptyset$$

(s-au eliminat toate arcele)

reprezentat **grafic** astfel :



Observatie !

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

D. Noțiunea de subgraf

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

Exemplu :

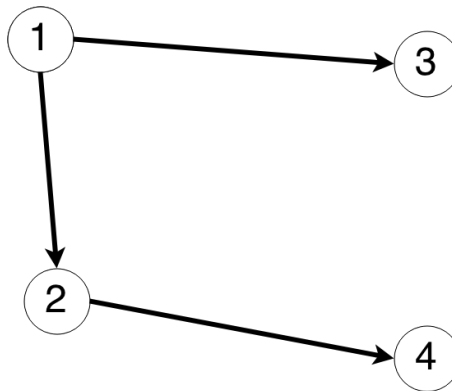
Fie graful orientat :

$$G = (V, U)$$

$$V = \{1, 2, 3, 4\}$$

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

reprezentat **grafic** astfel :



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

$$G = (V, U)$$

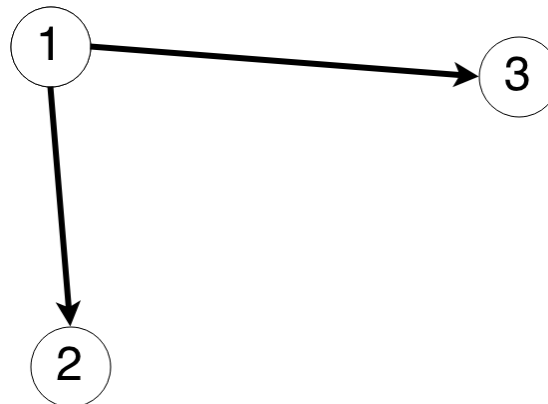
$$V = \{1, 2, 3\}$$

(s-a eliminat nodul 4)

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

(s-a eliminat arcul (2, 4))

reprezentat **grafic** astfel :



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

$$G = (V, U)$$

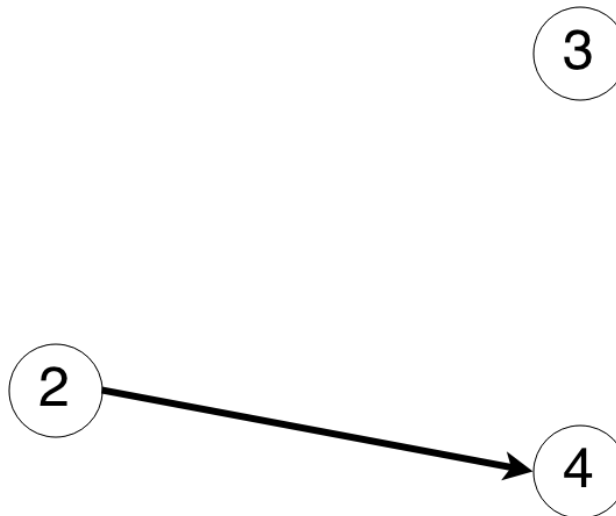
$$V = \{2, 3, 4\}$$

(s-a eliminat nodul 1)

$$U = \{(2, 4)\}$$

(s-au eliminat arcele (1,3) și (1,2))

reprezentat **grafic** astfel :



Observație !

Fie $G = (V, U)$ un graf orientat. Un subgraf, al grafului G , se poate obține ștergând eventual anumite noduri și odată cu acestea și arcele care le admit ca extremitate (nu se pot șterge toate vârfurile deoarece s-ar obține un graf cu mulțimea varfurilor vida)

E. Gradul unui nod

Ținând cont de faptul că “raportat la un vârf există arce care ies din acel vârf și arce care intră în acel vârf”, iau naștere următoarele noțiuni :

- grad interior
- grad exterior

Definiție. Fie $G=(V, U)$ un graf orientat și x un nod al său. Se numește **grad exterior** al nodului x , numărul arcelor de forma (x, y) (adică numărul arcelor care ies din x), notat $d^+(x)$.

Exemplu :

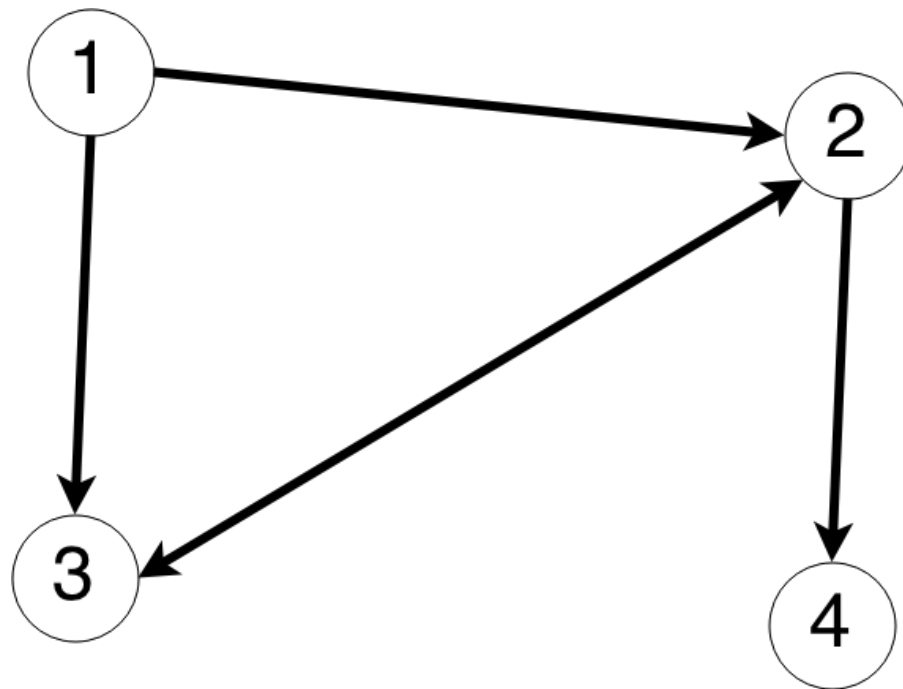
Fie grafiul :

$G = (V, U)$ 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) **Mulțimea succesorilor nodului x** se notează cu $\Gamma^+(x)$ și se reprezintă astfel :

$$\Gamma^+(x) = \{ y \in V \mid (x, y) \in U \}$$

2) **Mulțimea arcelor ce ies din nodul x** se notează cu $\omega^+(x)$ și se reprezintă astfel :

$$\omega^+(x) = \{ (x, y) \in U \mid y \in V \}$$

3) **Legat de cardinalele mulțimilor $\Gamma^+(x)$ și $\omega^+(x)$ putem scrie :**

$$|\Gamma^+(x)| = |\omega^+(x)| = d^+(x)$$

Raportat la același graf putem scrie următoarele:

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

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

Definiție. Fie $G=(V, U)$ un **graf orientat** și x un nod al său. Se numește **grad interior al nodului x** , numărul arcelor de forma (y, x) (adică numărul arcelor care intra din x), notat $d^-(x)$.

Exemplu :

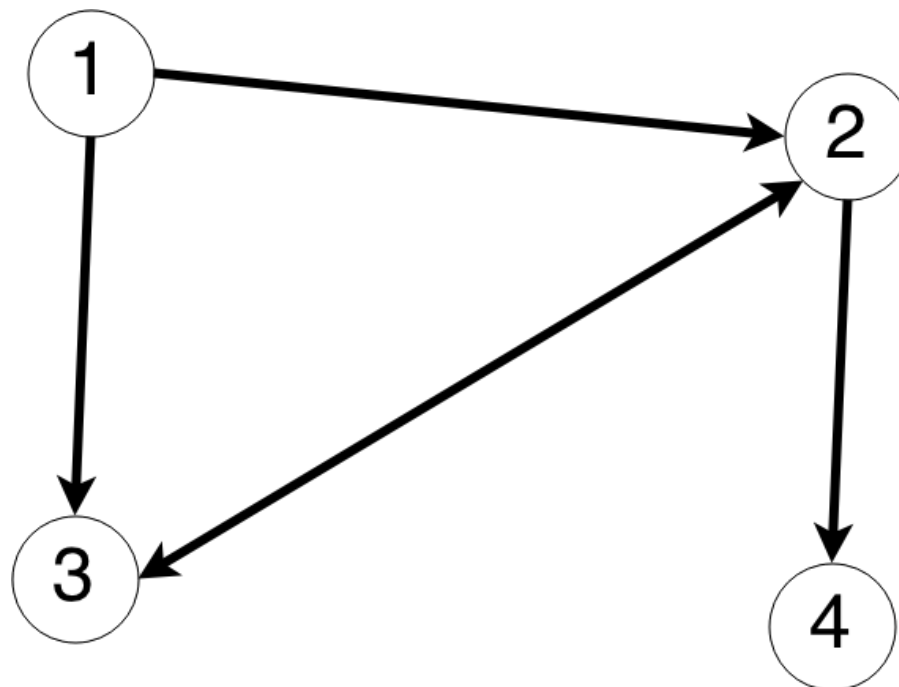
Fie graful :

$G = (V, U)$ 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) **Mulțimea predecesorilor nodului x** se notează cu $\Gamma^-(x)$ și se reprezintă astfel :

$$\Gamma^-(x) = \{ y \in V \mid (y, x) \in U \}$$

2) **Mulțimea arcelor ce intra din nodul x** se notează cu $\omega^-(x)$ și se reprezintă astfel :

$$\omega^-(x) = \{ (y, x) \in U \mid y \in V \}$$

3) **Legat de cardinalele mulțimilor $\Gamma^-(x)$ și $\omega^-(x)$ putem scrie :**

$$|\Gamma^-(x)| = |\omega^-(x)| = d^-(x)$$

Raportat la același graf putem scrie următoarele:

$\Gamma^-(1) = \emptyset$	$\omega^-(1) = \emptyset$	$ \Gamma^-(1) = \omega^-(1) = d^-(1) = 0$
$\Gamma^-(2) = \{1, 3\}$	$\omega^-(2) = \{(1, 2); (3, 2)\}$	$ \Gamma^-(2) = \omega^-(2) = d^-(2) = 2$
$\Gamma^-(3) = \{1, 2\}$	$\omega^-(3) = \{(1, 3); (2, 3)\}$	$ \Gamma^-(3) = \omega^-(3) = d^-(3) = 2$
$\Gamma^-(4) = \{2\}$	$\omega^-(4) = \{2, 4\}$	$ \Gamma^-(4) = \omega^-(4) = d^-(4) = 1$

F. Conexitate

În această secțiune vor fi prezentate următoarele noțiuni :

- Drum
- Drum elementar
- Drum neelementar

Drum

Definiție.

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

Se întâlnesc noțiunile:

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

extremități ale sale nodurile x_1 și x_k (x_1 - extremitate inițială; x_k - extremitate finală);

- **lungimea drumului**

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

Exemplu:

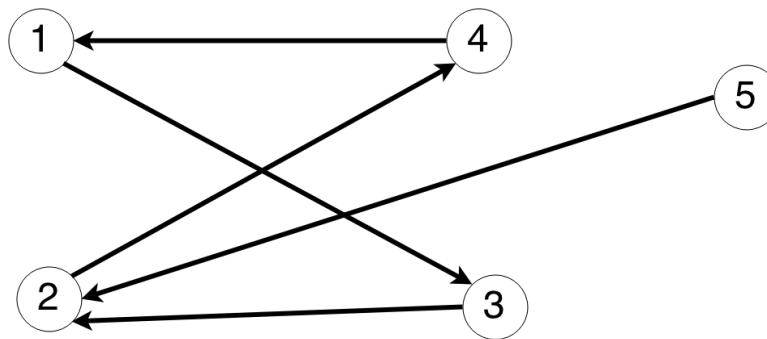
Fie graful :

$G = (V, U)$ 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** și extremitățile **1** și **2**.

$D2 = (4, 1, 3, 2)$ este, în graful G , **drum** cu lungimea **3** și extremitățile **4** și **2**.

Atenție : dacă $D = (x_1, x_2, x_3, \dots, x_k)$ este drum în graful G , atunci nu neapărat și

$D1 = (x_k, x_{k-1}, x_{k-2}, \dots, x_1)$ este drum în graful G .

Drum elementar

Definiție.

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

Exemplu:

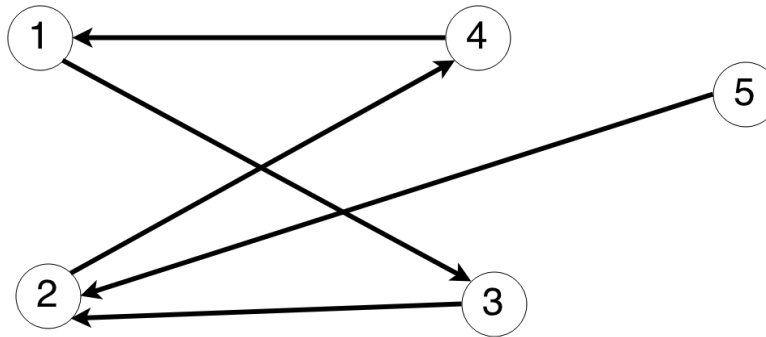
Fie graful :

$G = (V, U)$ 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 și extremitățile 4 și 2 .

Drum neelementar

Definiție.

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

Exemplu:

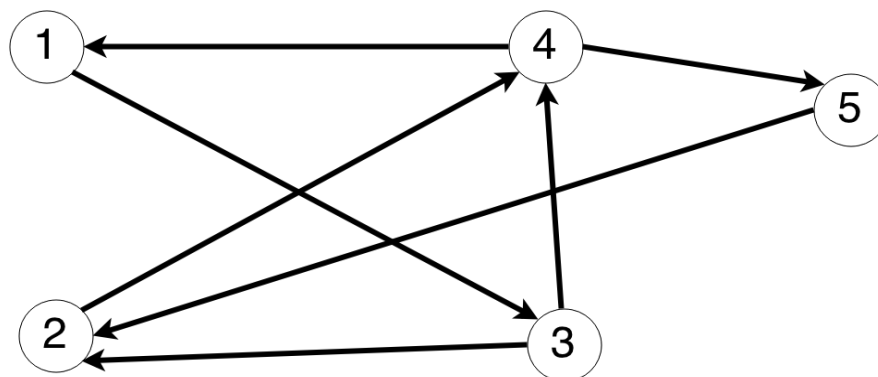
Fie graful :

$G = (V, U)$ 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** și extremitățile **4** și **2**.

G. Metode de stocare digitală a grafurilor orientate

Pentru a putea prelucra un graf orientat cu ajutorul unui program, trebuie mai întâi să fie reprezentat în program.

Există mai multe modalități de a reprezenta un graf orientat într-un program:

- reprezentare prin **matrice de adiacență**;
- reprezentare prin matrice **vârfuri-arce**;
- reprezentare prin **matrice a drumurilor**;
- reprezentare prin **lista de adiacență**;
- reprezentare prin **șirul arcelor** ;

În continuare vom prezenta cum se poate reprezenta un graf orientat prin **matricea de adiacență**.

Pentru a reprezenta un graf în memoria unui program o să ne folosim următoarele proprietăți 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}**) și **m** arce .

Definiție

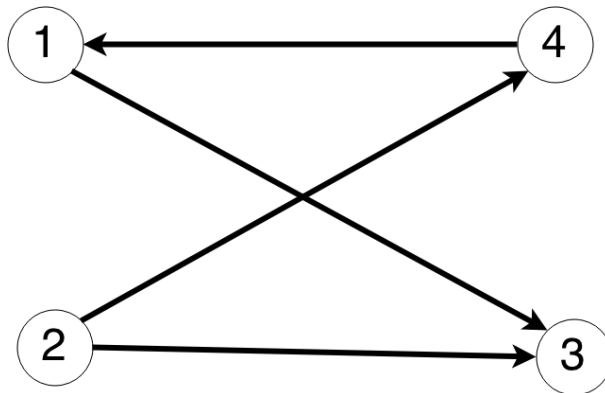
Matricea de adiacență ($A \in M_n(\{0, 1\})$), asociată grafului **G**, este o **matrice pătratică de ordin n**, cu elementele:

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

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

Exemplu 1 :

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

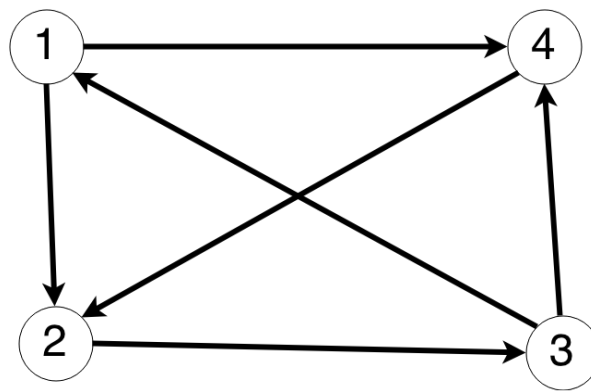


Matricea de adiacență asociată grafului este :

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

Exemplu 2 :

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



Matricea de adiacență 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 adiacență este o matrice **pătratică**, de ordin n , și **nu este neapărat simetrică** față de diagonala principală, așa cum este cazul grafurilor neorientate.

Secvențe de citire a matricei de adiacență:

```

program prprogram_1;
var a :array[1..100, 1..100] of 0..1;
  
```

```

    i, j, n :byte;

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

{ initializam matricea 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 prprogram_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 }

{ initializam matricea 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) }
    end;

```

```

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

```

Comentarii :

- **Matricea de adiacență** 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** și pe coloana **i** nu sunt elemente egale cu 1.
 - *function vf_izolat(i :byte):boolean;*
{ subprogramu returneaza TRUE dacă nodul <i> este izolat și
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 urmărește 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, să fie marcat o singură dată.

Există două metode principale de parcurgere a grafurilor :

- **în lățime**
- **în adâncime**

Metoda parcurgeri în lățime

Această metodă folosește o structură de tip coadă. Se pornește 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 și 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 }  
  
    { initializam matricea 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_coadă;
{ initializam coada }
begin
    n_c := -1;
    i_c := 0;
end;

```

```

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

```

```

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

```

```

function pop_coadă : byte;
{ returnam primul element din coada și îl eliminăm }
{ acest subprogram se utilizează doar când coada nu }
{ este vidă }
begin

```

```

pop_coadă := c[i_c]; { returnăm elementul }
inc(i_c); { eliminăm elementul }
end;

```

```

procedure init_viz;
{ initializăm vectorul cu nodurile vizitate cu FALSE, nici
  un nod nu a fost vizitat }
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;
{ returnează TRUE dacă nodul a fost vizitat, în caz contrar
  returnează FALSE }
begin
a_fost_vizitat := viz[i];
end;

```

```

procedure prelucrare_nod(x : byte);
{ prelucram nodul, în cazul de față o să îl afișăm doar }
begin
writeln('Prelucram nodul : ', x);
end;

```

```

begin
init_coadă; { initializăm coada }

```

```

citire_matrice_adiacenta; { citim toate elementele grafului }

```

```

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

```

```

push_coadă(x); { adaugăm <x> în coadă }

while not este_vidă_coadă() do { cât timp mai avem noduri de prelucrat în coadă }
begin

    p := pop_coadă; { scoatem un element din coadă }

    prelucrare_nod(p); { prelucram nodul }

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

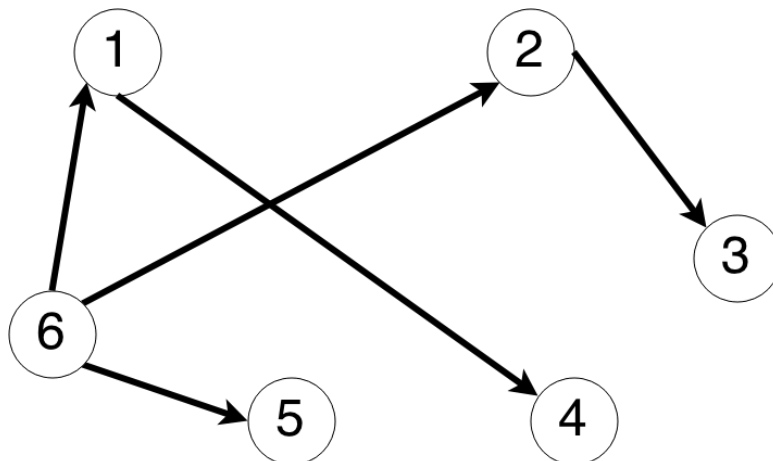
    { cautăm toate nodurile la care putem ajunge prin nodul x }
    for i := 1 to n do
        if (a[p, i] = 1) AND not a_fost_vizitat(i) then { dacă există arc de la <p> la
                                                         <i> și dacă nu am vizitat nodul <i> }
            push_coadă(i); { adăuga nodul în coadă }

    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 următoare :

1, 4

Observatie :

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

Metoda parcurgeri în adâncime

Această metodă folosește o structură de tip stivă. Se pornește de la nodul x se găsește 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 nevizitați, coborâm în stivă și 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 }
```

```
{ initializam matricea 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, și 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 vizitat, în caz contrar  
returneaza FALSE }  
begin  
a_fost_vizitat := viz[i];  
end;
```

```
procedure prelucrare_nod(x : byte);  
{ prelucram nodul, în cazul de față o să îl afișăm doar }  
begin  
writeln('Prelucram nodul : ', x);  
end;
```

```
begin  
init_stiva; { initializăm stiva }
```

```
citire_matrice_adiacenta; { citim toate elementele grafului }
```

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

```
push_stiva(x); { adăugăm <x> în stivă }
```

```
while not este_vida_stiva() do { cât timp mai avem noduri de prelucrat în stivă }  
begin
```

```
  p := acces_stiva; { accesăm elementul din vârful stivei }
```

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

```

    prelucrare_nod(p); { prelucreaza }
    vizitat(p); { noteaza 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;

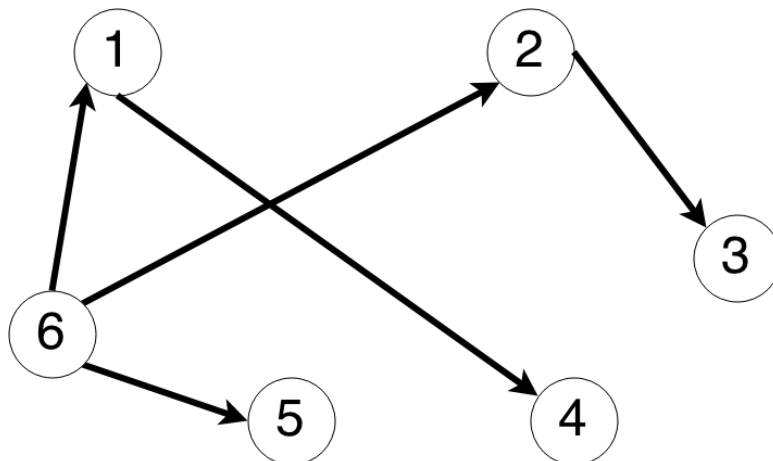
```

```

    writeln;
end.

```

Dacă avem graful reprezentat grafic astfel :



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

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 :

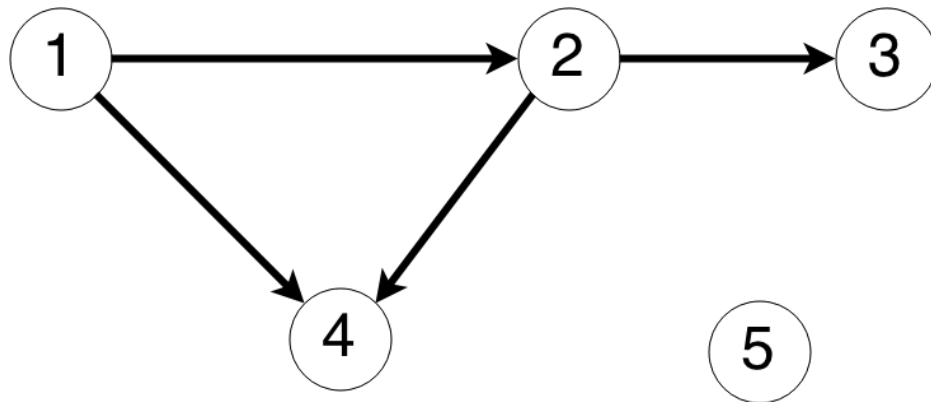
Aceste metode nu parcurg tot graful, dacă avem de exemplu

$$G = (V, U)$$

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

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

Reprezentat grafic astfel :



Dacă îl parcurgem în lățime 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 $x \in V$ a.î. $(x, 5) \in U$.

H. Drumuri minime și maxime

În această categorie vom trata problemele următoare:

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

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

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

- Se poate cere determinarea **drumului de lungime minimă (maximă)** **dintre un nod i și un nod j în graful $G = (V, U)$ știinduse matricea cu costuri asociată grafului $C \in M_n(\mathbb{R})$**
- Se poate cere determinarea **drumului de lungime minimă(maximă)** **dintre un nod i și restul nodurilor dintr-un graf $G = (V, U)$ știinduse matricea cu costuri asociată grafului $C \in M_n(\mathbb{R})$**

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

$C \in M_n(\mathbb{R})$, unde :

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

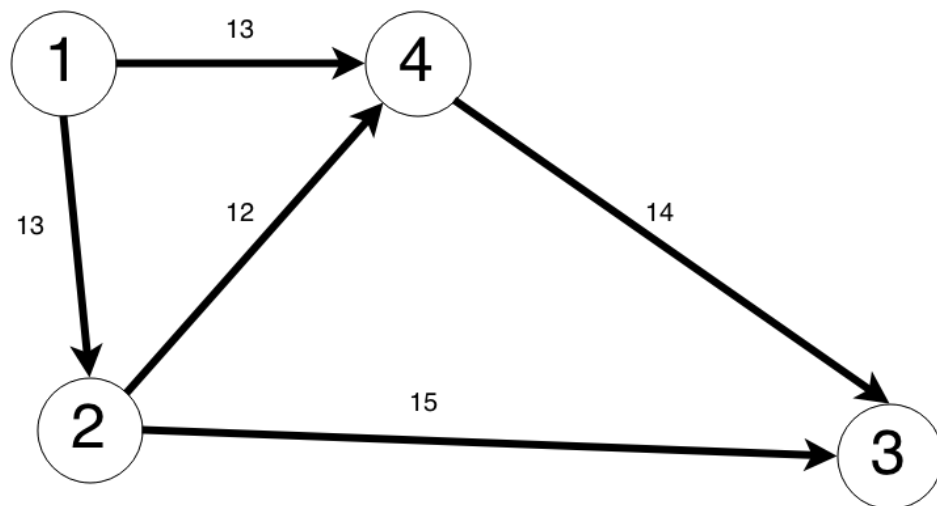
În cazul **problemelor de minimmaxim** , fiind dat graful $G = (V, U)$ i se asociază **matricea costurilor, forma 2**, definită astfel :

$C \in M_n(\mathbb{R})$, unde :

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

Exemplu :

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



Matricea costurilor de adiacență 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:

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

Observații :

1. Matricea costurilor **forma 1** diferă de matricea costurilor **forma 2** prin faptul că în loc de ∞ are $-\infty$.
2. În program, nu se poate folosi conceptul de ∞ sau $-\infty$, de aceea o să 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**

Algoritmul Roy-Floyd

Acest algoritm se aplică în cazul în care se dă un graf $G = (V, U)$, care are matricea costurilor C , și se cere să se determine **lungimea drumurilor minime(maxime)**, și în unele cazuri și **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 și drumul de la nodul i la k , precum și de la nodul k la j , este minim”

și constă defapt într-un șir de n transformări aplicate matricei costurilor C , astfel :

În esență, vom vedea dacă costul unui drum $i - j$ este mai mic decât suma costurilor drumurilor $i - k$ și $k - j$, dacă $i - j < i - k + k - j$, știm că drumul minim de la i la j trece prin k .

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

$$d_{i,j} = \text{drumul de la } i \text{ la } j$$

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

Matricea drumurilor se inițializează în felul următor :

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

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

Transformările sunt următoarele :

- dacă $c_{i,j} < c_{i,k} + c_{k,j}$ atunci $c_{i,j}$ devine $c_{i,k} + c_{k,j}$ și $d_{i,j} = \text{concat}(d_{i,k}, d_{k,j})$
- dacă $c_{i,j} > c_{i,k} + c_{k,j}$ atunci $c_{i,j}$ rămâne același, și $d_{i,j}$ rămâne același

În exemplu o să citim matricea costurilor dintr-un fișier *costuri.txt* și vom codifica $-\infty$ cu -1 , pentru a simplifica lucrurile, fișierul va avea pe prima linie numărul de noduri (linii și coloane a matricii) și pe următoarele linii matricea costurilor.

Exemplu a conținutului unui fișier :

```
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** :

Drumuri minime și maxime în grafuri orientate

```

program algoritmul_roy_floyd;
uses crt;
const p_infin = 1.e10;
      m_infin = -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 altul }
    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 și ce muchii exista în graf
  citirea se face dintr-un fisier 'costuri.txt' care are pe prima linie
  numărul 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_infin;
          end;
        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_infinity) 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;
  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;
    end;
  end;
end;

```

```
concat_drum := aux;
end;
```

```
procedure afis;
{ afisam drumurile Sicomurie }
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_infinif) then { dac  exista drum de la i -> j }
begin
{ afisam costul  i 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;
end;
```

```
procedure roy_floyd;
{ algoritmul principal }
var i, j, k :integer;
begin
for i := 1 to n do { pentru fiecare nod  n graf }
for j := 1 to n do { pentru pereche de noduri i, j }
for k := 1 to n do { verificam pentru fiecare nod  n graf }
if (k <> i) AND ( k <> j) then { dac  nodul k este diferit de nodurile i  i j }
if ( costuri[i, j] > costuri[i, k] + costuri[k, j]) then {  i dac  drumul i - k - j este mai
scurt decat drumul i - j }
begin
costuri[i, j] := costuri[i, k] + costuri[k, j]; { modific  costul cu noul cost, mai mic}
drumuri[i, j] := concat_drum(drumuri[i, k], drumuri[k, j]); { alipeste drumurile i -
k  i k - j  i salveazale  n locul drumului i - j }
end;
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 cauta drumul maxim dintre toate nodurile trebuie făcute niște modificări minore algoritmului:

- În timpul citirii matricii costurilor trebuie să inițializăm matricea costurilor cu $-\infty$ (*m_infin*) în loc de $+\infty$ (*p_infin*)

```

.....
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_infin;
end;
writeln;
end;
.....

```

- Mai trebuie modificată și comparația făcută între costul actual al drumului **i - j** și suma costurilor drumului **i - k** și **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_infin )

```


AND (costuri[k, j] > m_infini) then { dacă nodul k este diferit de nodurile i

si j, și dacă nu avem drumuri inexistente, adică

costuri = m_infini }

if (costuri[i, j] < costuri[i, k] + costuri[k, j]) then { și 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 și k - j și salvează-le în locul drumului i - j }

end;

.....

- Și inițializarea matricii drumurilor trebuie modificată

.....

if (i <> j) AND (costuri[i, j] > m_infini) 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 afișare

.....

if i <> j then { un drum de la un nod la el însuși nu are sens }

if (costuri[i, j] < m_infini) then { dacă exista drum de la i -> j }

begin

{ afișm costul și drumul cel mai scurt }

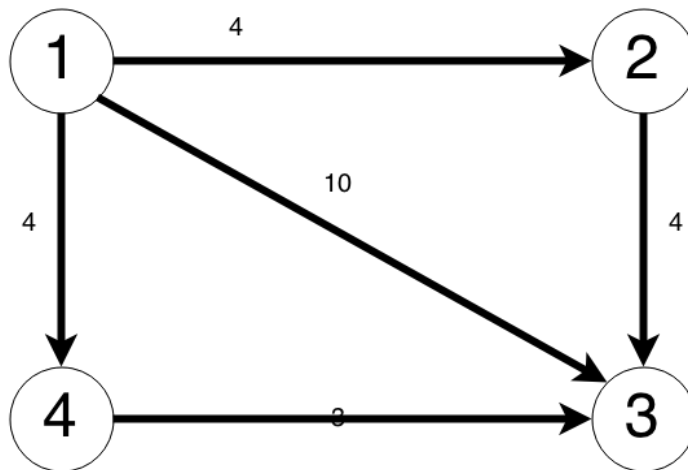
writeln('Intre nodul ', i, ', ', j, ' are costul ', costuri[i, j]:4:2);

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 !

Bineînțeles 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 , și se cere să se determine **lungimea drumurilor minime(maxime)**, și în unele cazuri și **nodurile care constituie drumurile respective** , între un nod și oricare alt nod al grafului.

Programul folosește următoarele variabile :

- n : reprezintă **numărul de noduri** al grafului
- c : reprezintă matricea costurilor asociată grafului
- $pornire$: reprezintă **nodul de plecare**
- $costuri_drumuri$: vector care conține 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 ține evidența nodurilor care au fost vizitate

Algoritmul procedează astfel :

- se citește matricea costurilor
- vectorul $vizitate$ se initializează cu *false*
- se citește nodul de pornire și se notează $vizitate[pornire] = true$
- se completează vectorul $drum$ astfel :
 - pentru $i = 1 \dots n$, $drum[i] = pornire$, $c[pornire, i] \neq \infty$ și $i \neq pornire$;
 - $drum[i] = 0$, altfel;
- se completează vectorul $costuri_drumuri$ astfel :
 - $costuri_drumuri[i] = c[pornire, i]$, pentru $i = 1 \dots n$ dacă $i \neq pornire$
 - $costuri_drumuri[pornire] = 0$
- cât timp există noduri nevizitate executăm următoarele:
 - se caută nodul k dintre cele nevizitate, a carui drum de la $pornire$ la k este minim, adică $costuri_drumuri[k]$ minim și $vizitate[k] = false$

- 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 știm momentan de la *pornire* la i) și $costuri_drumuri[k]$ (costul pe care îl știm 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 și apoi de la k la i
 - dacă costul de la *pornire* la i este cel mai mic, atunci nimic nu se întâmplă
 - dacă costul de la *pornire* la k este cel mai mic, atunci $costuri_drumuri[i] = costuri_drumuri[k] + c[k, i]$ și $drum[i] = k$ (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_infin = 1.e10;
      m_infin = -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 și <-1> }
          c[i, j] := p_infinif;
        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 caz contrar }
begin
este_neviz := not vizitate[x];
end;

```

```

procedure init_drum_costuri;
{ initializam 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 și costul este cher
costul arcului }
  else
    costuri_drum[i] := 0; { nu exista arc, deci initializam cu 0 }
end;

```

```

procedure init_drum;
{ initializam 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;

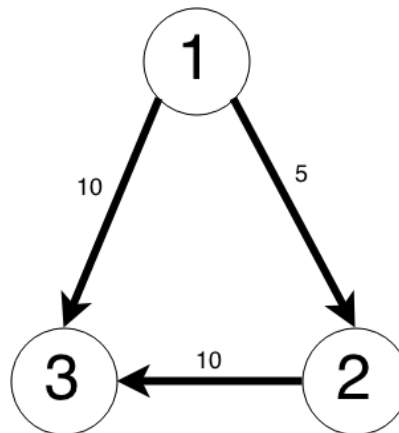
    dijkstra;
    afis;
end.

```

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
 Și este compus din
 1 2
 Drumul de la 1 la 3 are lungimea :5.00

Și este compus din

1 3

Nu se cunoste înca un algoritm eficient pentru căutarea drumurilor maxime !

Dar o implementare ar fi :

```
program algoritmul_dijkstra_minim;
uses crt;

const p_infin = 1.e10;
      m_infin = -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_infnit;
    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;
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;
{ initializam 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 chiar costul arcului }
    else
        costuri_drum[i] := 0; { nu exista arc, deci initializam cu 0 }

end;

procedure init_drum;
{ initializam 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_infini) then { daca exista arc de la
nodu de pornire la i }
        drum[i] := pornire { fiind doar un arc, extremitatea initiala e chiar 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_infini;
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_infini) then { daca exista arc de la k la i
}

    if (costuri_drum[i] > costuri_drum[k] + c[k, i]) OR (costuri_drum[i] =
p_infini) 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;
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;
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_infinity) AND (c[i, j] > 0) then
            c[i, j] := 1 / c[i, j];
        end;
    end;
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.

```

Program auxiliar :

Deoarece programele din această lucrare citesc matricea de adiacență și cea a costurilor dintr-un fișier, programul următor va citi arcele și costul acestora și va genera și salva într-un fișier 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 toți algoritmi prezentați până 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 elemente 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); { initializam 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> }
    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;
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;
        end;
    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;
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;
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;
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);
{ initializam toate componentele in functie de n }
var i, j :integer;
begin
  n := x;

  { initializam 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);

  { initializam 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_adiacentă[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;
        end;
    end;

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

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

    readln(f, x);

    { initializam elementele matrici in functie 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_adiacentă[i, j] := true;
                    end;
                end;
            end;
        end;
    end;

```

```

    readln(f);
    end;
close(f);

{ initiazam 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_adiacentia[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_adiacentia[i, j] then
                writeln(i, ' - ', j);
        end;
    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_adiacentă[p, i] AND vizitate.este_neviz(i) then
                    coada.push(i);
                end;
            end;

        writeln;
    end;

```

```

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('Li 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 gastit 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_adiacentă;
{ afisam matricea de adiacenta }
var i, j :integer;
begin
    for i := 1 to n do
        begin
            for j := 1 to n do
                if mat_adiacentă[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;

```

```

    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;
        end;
    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;
            end;
        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 costurilor !');
          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 costurilor !');
                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;
      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 drumurilor 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;
                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], ' ')
                end
            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
            end
        else

```

```

begin
  writeln('Drumul minim nu a fost deja calculat asa ca il vom calcula folosind algoritmul
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);
var i, k :integer;
    s :string;
    aux :drum;
begin
  str(a, s);
  if cauta_roy = 'max' then
    begin
      writeln('Drumul maxim a fost folosit deja folosind algoritmul roy-floyd');
      writeln('Drumul maxim 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
      begin

```

```
writeln('Drumul maxim nu a fost deja calculat asa ca il vom calcula folosind algoritmul
roy-floyd ');
```

```
roy_floyd_max;
```

```
writeln;
writeln('Am terminat de calculat drumul roy-floyd!');
writeln;
```

```
if (mat_costuri_roy[a, b] <> m_inf) AND (mat_costuri_roy[a, b] <> p_inf) then
begin
writeln('Intre nodurile ', a, ' si ', b, ' drumul are costul : ', mat_costuri_roy[a, b]:4:2);
writeln('Drumul este :');
aux := mat_drum[a, b];
write(' ':5);
for k := 1 to aux.n do
write(aux.v[k], ' ');
writeln;
writeln;
end
else
writeln('Intre nodurile ', a, ' si ', b, ' nu exista drum !');
end;
writeln;
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;
{=====
=====}
{ 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("");
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 citi, 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 ajutorul 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 roy-floyd. ');
    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 ajutorul 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;
'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;
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;
end;

```

```

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

```

```

procedure afis_mat_cost(inp :string);
begin
  if gf.citit then
    gf.afis_mat_costuri
  else
    begin
      writeln(' ERROARE:');
    end;
  end;
end;

```

```

    writeln(' Nu avem nici un graf citit, consultati 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, consultati comanda <help> ');
            writeln(' pentru mai multe informatii . ');
        end;
    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, consultati 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;

    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
      end
    end
  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 citire si afisare a matricilor de adiacenta si
cea a costurilor !');
    writeln("");

```

```

writeln("");
end;

{=====
=====}
{ Metode de manageriere }
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 folosind
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;

{ 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. Noțiunea de graf parțial	3
D. Noțiunea de subgraf	5
E. Gradul unui nod	7
G. Metode de parcurgere a grafurilor orientate	17
H. Drumuri minime și maxime	25
Algoritmul Roy-Floyd	27
Algoritmul Dijkstra	35
Program auxiliar	41
I. Program Complex	43

Bibliografie

Informatica (Manual pentru clasa a XI-a) -

Mioara Gheorghe, Monica Tătărâm, Corina Achinca, Constanța Năstase

Pascal Teorie Și Aplicații (clasa a XI-a) -

Cristian Udrea, Claudia Elena Udrea, Dan Cristian Țacu, 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 și 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>