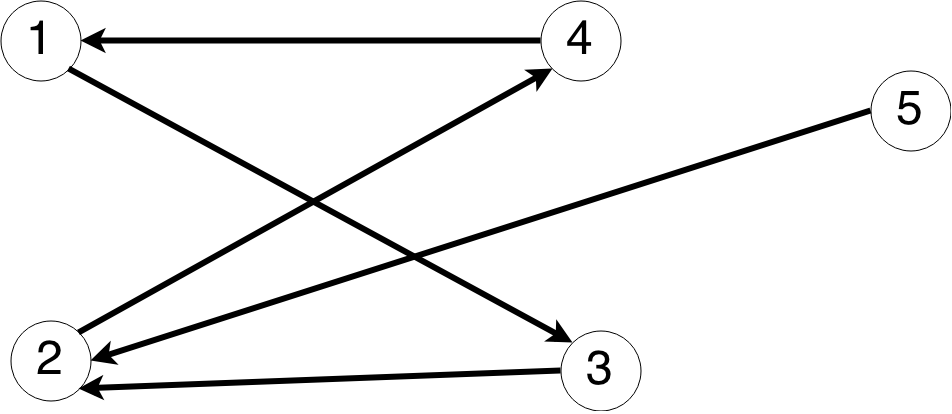
**Drumuri Minime și Maxime în grafuri Orientate**

**Atesta 2015**



**Autor : Micu Matei-Marius**

**Profesot coordonator : Oancea Constantin**

Drumuri Minime și Maxime în grafuri Orientate

**￼￼**

# Aspecte teoretice

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

unde :

**V :**  este o mulțime ,**finintă** şi **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**  şi  **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:

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

**Exemplu :**

**u1 = (x, y)**  şi **u2 = (y, z)** sunt incidente

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

# B. Metode de reprezentare a grafului orientat

Sunt admise două metode de reprezentare :

* **reprezentare textuală :** asa cum au fost prezentate până acum
* **reprezentare grafica :**  **arcele**  sunt reprezentate prin **săgeți orientate**

iar **nodurile**  prin **puncte.**

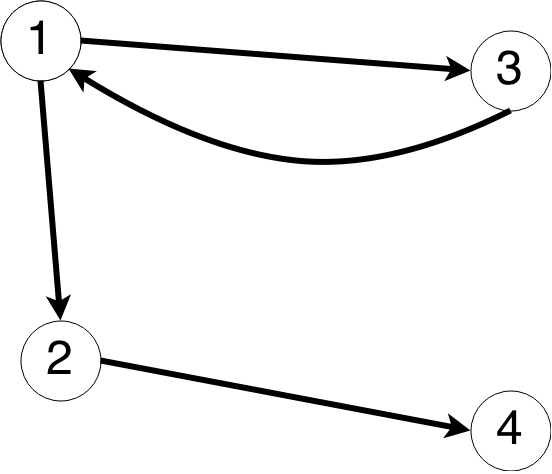
**Exemplu** de graf orientat reprezentat **textual:**

**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:**

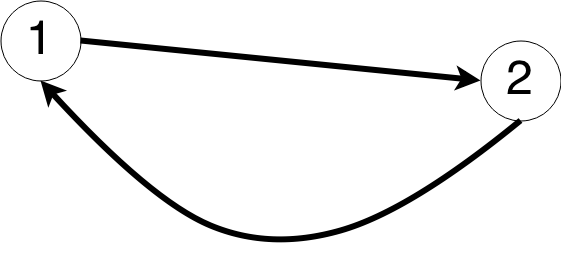
Considerând graful :

**G = (V, U)** unde :

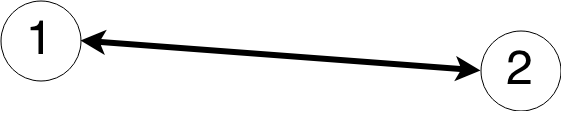
**V = { 1, 2 }**

**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 numeşte **graf parțial**  al grafului **G** **graful**

**orientat G1 = (V, U1)**  unde  **U1 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 şi **G** , iar mulțimea arcelor este o submulțime a

lui **U**  sau chiar **U**.

**Exemplu :**

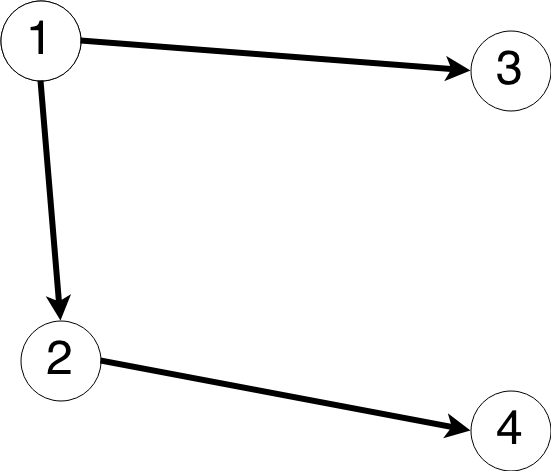
Fie graful orientat :

**G = (V, U)**

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

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

reprezentat **grafic**  astfel :



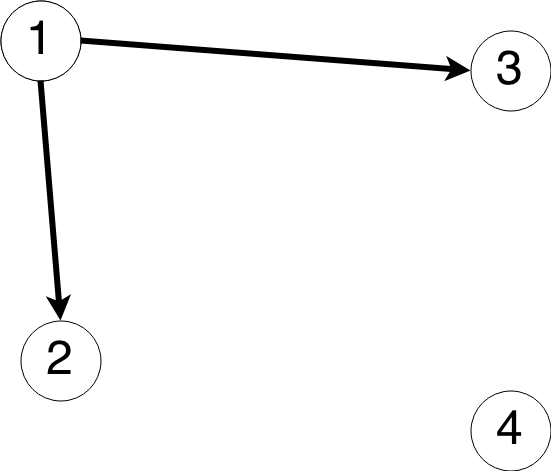
1. Un exemplu de **grafic parțial al lui G**  este graful orientat :

**G = (V, U)**

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

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

reprezentat **grafic**  astfel :



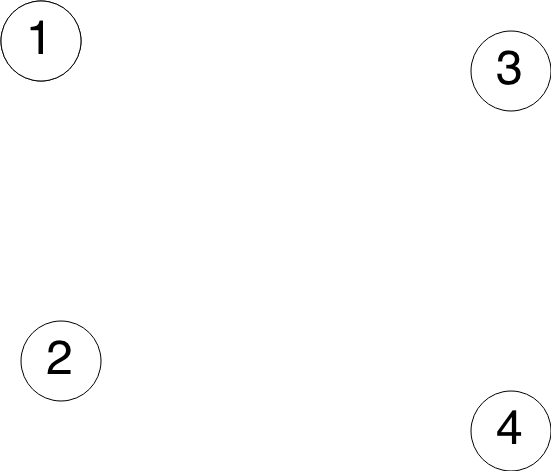
1. Un exemplu de **grafic parțial al lui G**  este graful orientat :

**G = (V, U)**

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

**U =** ( s-au eliminat toate arcele)

reprezentat **grafic**  astfel :



**Observatie !**

Fie **G = (V, U)** un **graf orientat**.**Un graf parțian** , al grafului **G**, **se poate obține**

**pastrând vârfurile și eliminănd 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 numeşte **subgraf**  al grafului **G** **graful**

**orientat G1 = (V1, U1)**  unde  **V1 V** , iar **U1** conține toate arcele din **U**

care au extremitățile în **V1**.

**Exemplu :**

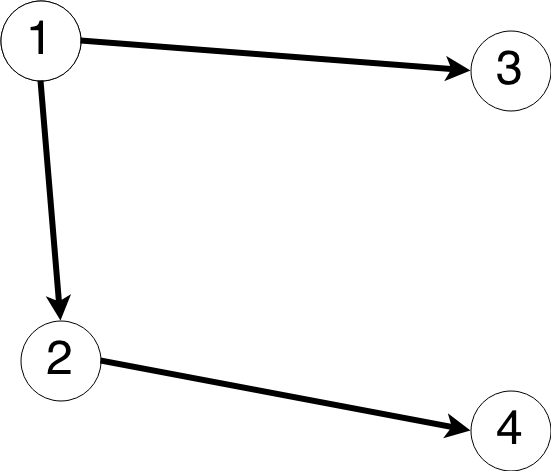
Fie graful orientat :

**G = (V, U)**

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

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

reprezentat **grafic**  astfel :



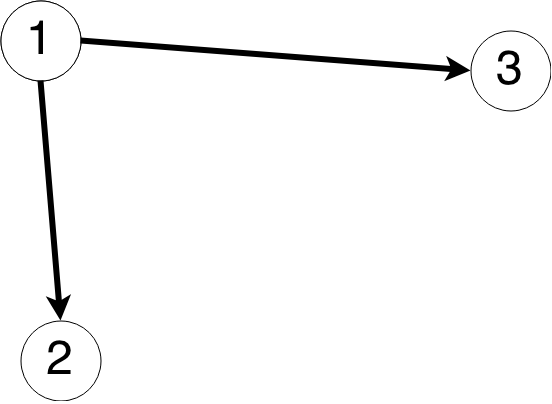
1. Un exemplu de **subgraf al lui G**  este graful orientat :

**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 :



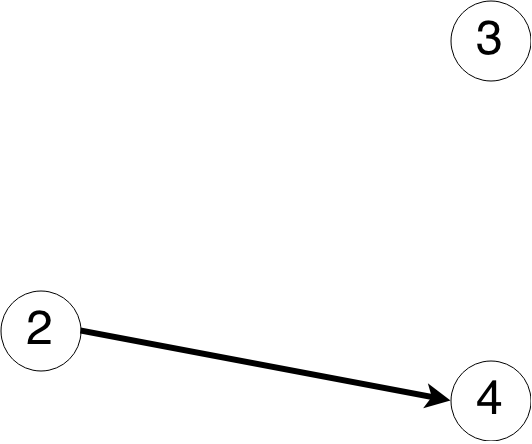
1. Un exemplu de **graf parțial al lui G**  este graful orientat :

**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 şi arcele care le admit ca extremitate** ( **nu**  se pot şterge **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 şi arce**

**care intră în acel vârf** “, iau naştere următoarele noțiuni :

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

**Definiție.** Fie **G=(V, U)** un graf orientat și **x**  un nod al său. Se numeşte **grad exterior al**

**nodului x , numărul arcelor de forma (x, y)** ( adică numărul arcelor care ies din x ),

notat .

**Exemplu :**

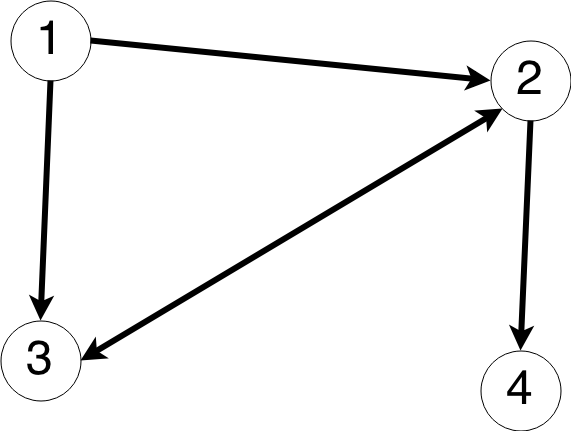
Fie grafiul :

**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  **(** în graf, sunt **trei** arce care  **ies**  din nodu 1)

**Gradul exterior** al  **nodului 2**  este  **(** în graf, sunt **doua** arce care  **ies**  din nodu 2)

**Gradul exterior** al  **nodului 3**  este  **(** în graf, sunt **zero** arce care  **ies**  din nodu 3)

**Gradul exterior** al  **nodului 4**  este  **(** în graf, sunt **zero** arce care  **ies**  din nodu 4)

**Observatii**

1. **Mulțimea succesorilor nodului x**  se notează cu şi se reprezintă astfel :

**2) Mulțimea arcelor ce ies din nodul x**  se notează cu şi se reprezintă astfel :

**3) Legat de cardinalele mulțimilor şi**  putem scrie :

Raportat la acelaşi graf putem scrie următoarele:

**Definiție.** Fie **G=(V, U)** un **graf orientat** şi **x**  un nod al său. Se numeşte **grad interior al**

**nodului x , numărul arcelor de forma (y, x)** ( adică numărul arcelor care intra din x ),

notat .

**Exemplu :**

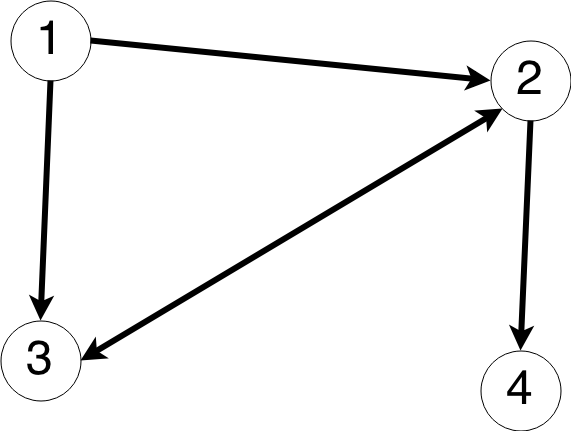
Fie graful :

**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  **(**în graf, sunt **zero** arce care  **intra**  din nodu 1)

**Gradul exterior** al  **nodului 2**  este  **(**în graf, sunt **doua** arce care  **intra**  din nodu2)

**Gradul exterior** al  **nodului 3**  este  **(**în graf, sunt **doua** arce care  **intra**  din nodu 3)

**Gradul exterior** al  **nodului 4**  este  **(**în graf, este **un**  arc care  **intra**  din nodu 4)

**Observatii.**

1. **Mulțimea predecesorilor nodului x**  se notează cu şi se reprezintă astfel :

**2) Mulțimea arcelor ce intra din nodul x**  se notează cu şi se reprezintă astfel :

**3) Legat de cardinalele mulțimilor şi**  putem scrie :

Raportat la același graf putem scrie următoarele:

**F. Conexitate**

În această secṭiune vor fi prezentate următoarele noțiuni :

* Drum
* Drum elementar
* Drum neelementar

**Drum**

**Definiție.**

Fie **G=(V, U)** un **graf orientat**. Se numeşte **drum** , în graful **G**, o

succesiune de noduri notată cu

proprietatea ( altfel spus

sunt arce).

Se întalnesc noțiunile:

* **extremitățile drumului**
  + fiind dat drumul se numesc

**extremități ale sale nodurile și** ( - **extremitate**

**iniṭială**; - **extremitate finală** );

* **lungimea drumului**
  + fiind dat drumul , prin **lungimea sa** se ințelege **numarul de arce care apar în cadrul său**;

**Exemplu:**

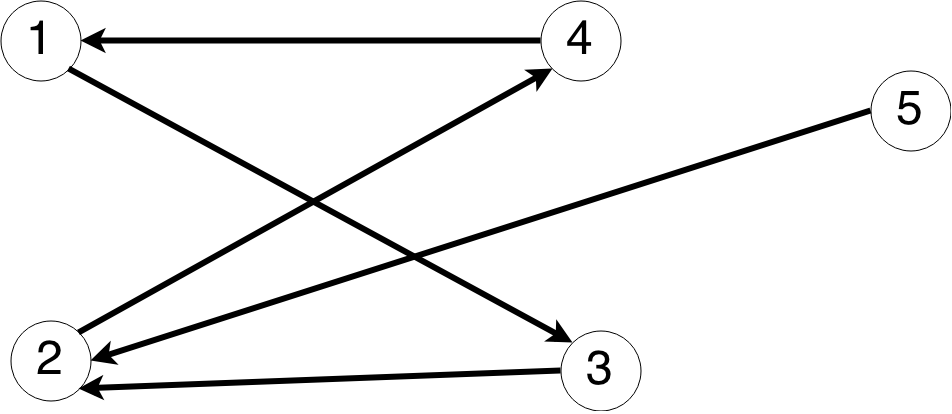
Fie graful :

**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 este drum în graful **G**, atunci nu neapărat şi

este drum în graful **G.**

**Drum elementar**

**Definiție.**

Fie **G=(V, U)** un **graf orientat**. Se numeşte **drum elementar** , în graful

**G**, drumul cu proprietatea că **oricare două noduri ale sale sunt distincte** ( altfel spus, **printr-un nod nu se trece decât o singură dată** ).

**Exemplu:**

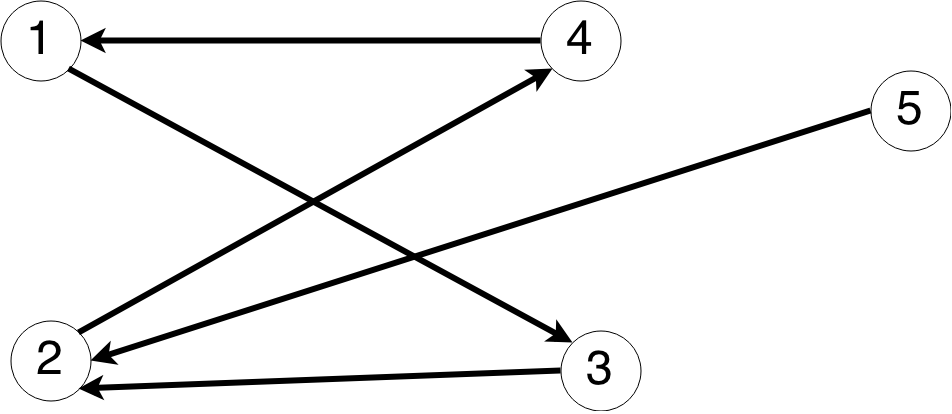
Fie graful :

**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 numeşte **drum elementar** , în graful

**G**, drumul cu proprietatea că **nodurile**

**sale nu sunt distincte două câte două**( altfel spus, **printr-un nod s-a**

**trecut de mai multe ori** ).

**Exemplu:**

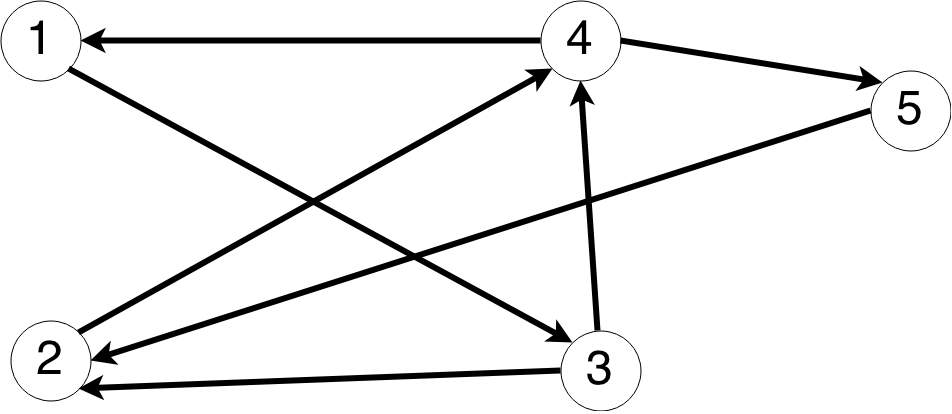
Fie graful :

**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 întai sa 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 sa ne folosim următoarele proprietăi ale unui graf orientat:

* **arcul (x, y) este diferit de arcul (y, x)**

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

**Definiție**

**Matricea de adiacență** ( ), asociată grafului **G**,este

**o matrice pătratică de ordin n**, cu elementele:

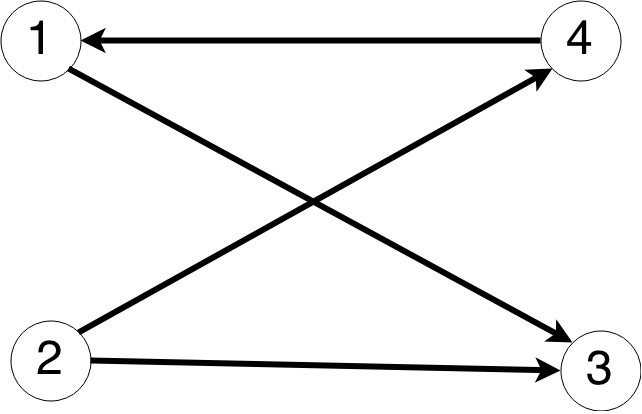


( altfel spus, dacă **există**  arc între **i**  şi  **j** şi dacă între  **i** şi  **j**

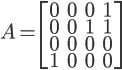
**nu există** arc )

**Exemplu 1 :**

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

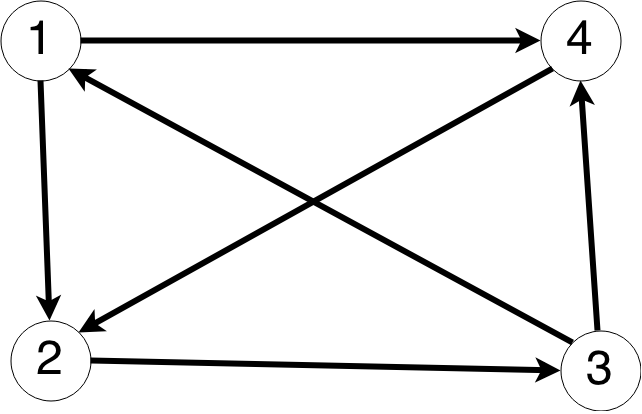


Matricea de adiacență asociată grafului este :

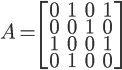


**Exemplu 2 :**

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



Matricea de adiacență asociată grafului este :



**Comentarii :**

**Matricea de adiacență**  este o matrice **pătratică**, de ordin n, și **nu este neapărat**

**simetrică** fața de diagonala principală, așa cum este cazul grafurilor neorientate.

**Secvențe de citire a matricei de adiacență:**

*program prrogram\_1;*

*var a :array[1..100, 1..100] of 0..1;*

*i, j, n :byte;*

*begin*

*write('n=');*

*readln(n); { numarul de noduri }*

*{ initlializam matriea cu 0 }*

*fillchar(a, sizeof(a), 0);*

*for i := 1 to n do*

*for j := 1 to n do*

*begin*

*{ citim valoarea (0 sau 1) a fiecarui element din matricea de adiacenta }*

*write('a[', i, ', ', j, ']=');*

*readln(a[i, j]);*

*end;*

……………

**sau**

*program prrogram\_2;*

*var a :array[1..100, 1..100] of 0..1;*

*i, j, n, m, x, y :byte;*

*begin*

*write('n=');*

*readln(n); { numarul de noduri }*

*write('m=');*

*readln(m); { numarul de muchii }*

*{ initlializam matriea cu 0 }*

*fillchar(a, sizeof(a), 0);*

*for i := 1 to m do*

*begin*

*write('x=');*

*readln(x); { extremitatea initiala a arcului (x, y) }*

*write('y=');*

*readln(y); { extremitatea finala a arcului (x, y) }*

*a[x, y] := 1;*

*end;*

…………...

**Comentarii :**

* **Matricea de 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 şi 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 urmăreste examinarea nodurilor

sale, plecând dintr-un nod dat **x** , astfel încât fiecare nod, la care se poate ajunge din x pe

muchii adiacente două câte două, sa fie marcat o singură dată.

Există două metode principale de parcurgere a grafurilor :

- **în lățime**

**- în adâncime**

**Metoda parcurgeri în lățime**

Această metodă foloseşte o structură de tip coadă. Se porneşte de la nodul **x** se

găsesc toate nodurile  **y a. î. (x, y) U,** apoi se ia fiecare nod y găsit şi se

repetă procesul pentru acesta .

*program program\_3;*

*var a :array[1..100, 1..100] of 0..1; { matricea de adiacenta }*

*c :array[0..100] of byte; { coada }*

*viz :array[1..100] of boolean; { vectorul cu noduri vizitate }*

*i, n, m, x, p :byte;*

*i\_c, n\_c :integer;*

*procedure citire\_matrice\_adiacenta;*

*{ citim matricea de adiacenta a grafului }*

*var i, x, y :integer;*

*begin*

*write('n=');*

*readln(n); { numarul de noduri }*

*write('m=');*

*readln(m); { numarul de muchii }*

*{ initlializam matriea cu 0 }*

*fillchar(a, sizeof(a), 0);*

*writeln('In continuare introduceti extremitatile arcelor :');*

*for i := 1 to m do*

*begin*

*write('x=');*

*readln(x); { extremitatea initiala a arcului (x, y) }*

*write('y=');*

*readln(y); { extremitatea finala a arcului (x, y) }*

*a[x, y] := 1;*

*writeln;*

*end;*

*end;*

*procedure init\_coada;*

*{ initializam coada }*

*begin*

*n\_c := -1;*

*i\_c := 0;*

*end;*

*procedure push\_coada(x :byte);*

*{ adaugam elementul <x> în coada }*

*begin*

*inc(n\_c);*

*c[n\_c] := x;*

*end;*

*function este\_vida\_coada:boolean;*

*{ returnam TRUE dacă coada este vida, și FALSE în caz contrar }*

*begin*

*este\_vida\_coada := (n\_c < i\_c );*

*end;*

*function pop\_coada:byte;*

*{ returnam primul element din coada și il eliminam*

*acest subprogram se utilizeaza doar cand coada nu*

*este vida }*

*begin*

*pop\_coada := c[i\_c]; { returnam elementul }*

*inc(i\_c); { eliminam elementul }*

*end;*

*procedure init\_viz;*

*{ initializam vectorul cu nodurile vizitate cu FALSE, nici*

*un nod nu a fost viziat }*

*begin*

*fillchar(viz, sizeof(viz), FALSE);*

*end;*

*procedure vizitat(i :byte);*

*{ marcam nodul <i> ca fiind vizitat }*

*begin*

*viz[i] := TRUE;*

*end;*

*function a\_fost\_vizitat(i :byte):boolean;*

*{ returneaza TRUE dacă nodul a fost viziat, în caz contrar*

*returneaza FALSE }*

*begin*

*a\_fost\_vizitat := viz[i];*

*end;*

*procedure prelucrare\_nod(x :byte);*

*{ prelugram nodul, în cazu de fata o sa il afisam doar }*

*begin*

*writeln('Prelugram nodul :', x);*

*end;*

*begin*

*init\_coada; { initializam coada }*

*citire\_matrice\_adiacenta; { citim toate elementele grafului }*

*write('De la ce nod doriti sa pornim : ');*

*readln(x); { citim nodul de pornire }*

*push\_coada(x); { adaugam <x> în coada }*

*while not este\_vida\_coada() do { cat timp mai avem noduri de prelucrat în coada }*

*begin*

*p := pop\_coada; { scoatem un element din coada }*

*prelucrare\_nod(p); { prelucram nodul }*

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

*{ cautam toate nodurile la care putem ajunge prin nodul x }*

*for i := 1 to n do*

*if (a[p, i] = 1) AND not a\_fost\_vizitat(i) then { dacă exista arc de la <p> la*

*<i> și dacă nu am vizitat nodul <i> }*

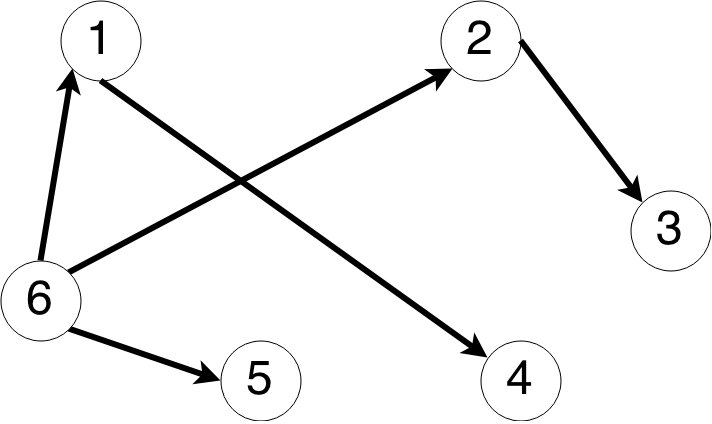
*push\_coada(i); { adauga nodul în coada }*

*end;*

*writeln;*

*end.*

Dacă avem graful reprezentat grafic astfel :



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

**6, 1, 2, 5, 4, 1**

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

**1, 4**

**Observatie :**

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

**Metoda parcurgeri în adâncime**

Această metodă foloseşte o structură de tip stiva. Se porneşte de la nodul **x** se

găseşte primul nod  **y a. î. (x, y) U,** iar pentru acest **y**  se repetă

procesul, când ajungem la un nod care nu are vecini nevizitați, coborâm în

stivă şi cautăm un alt vecin nevizitat, până stiva devine vidă .

*program program\_4;*

*var a :array[1..100, 1..100] of 0..1; { matricea de adiacenta }*

*s :array[0..100] of byte; { stiva }*

*viz :array[1..100] of boolean; { vectorul cu noduri vizitate }*

*i, n, m, x, p :byte;*

*i\_s, n\_s :integer;*

*ok :boolean;*

*procedure citire\_matrice\_adiacenta;*

*{ citim matricea de adiacenta a grafului }*

*var i, x, y :integer;*

*begin*

*write('n=');*

*readln(n); { numarul de noduri }*

*write('m=');*

*readln(m); { numarul de muchii }*

*{ initlializam matriea cu 0 }*

*fillchar(a, sizeof(a), 0);*

*writeln('In continuare introduceti extremitatile arcelor :');*

*for i := 1 to m do*

*begin*

*write('x=');*

*readln(x); { extremitatea initiala a arcului (x, y) }*

*write('y=');*

*readln(y); { extremitatea finala a arcului (x, y) }*

*a[x, y] := 1;*

*writeln;*

*end;*

*end;*

*procedure init\_stiva;*

*{ initializam stiva }*

*begin*

*n\_s := 0;*

*i\_s := 0;*

*end;*

*procedure push\_stiva(x :byte);*

*{ adaugam elementul <x> în stiva }*

*begin*

*inc(n\_s);*

*s[n\_s] := x;*

*end;*

*function este\_vida\_stiva:boolean;*

*{ returnam TRUE dacă stiva este vida, 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 viziat, în caz contrar*

*returneaza FALSE }*

*begin*

*a\_fost\_vizitat := viz[i];*

*end;*

*procedure prelucrare\_nod(x :byte);*

*{ prelugram nodul, în cazu de fata o sa il afisam doar }*

*begin*

*writeln('Prelugram nodul :', x);*

*end;*

*begin*

*init\_stiva; { initializam stiva }*

*citire\_matrice\_adiacenta; { citim toate elementele grafului }*

*write('De la ce nod doriti sa pornim : ');*

*readln(x); { citim nodul de pornire }*

*push\_stiva(x); { adaugam <x> în stiva }*

*while not este\_vida\_stiva() do { cat timp mai avem noduri de prelucrat în stiva }*

*begin*

*p := acces\_stiva; { accesam elementul din varful stivei }*

*if not a\_fost\_vizitat(p) then { dacă elementul nu a fost vizitat }*

*begin*

*prelucrare\_nod(p); { prelucreazal }*

*vizitat(p); { noteazal ca fiind vizitat }*

*end;*

*{ cautam un nod de la care putem ajunge de la p, care nu a fost vizitat }*

*ok := false; { presupunem ca nu exista nici un nod }*

*for i := 1 to n do*

*if (a[p, i] = 1) AND ( not a\_fost\_vizitat(i) ) then*

*begin*

*push\_stiva(i);*

*ok := true;*

*break;*

*end;*

*if not ok then*

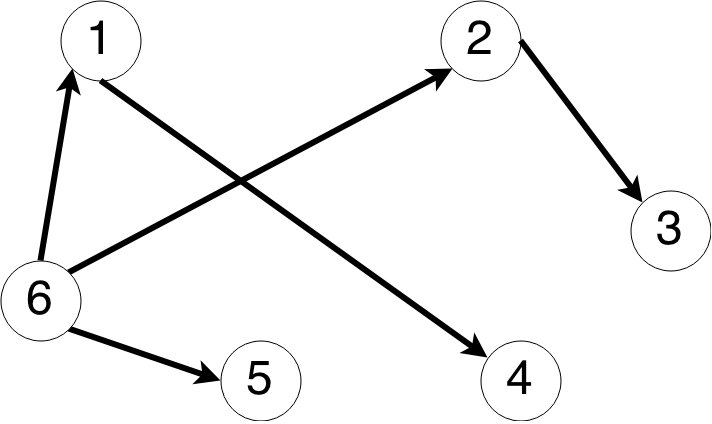
*pop\_stiva();*

*end;*

*writeln;*

*end.*

Dacă avem graful reprezentat grafic astfel :



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

**6, 1, 4, 2, 3, 5**

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

**1, 4**

**Observatie :**

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

**Comentarii :**

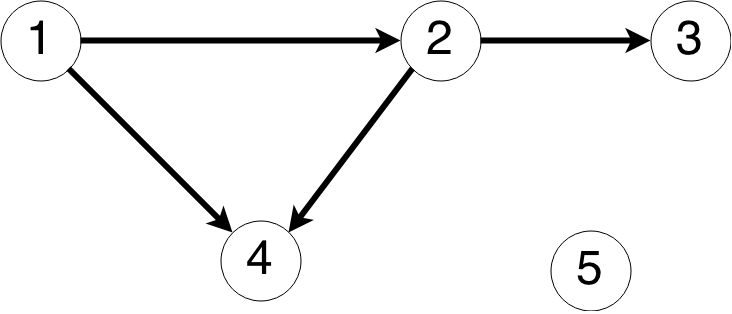
Aceste metode nu parcurg tot graful, dacă avem de exemplu

**G = (V, U)**

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

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

Reprezentat grafic astfel :



Dacă îl parcurgem în 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 V a. î. (x, 5) 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**

* Se poate cere determinarea **drumului de lungime minimă ( maximă )**

**dintre un nod i și un nod j în graful G = (V, U) ştiinduse matricea**

**cu costuri asociată grafului C**

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

În cazul **problemelor de minim** , fiind dat graful **G = (V, U)** i se asociază **matricea**

**costurilor, forma 1**, definită astfel :

**C , unde :**



În cazul **problemelor de minimmaxim** , fiind dat graful **G = (V, U)** i se asociază **matricea**

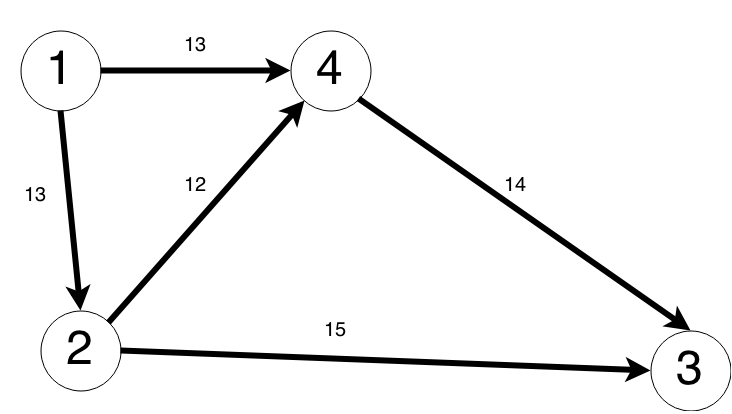
**costurilor, forma 2**, definită astfel :

**C , unde :**



**Exemplu :**

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



Matricea costurilor de adiacența ar arăta în felul următor :

**Forma 1 :**



**Forma 2:**



**Observații :**

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

*const m\_infinit = -1.e10;*

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

* **Roy-Floyd**
* **Dijkstra**

## Algoritmul Roy-Floyd

Acest algoritm se aplică în cazul în care se dă un graf **G = (V, U )**, care are matricea costurilor **C**, şi se cere să se determine **lungimea drumurilor minime( maxime ),** 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 şir de n transformări aplicate matricei costurilor **C**, astfel :

În esentă, vom vedea dacă costul unui drum  **i - j**  este mai mic decât suma costurilor drumurilor

**i - k**  şi **k - j** , dacă **i - j < i - k + k - j,** ştim că drumul minim de la  **i**  la  **j**  trece prin  **k.**

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

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

Matricea drumurilor se inițializează în felul următor :

* dacă atunci exista arc între **i și j** ,adică
* dacă atunci nu există arc între  **și j** ,adică

Matricea drumurilor se inițializează după citirea matricii costurilor ( de unde vedem ce arce sunt în graf şi 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 atunci şi
* dacă atunci rămane același, şi rămane același

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

**Exemplu**  a conținutului unui fişier :

*4*

*0 4 4 10*

*-1 0 -1 4*

*-1 -1 0 3*

*-1 -1 -1 0*

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

*program algoritmul\_roy\_floyd;*

*uses crt;*

*const p\_infinit = 1.e10;*

*m\_infinit = -1.e10;*

*n\_max = 100;*

*type*

*vector=array[1..n\_max] of integer;*

*drum = record { va retine un drum }*

*v :vector; { nodurile drumului }*

*n :integer; { cate noduri are drumul }*

*end;*

*var drumuri :array[1..n\_max, 1..n\_max] of drum; { matricea care va retine drumul de la un*

*element la latul }*

*costuri :array[1..n\_max, 1..n\_max] of real; { matricea costurilor }*

*i, j, n :integer;*

*f :text;*

*procedure citire\_mat\_costuri;*

*{ citim matricea de costuri, de unde aflam și ce muchii exista în graf*

*citirea se face dintr-un fisier 'costuri.txt' care are pe prima linie*

*numarul de noduri apoi pe fiecare linie următoare o linie din matricea costurilor }*

*var i, j :integer;*

*begin*

*assign(f, 'costuri.txt');*

*reset(f);*

*readln(f, n);*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*begin*

*read(f, costuri[i, j]);*

*if costuri[i, j] = -1 then*

*costuri[i, j] := p\_infinit;*

*end;*

*writeln;*

*end;*

*close(f);*

*end;*

*procedure init\_drum(var x :drum);*

*{ initializam un drum }*

*begin*

*x.n := 0;*

*end;*

*procedure init\_mat\_drumuri;*

*{ initializam matricea de drumuri, pentru arcele existente }*

*begin*

*for i := 1 to n do*

*for j := 1 to n do*

*begin*

*if (i <> j) AND (costuri[i, j] < p\_infinit) then*

*begin*

*drumuri[i, j].n := 2;*

*drumuri[i, j].v[1] := i;*

*drumuri[i, j].v[2] := j;*

*end*

*else*

*init\_drum(drumuri[i, j]);*

*end;*

*end;*

*function concat\_drum(x, y :drum):drum;*

*{ alipim doua drumuri, nodul din mijloc se scrie o singura data }*

*var i :integer;*

*aux :drum;*

*begin*

*init\_drum(aux); { initializam drumul }*

*if (x.v[x.n] = y.v[1]) AND (x.n > 0) AND (y.n > 0) then { dacă exista macar un nod în ambele drumuri }*

*begin*

*for i := 1 to x.n do*

*begin*

*aux.n := aux.n + 1;*

*aux.v[aux.n] := x.v[i];*

*end;*

*for i := 2 to y.n do*

*begin*

*aux.n := aux.n + 1;*

*aux.v[aux.n] := y.v[i];*

*end;*

*end;*

*concat\_drum := aux;*

*end;*

*procedure afis;*

*{ afisam drumurile Sicosturie }*

*var i, j, k :integer;*

*aux :drum;*

*begin*

*for i := 1 to n do*

*for j := 1 to n do*

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

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

*begin*

*{ afisam costul și drumul cel mai scurt }*

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

*writeln('Drumul este :');*

*aux := drumuri[i, j];*

*write(' ':5);*

*for k := 1 to aux.n do*

*write(aux.v[k], ' ');*

*writeln;*

*end*

*else*

*writeln('Intre nodul ', i, ', ', j, ' nu exista drum !');*

*end;*

*procedure roy\_floyd;*

*{ algoritmul principal }*

*var i, j, k :integer;*

*begin*

*for i := 1 to n do { pentru fiecare nod în graf }*

*for j := 1 to n do { pentru pereche de noduri i, j }*

*for k := 1 to n do { verificam pentru fiecare nod în graf }*

*if (k <> i) AND ( k <> j) then { dacă nodul k este diferit de nodurile i și j }*

*if ( costuri[i, j] > costuri[i, k] + costuri[k, j]) then { și dacă drumul i - k - j este mai*

*scurt decat drumul i - j }*

*begin*

*costuri[i, j] := costuri[i, k] + costuri[k, j]; { modifică costul cu noul cost, mai mic}*

*drumuri[i, j] := concat\_drum(drumuri[i, k], drumuri[k, j]); { alipeste drumurile i -*

*k și k - j și salveazale în locul drumului i - j }*

*end;*

*end;*

*begin*

*citire\_mat\_costuri; { citim costurile }*

*init\_mat\_drumuri; { initializam drumurile }*

*clrscr; { curatam ecranul }*

*roy\_floyd;*

*afis; { afisam rezultatul }*

*readln;*

*end.*

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

* În timpul citirii matricii costurilor trebuie să inițializăm matricea costurilor cu -∞ ( *m\_infinit* ) în loc de +∞ ( *p\_infinit* )

……………..

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*begin*

*read(f, costuri[i, j]);*

*if costuri[i, j] = -1 then*

*costuri[i, j] :=* ***m\_infinit****;*

*end;*

*writeln;*

*end;*

……………..

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

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

……………..

*for i := 1 to n do { pentru fiecare nod în graf }*

*for j := 1 to n do { pentru pereche de noduri i, j }*

*for k := 1 to n do { verificam pentru fiecare nod în graf }*

***if (k <> i) AND ( k <> j) AND ( costuri[i, k] > m\_infinit)***

***AND ( costuri[k, j] > m\_infinit) then { dacă nodul k este d iferit de nodurile i***

***si j, și dacă nu avem drumuri inexistente, adică***

***costuri = m\_infinit }***

*if ( costuri[i, j]* ***<*** *costuri[i, k] + costuri[k, j]) then { şi dacă drumul i - k - j este mai*

*lung decat drumul i - j }*

*begin*

*costuri[i, j] := costuri[i, k] + costuri[k, j]; { modifică costul cu noul cost, mai mic}*

*drumuri[i, j] := concat\_drum(drumuri[i, k], drumuri[k, j]); { alipeste drumurile i -*

*k și k - j şi salvează-le în locul drumului i - j }*

*end;*

……………..

* Și inițializarea matricii drumurilor trebuie modificată

……………..

*if (i <> j) AND (costuri[i, j]* ***> m\_infinit)*** *then*

*begin*

*drumuri[i, j].n := 2;*

*drumuri[i, j].v[1] := i;*

*drumuri[i, j].v[2] := j;*

*end*

……………..

* Un ultim lucru care trebuie modificat e procedura de afişare

……………..

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

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

*begin*

*{ afişm costul şi drumul cel mai scurt }*

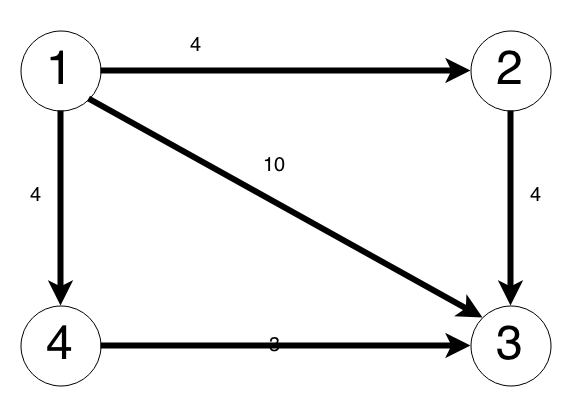
*writeln('Intre nodul ', i, ' ,', j, ' are costul ', costuri[i, j]:4:2);*

*writeln('Drumul este :');*

……………..

**Exemplu :**

Pentru graful :



care are matricea costurilor :

*4*

*0 4 4 10*

*-1 0 -1 4*

*-1 -1 0 3*

*-1 -1 -1 0*

Rezultatul ar trebui sa fie :

*Între nodul 1 ,2 are costul 4.00  
Drumul este :  
 1 2   
Între nodul 1 ,3 are costul 4.00  
Drumul este :  
 1 3   
Între nodul 1 ,4 are costul 7.00  
Drumul este :  
 1 3 4   
Între nodul 2, 1 nu există drum !  
Între nodul 2, 3 nu există drum !  
Între nodul 2 ,4 are costul 4.00  
Drumul este :  
 2 4   
Între nodul 3, 1 nu există drum !  
Între nodul 3, 2 nu există drum !  
Între nodul 3 ,4 are costul 3.00  
Drumul este :  
 3 4   
Între nodul 4, 1 nu există drum !  
Intre nodul 4, 2 nu exista drum !  
Intre nodul 4, 3 nu exista drum !*

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**, şi 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 procedeză astfel :**

* se citește matricea costurilor
* vectorul ***vizitate*** se initializează cu ***false***
* se citește nodul de pornire şi se noteaza ***vizitate[pornire] = true***
* se completează vectorul ***drum*** astfel :
  + pentru **i = 1 ... n,**  ***drum[i] = pornire, c[pornire, i] ≠ ∞ și i pornire;***
  + ***drum[i] = 0,*** *altfel;*
* se completează vectorul ***costuri\_drumuri*** astfel :
  + ***costuri\_drumuri[i] = c[pornire, i],*** pentru ***i = 1 ... n dacă i pornire***
  + ***costuri\_drumuri[pornire] = 0***

* cât timp există noduri nevizitate executăm următoarele:
  + se caută nodul ***k*** dintre cele nevizitate, a carui drum de la ***pornire*** la ***k*** este minim, adica ***costuri\_drumuri[k] minim*** şi ***vizitate[k] = false***
  + se calculează drumurile de la nodul de pornire la restul nodurilor prin nodul ***k***în felul următor :
    - calculăm costul minim dintre ***costuri\_drumuri[i]*** ( costul pe care îl ştim momentan de la ***pornire***  la ***i*** ) şi ***costuri\_drumuri[k]*** ( costul pe care îl ştim momentan de la ***pornire***  la ***k ) + c[k, i]*** ( adică costul arcului ***(k, i)***  *),* cu alte cuvinte, vedem care drum e mai scurt, cel pe care îl avem momentan de la ***pornire*** la  ***i*** sau cel de la  ***pornire***  la  ***k***  și apoi de la ***k***  la  ***i*** 
      * dacă costul de la ***pornire***  la ***i*** este cel mai mic, atunci nimic nu se întamplă
      * dacă costul de la ***pornire***  la ***k***  este cel mai mic, atunci ***costuri\_drumuri[i] = costuri\_drumuri[k] + c[k, i]***  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\_infinit = 1.e10;*

*m\_infinit = -1.e10;*

*max\_noduri = 1000;*

*type matrice=array[1..max\_noduri, 1..max\_noduri] of real;*

*vector\_i=array[1..max\_noduri] of integer;*

*vector=array[1..max\_noduri] of real;*

*var c :matrice; { matricea costurilor }*

*costuri\_drum : vector; { costul drumului de la nodu de pornire la <i> }*

*drum : vector\_i; { drumul de la nodu de pornire la oricare nod }*

*vizitate :array[1..max\_noduri] of boolean; { tinem cont de ce noduri am vizitat }*

*n, pornire :integer;*

*f :text;*

*procedure citire\_matrice\_costuri;*

*var i, j :integer;*

*begin*

*assign(f, 'costuri.txt');*

*reset(f);*

*readln(f, n);*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*begin*

*read(f, c[i, j]);*

*if c[i, j] = -1 then { intifint în text e codificat ca și <-1> }*

*c[i, j] := p\_infinit;*

*end;*

*readln(f);*

*end;*

*close(f);*

*end;*

*procedure init\_vizitate;*

*{ initializam vectorul cu noduri vizitate }*

*var i :integer;*

*begin*

*for i := 1 to n do*

*vizitate[i] := False;*

*end;*

*procedure vizitat(x :integer);*

*{ notam nodul <x> ca fiind vizitat }*

*begin*

*vizitate[x] := True;*

*end;*

*function exista\_neviz:boolean;*

*{ returnam True dacă mai exista noduri nevizitate }*

*var i :integer;*

*aux :boolean;*

*begin*

*aux := false; { presupunem ca nu mai exista noduri de vizitat }*

*for i := 1 to n do*

*if vizitate[i] = False then { dacă nodul <i> nu a fost vizitat }*

*begin*

*aux := true; { atunci mai exista noduri de vizitat }*

*break; { oprin cautarea }*

*end;*

*exista\_neviz := aux;*

*end;*

*function este\_neviz(x :integer):boolean;*

*{ returnam True dacă nodul <x> este nevizitat, false în faz contrar }*

*begin*

*este\_neviz := not vizitate[x];*

*end;*

*procedure init\_drum\_costuri;*

*{ initlializam costurile fiecarui drum }*

*var i : integer;*

*begin*

*for i := 1 to n do*

*if (i <> pornire) then { dacă i <> pornire }*

*costuri\_drum[i] := c[pornire, i] { drumul e constituit doar dintr-un arc și costul este cher costul arcului }*

*else*

*costuri\_drum[i] := 0; { nu exista arc, deci initializam cu 0 }*

*end;*

*procedure init\_drum;*

*{ initlializam vectorul drumurilor de la nodu de pornire la restu }*

*var i :integer;*

*begin*

*for i := 1 to n do*

*if (i <> pornire) AND (c[pornire, i] < p\_infinit) then { dacă exista arc de la nodu de pornire la i }*

*drum[i] := pornire { fiind doar un arc, extremitatea initiala e chear nodul de pornire }*

*else*

*drum[i] := 0 ;*

*end;*

*procedure dijkstra;*

*var i, k :integer;*

*min :real;*

*begin*

*while exista\_neviz() do { cat mai exista noduri nevizitate }*

*begin*

*min := p\_infinit;*

*k := 0;*

*for i := 1 to n do { cautam nodul la care putem ajunge cel mai usor }*

*if este\_neviz(i) AND (costuri\_drum[i] < min )then*

*begin*

*min := costuri\_drum[i];*

*k := i;*

*end;*

*vizitat(k);*

*for i := 1 to n do*

*if este\_neviz(i) AND (c[k, i] < p\_infinit) then { dacă exista arc de la k la i }*

*if costuri\_drum[i] > costuri\_drum[k] + c[k, i] then { dacă drumul de la nodu pe care il avem de la nodu de*

*pornire pana la <i> este mai lung decat drumul prin nodul*

*<k> atunci drumul minim trece prin k }*

*begin*

*costuri\_drum[i] := costuri\_drum[k] + c[k, i]; { costul minim e cel al drumului pana la <k> + de la <k> la <i>}*

*drum[i] := k; { la nodul <i> se ajunge prin nodul <k> }*

*end;*

*end;*

*end;*

*procedure afisare\_drum(i :integer);*

*{ afisam drumul de la nodu de pornire pana la nodul <i> }*

*begin*

*if i <> 0 then*

*begin*

*afisare\_drum(drum[i]);*

*write(i, ' ');*

*end;*

*end;*

*procedure afis;*

*var i :integer;*

*begin*

*for i := 1 to n do*

*if i <> pornire then*

*if drum[i] <> 0 then { am gasit un drum de la <pornire> la <i> }*

*begin*

*writeln('Drumul de la ', pornire, ' la ', i, ' are lungimea :', costuri\_drum[i]:4:2);*

*writeln('Si este compus din ');*

*write(' ':7);*

*afisare\_drum(i);*

*writeln;*

*end*

*else*

*writeln('Nu exista drum de la ', pornire, ' la ', i);*

*end;*

*begin*

*citire\_matrice\_costuri; { citim matricea de costuri }*

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

*readln(pornire);*

*vizitat(pornire);*

*init\_drum;*

*init\_drum\_costuri;*

*dijkstra;*

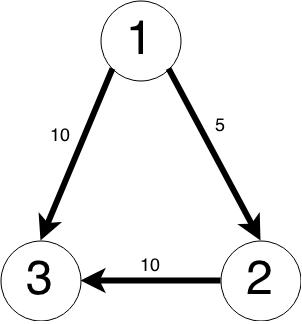
*afis;*

*end.*

Algoritmul lui **Dijkstra** nu poate fi modificat pentru a căuta drumurile maxime, el va gǎsi 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\_infinit = 1.e10;

m\_infinit = -1.e10;

max\_noduri = 1000;

type matrice=array[1..max\_noduri, 1..max\_noduri] of real;

vector\_i=array[1..max\_noduri] of integer;

vector=array[1..max\_noduri] of real;

var c, c2 :matrice; { matricea costurilor }

costuri\_drum : vector; { costul drumului de la nodu de pornire la <i> }

drum : vector\_i; { drumul de la nodu de pornire la oricare nod }

vizitate :array[1..max\_noduri] of boolean; { tinem cont de ce noduri am vizitat }

n, pornire :integer;

f :text;

procedure citire\_matrice\_costuri;

var i, j :integer;

begin

assign(f, 'costuri.txt');

reset(f);

readln(f, n);

for i := 1 to n do

begin

for j := 1 to n do

begin

read(f, c[i, j]);

if c[i, j] = -1 then { intifint in text e codificat ca si <-1> }

c[i, j] := p\_infinit;

end;

readln(f);

end;

close(f);

end;

procedure init\_vizitate;

{ initializam vectorul cu noduri vizitate }

var i :integer;

begin

for i := 1 to n do

vizitate[i] := False;

end;

procedure vizitat(x :integer);

{ notam nodul <x> ca fiind vizitat }

begin

vizitate[x] := True;

end;

function exista\_neviz:boolean;

{ returnam True daca mai exista noduri nevizitate }

var i :integer;

aux :boolean;

begin

aux := false; { presupunem ca nu mai exista noduri de vizitat }

for i := 1 to n do

if vizitate[i] = False then { daca nodul <i> nu a fost vizitat }

begin

aux := true; { atunci mai exista noduri de vizitat }

break; { oprin cautarea }

end;

exista\_neviz := aux;

end;

function este\_neviz(x :integer):boolean;

{ returnam True daca nodul <x> este nevizitat, false in faz contrar }

begin

este\_neviz := not vizitate[x];

end;

procedure init\_drum\_costuri;

{ initlializam costurile fiecarui drum }

var i : integer;

begin

for i := 1 to n do

if (i <> pornire) then { daca i nu este nodul de pornire }

costuri\_drum[i] := c[pornire, i] { drumul e constituit doar dintr-un arc si costul este cher costul arcului }

else

costuri\_drum[i] := 0; { nu exista arc, deci initializam cu 0 }

end;

procedure init\_drum;

{ initlializam vectorul drumurilor de la nodu de pornire la restu }

var i :integer;

begin

for i := 1 to n do

if (i <> pornire) AND (c[pornire, i] < p\_infinit) then { daca exista arc de la nodu de pornire la i }

drum[i] := pornire { fiind doar un arc, extremitatea initiala e chear nodul de pornire }

else

drum[i] := 0 ;

end;

procedure dijkstra;

var i, k, c1 :integer;

min :real;

begin

c1 := 0;

while exista\_neviz() and (c1 < n) do { cat mai exista noduri nevizitate }

(\* for j := 1 to n-2 do \*)

begin

c1 := c1 + 1;

min := p\_infinit;

k := 0;

for i := 1 to n do { cautam nodul la care putem ajunge cel mai usor }

if este\_neviz(i) AND (costuri\_drum[i] < min )then

begin

min := costuri\_drum[i];

k := i;

end;

vizitat(k);

for i := 1 to n do

if este\_neviz(i) AND (c[k, i] <> p\_infinit) then { daca exista arc de la k la i }

if (costuri\_drum[i] > costuri\_drum[k] + c[k, i]) OR (costuri\_drum[i] = p\_infinit) then { daca drumul de la nodu pe care il avem de la nodu de

pornire pana la <i> este mai lung decat drumul prin nodul

<k> atunci drumul minim trece prin k }

begin

costuri\_drum[i] := costuri\_drum[k] + c[k, i]; { costul minim e cel al drumului pana la <k> + de la <k> la <i>}

drum[i] := k; { la nodul <i> se ajunge prin nodul <k> }

end;

end;

end;

procedure afisare\_drum(i :integer);

{ afisam drumul de la nodu de pornire pana la nodul <i> }

begin

if i <> 0 then

begin

afisare\_drum(drum[i]);

write(i, ' ');

end;

end;

function calc\_cost(k :integer):real;

begin

if k <> 0 then

begin

calc\_cost := c2[drum[k], k] + calc\_cost(drum[k]);

end

else

calc\_cost := 0;

end;

procedure afis;

var i :integer;

begin

for i := 1 to n do

if i <> pornire then

if drum[i] <> 0 then { am gasit un drum de la <pornire> la <i> }

begin

writeln('Drumul de la ', pornire, ' la ', i, ' are lungimea :', calc\_cost(i):4:2);

writeln('Si este compus din ');

write(' ':7);

afisare\_drum(i);

writeln;

end

else

writeln('Nu exista drum de la ', pornire, ' la ', i);

end;

procedure re\_init\_costuri;

var i, j :integer;

begin

c2 := c;

for i := 1 to n do

for j := 1 to n do

if (c[i, j] < p\_infinit) AND (c[i, j] > 0) then

c[i, j] := 1 / c[i, j];

end;

begin

citire\_matrice\_costuri; { citim matricea de costuri }

re\_init\_costuri;

write('Nodul de pornire :');

readln(pornire);

vizitat(pornire);

init\_drum;

init\_drum\_costuri;

dijkstra;

afis;

end.

## Program auxiliar :

Deoarece programele din această lucrare citesc matricea de adiacență și cea a costurilor

dintr-in fisier, programul următor va citi arcele și costul acestora și va genera și salva

într-un fisier matricea costurilor automat :

*program algoritm\_creeare\_matrice\_costuri;*

*var i, j, n, m, x, y :integer;*

*v :array[1..1000, 1..1000] of real;*

*f :text;*

*s :string;*

*begin*

*write('Numarul de noduri :');*

*readln(n);*

*write('Numarul de arce :');*

*readln(m);*

*{ initializam matricea costurilor }*

*for i := 1 to n do*

*for j := 1 to n do*

*if i = j then*

*v[i, j] := 0*

*else*

*v[i, j] := -1;*

*{ citim muchiile }*

*for i := 1 to m do*

*begin*

*write('x= ');*

*readln(x);*

*write('y= ');*

*readln(y);*

*write('Costul arcului ', x, ' ', y,' :');*

*readln(v[x, y]);*

*end;*

*{ afisam matricea costurilor }*

*writeln;*

*writeln;*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*write(v[i, j]:4:2, ' ');*

*writeln;*

*end;*

*writeln;*

*writeln;*

*repeat*

*write('Doriti sa salvam matricea costurilor intr-un fisier?(y/n) ');*

*readln(s);*

*until (s = 'y') OR (s = 'n');*

*if s = 'y' then*

*begin*

*write('Numele fisierului in care doriti sa salvam : ');*

*readln(s);*

*assign(f, s);*

*rewrite(f);*

*writeln(f, n);*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*write(f, v[i, j]:4:2, ' ');*

*writeln(f);*

*end;*

*close(f);*

*end;*

*end.*

# I. Program Complex :

În continuare vă voi prezenta un program care toți algoritmii prezentați pană acum :

*program program\_complex;*

*uses crt;*

*const n\_max = 1000;*

*p\_inf = 1.e10;*

*m\_inf = -1.e10;*

*type*

*vector\_int = array[1..n\_max] of integer;*

*vector\_bool = array[1..n\_max] of boolean;*

*matrice\_int = array[1..n\_max, 1..n\_max] of integer; { vector cu elementei integer }*

*matrice\_real = array[1..n\_max, 1..n\_max] of real; { vector cu elemente reale }*

*matrice\_bool = array[1..n\_max, 1..n\_max] of boolean; { vector cu elemente de tip boolean }*

*drum = record { o structura care retine un drum }*

*v :vector\_int; { elementele drumului }*

*n : integer; { numarul de elemente ale drumului }*

*end;*

*stiva = object*

*vector :vector\_int; { vector de stocare }*

*n :integer; { numar de elemente }*

*procedure init; { initializam stiva }*

*procedure push(x :integer); { adaugam un element in stiva }*

*procedure pop; { eliminam un element din stiva }*

*function acces:integer; { accesam elementul din varful stivei }*

*function vida:boolean; { verificam daca stiva este sau nu vida }*

*end;*

*coada = object*

*vector :vector\_int; { vector de stocare }*

*n, i :integer; { numar de elemente }*

*procedure init; { initializam coada }*

*procedure push(x :integer); { adaugam un element in coada }*

*procedure pop; { scoatem un element din coada }*

*function acces:integer; { accesam primul element din coada }*

*function vida:boolean; { verificam daca coada este vida sau nu }*

*end;*

*vizitate = object*

*vector :vector\_bool;*

*n :integer;*

*procedure init(x :integer); { initializam vectorul de noduri vizitate }*

*procedure vizitat(x :integer); { notam nodul <x> ca fiind vizitat }*

*function exista\_neviz: boolean; { vedem daca mai avem noduri nevizitate }*

*function este\_neviz(x :integer): boolean; { vedem daca nodul <x> este nevizitat }*

*end;*

*matrice\_drum = array[1..n\_max, 1..n\_max] of drum;*

*graf = object*

*nume, fisier, cauta\_roy, cauta\_dj :string; { numele grafului, si numele fisierului de unde a fost citit, cauta - reprezinta ce s-a cautat, drumul*

*minim sau maxim }*

*n :integer; { n - numarul de noduri al grafului }*

*citit, roy\_floyd, dijkstra :boolean; { retine daca graful a fost sau nu citit }*

*mat\_adiacenta :matrice\_bool; { matricea de adiacenta }*

*mat\_drum :matrice\_drum; { matricea de drumuri, pentru roy\_floyd }*

*mat\_costuri, mat\_costuri\_roy, mat\_costuri\_dj :matrice\_real; { matricea de costuri }*

*matrice\_drum\_dj :matrice\_int; { fiecare linie din matrice va fi un vector de drumuri pentru algoritmul dikkstra }*

*coada :coada; { coada }*

*stiva :stiva; { stiva }*

*vizitate :vizitate; { vector cu noduri vizitate }*

*procedure init;*

*procedure initializare(x :integer); { procedure de initializare a grafului dupa ce stim numarul de noduri }*

*procedure initializare\_fisier(x :string); { initlializam numele fisierului }*

*procedure afisare; { afisam lista arcelor }*

*procedure citire\_mat\_adiacenta; { procedure de citire a matrici de adiacenta din fisier }*

*procedure citire\_mat\_costuri; { procedura care citeste matricea de costuri }*

*procedure config\_mat\_costuri(x :real); { initializam matricea costurilor pentru a cauta drumurile maxime sau minime }*

*procedure parcurgere\_in\_latime(x :integer); { parcurgem graful in latime pornind de la nodul <x> si afisam nodurile gasite }*

*procedure parcurgere\_in\_adancime(x :integer); { parcurgem graful in adancime pornind de la nodul <x> si afisam nodurile gasite }*

*procedure set\_nume(s :string); { setam un nume }*

*procedure set\_file(s :string); { setam un fisier }*

*procedure afis\_mat\_adiacenta; { afisam matricea de adiacenta }*

*procedure afis\_mat\_costuri; { afisam matricea costurilor }*

*procedure afis\_lista\_noduri; { afisam lista nodurilor }*

*procedure afis\_lista\_noduri\_cost; { afisam lista nodurilor si a costurilor }*

*procedure roy\_floyd\_min; { aflam drumurile minime folosind algoritmul roy\_floyd }*

*procedure roy\_floyd\_max; { aflam drumurile minime folosind algoritmul roy\_floyd }*

*procedure dijkstra\_min(x :integer); { aflam drumurile minime del a <x> la restul nodurilor folosind algorimtul dijkstra}*

*procedure afis\_drum\_roy(s :string); { afisam drumurile maxime }*

*procedure afis\_drum\_dj(x :integer); {afisam drumurile minime de la un nod anume la alte noduri }*

*procedure drum\_min(a, b :integer); { afisam drumul minim dintre nodul <a> si nodul <b> }*

*procedure drum\_max(a, b :integer); { afisam drumul maxim dintre nodul <a> si nodul <b> }*

*end;*

*var gf :graf;*

*{===================================================================================}*

*{ Metode drumuri }*

*procedure afis\_dr(x, i :integer);*

*begin*

*if i <> 0 then*

*begin*

*afis\_dr(x, gf.matrice\_drum\_dj[x, i]);*

*write(i, ' ');*

*end;*

*end;*

*procedure afis\_mat\_drum;*

*var i, j, k :integer;*

*begin*

*for i := 1 to gf.n do*

*for j := 1 to gf.n do*

*begin*

*writeln('Drum : ', i, ' - ', j);*

*for k := 1 to gf.mat\_drum[i, j].n do*

*write(gf.mat\_drum[i, j].v[k], ' ');*

*writeln;*

*end;*

*end;*

*function inf\_or\_nr(x :real):string;*

*begin*

*if x = p\_inf then*

*inf\_or\_nr := 'infinit'*

*else*

*if x = m\_inf then*

*inf\_or\_nr := '-infinit'*

*else*

*str(x:4:2, inf\_or\_nr);*

*end;*

*procedure init\_drum(var x :drum);*

*{ initializam un drum }*

*begin*

*x.n := 0;*

*end;*

*procedure init\_mat\_drumuri\_min;*

*{ initializam matricea de drumuri, pentru arcele existente }*

*var i, j :integer;*

*begin*

*for i := 1 to gf.n do*

*for j := 1 to gf.n do*

*begin*

*if (i <> j) AND (gf.mat\_costuri[i, j] < p\_inf) then*

*begin*

*gf.mat\_drum[i, j].n := 2;*

*gf.mat\_drum[i, j].v[1] := i;*

*gf.mat\_drum[i, j].v[2] := j;*

*end*

*else*

*init\_drum(gf.mat\_drum[i, j]);*

*end;*

*end;*

*procedure init\_mat\_drumuri\_max;*

*{ initializam matricea de drumuri, pentru arcele existente }*

*var i, j :integer;*

*begin*

*for i := 1 to gf.n do*

*for j := 1 to gf.n do*

*begin*

*if (i <> j) AND (gf.mat\_costuri[i, j] > m\_inf) then*

*begin*

*gf.mat\_drum[i, j].n := 2;*

*gf.mat\_drum[i, j].v[1] := i;*

*gf.mat\_drum[i, j].v[2] := j;*

*end*

*else*

*init\_drum(gf.mat\_drum[i, j]);*

*end;*

*end;*

*function concat\_drum(x, y :drum):drum;*

*{ alipim doua drumuri, nodul din mijloc se scrie o singura data }*

*var i :integer;*

*aux :drum;*

*begin*

*init\_drum(aux); { initializam drumul }*

*if (x.v[x.n] = y.v[1]) AND (x.n > 0) AND (y.n > 0) then { daca exista macar un nod in ambele drumuri }*

*begin*

*for i := 1 to x.n do*

*begin*

*aux.n := aux.n + 1;*

*aux.v[aux.n] := x.v[i];*

*end;*

*for i := 2 to y.n do*

*begin*

*aux.n := aux.n + 1;*

*aux.v[aux.n] := y.v[i];*

*end;*

*end;*

*concat\_drum := aux;*

*end;*

*procedure init\_vector\_drum(x :integer);*

*var i :integer;*

*begin*

*for i := 1 to gf.n do*

*begin*

*if (x <> i) AND (gf.mat\_costuri[x, i] < p\_inf) then*

*gf.matrice\_drum\_dj[x, i] := x*

*else*

*gf.matrice\_drum\_dj[x, i] := 0;*

*writeln;*

*end;*

*end;*

*{===================================================================================}*

*{ Metode obiect vizitate }*

*procedure vizitate.init(x :integer);*

*{ initializam vectorul de vizitate cu false }*

*var i :integer;*

*begin*

*n := x;*

*for i := 1 to n do*

*vector[i] := False;*

*end;*

*procedure vizitate.vizitat(x :integer);*

*{ notam nodul <x> ca fiind vizitat }*

*begin*

*vector[x] := true;*

*end;*

*function vizitate.exista\_neviz:boolean;*

*{ verificam daca mai exista sau nu noduri nevizitate }*

*var i :integer;*

*begin*

*exista\_neviz := false;*

*for i := 1 to n do*

*if vector[i] = false then*

*begin*

*exista\_neviz := true;*

*break;*

*end;*

*end;*

*function vizitate.este\_neviz(x :integer):boolean;*

*{ verificam daca nodul <x> a fost sau nu vizitat }*

*begin*

*este\_neviz := not vector[x];*

*end;*

*{===================================================================================}*

*{ Metode obiect stiva }*

*procedure stiva.init;*

*{ initializam stiva }*

*begin*

*n := 0;*

*end;*

*procedure stiva.push(x :integer);*

*{ adaugam un element in stiva }*

*begin*

*n := n + 1;*

*vector[n] := x;*

*end;*

*procedure stiva.pop;*

*{ eliminam un vector din stiva }*

*begin*

*n := n - 1;*

*if vida() then { daca stiva devine vida }*

*init(); { o initializam }*

*end;*

*function stiva.acces:integer;*

*{ returnam elementul din capul stivei }*

*begin*

*acces := vector[n];*

*end;*

*function stiva.vida:boolean;*

*{ returnam <True> daca stiva e goala si false in caz contrar }*

*begin*

*vida := ( n <= 0 );*

*end;*

*{===================================================================================}*

*{ Metode obiect coada }*

*procedure coada.init;*

*{ initializam coada }*

*begin*

*i := 1;*

*n := 0;*

*end;*

*procedure coada.push(x :integer);*

*{ adaugam un element in coada }*

*begin*

*n := n + 1;*

*vector[n] := x;*

*end;*

*procedure coada.pop;*

*{ eliminam un vector din coada }*

*begin*

*i := i + 1;*

*if vida() then { daca stiva este vida }*

*init(); { o initializam }*

*end;*

*function coada.acces:integer;*

*begin*

*acces := vector[i];*

*end;*

*function coada.vida:boolean;*

*begin*

*vida := (i > n);*

*end;*

*{===================================================================================}*

*{ Metode obiect Graf }*

*procedure graf.initializare(x :integer);*

*{ initlializam toate componentele in functie de n }*

*var i, j :integer;*

*begin*

*n := x;*

*{ initlializam matricea de adiacenta cu False }*

*for i := 1 to n do*

*for j := 1 to n do*

*mat\_adiacenta[i, j] := False;*

*end;*

*procedure graf.initializare\_fisier(x :string);*

*begin*

*fisier := x;*

*end;*

*procedure graf.citire\_mat\_costuri;*

*{ citim matricea de costuri si initializam si matricea de adiacenta, si n o data cu asta }*

*var f :text;*

*x, i, j :integer;*

*begin*

*assign(f, fisier);*

*reset(f);*

*readln(f, x);*

*{ initlializam graful cu numarul de noduri <x> }*

*initializare(x);*

*for i := 1 to x do*

*begin*

*for j := 1 to x do*

*begin*

*read(f, mat\_costuri[i, j]);*

*if mat\_costuri[i, j] > 0 then*

*mat\_adiacenta[i, j] := True;*

*end;*

*readln(f);*

*end;*

*close(f);*

*n := x;*

*citit := true; { marcam graful ca fiind citit }*

*end;*

*procedure graf.config\_mat\_costuri(x :real);*

*{ configuram matricea costurilor, adica inlocuim <-1> cu x, care va fi <p\_inf> sau <m\_inf> }*

*var i, j :integer;*

*begin*

*for i := 1 to n do*

*for j := 1 to n do*

*if (mat\_costuri[i, j] = -1) OR (mat\_costuri[i, j] = p\_inf) OR (mat\_costuri[i, j] = m\_inf) then*

*mat\_costuri[i, j] := x;*

*end;*

*procedure graf.citire\_mat\_adiacenta;*

*var f :text;*

*aux, i, j, x :integer;*

*begin*

*assign(f, fisier);*

*reset(f);*

*readln(f, x);*

*{ initlializam elementele matrici in fuctie de <x> noduri }*

*initializare(x);*

*for i := 1 to x do*

*begin*

*for j := 1 to x do*

*begin*

*read(f, aux);*

*if aux = 1 then*

*mat\_adiacenta[i, j] := true;*

*end;*

*readln(f);*

*end;*

*close(f);*

*{ initliazizam matricea costurilor cu 1 }*

*for i := 1 to n do*

*for j := 1 to n do*

*if i = j then*

*mat\_costuri[i, j] := 0*

*else*

*if mat\_adiacenta[i, j] then*

*mat\_costuri[i, j] := 1*

*else*

*mat\_costuri[i, j] := -1;*

*citit := true; { marcam graful ca fiind citit }*

*end;*

*procedure graf.afisare;*

*var i, j :integer;*

*begin*

*writeln('Graful cu numele :', nume);*

*writeln('Are urmatoarele arce :');*

*for i := 1 to n do*

*for j := 1 to n do*

*if mat\_adiacenta[i, j] then*

*writeln(i, ' - ', j);*

*end;*

*procedure graf.init;*

*{ procedura care initializeaza un graf }*

*begin*

*nume := 'No Name';*

*fisier := '';*

*n := 0;*

*cauta\_roy := 'none';*

*cauta\_dj := 'none';*

*citit := false;*

*end;*

*procedure graf.set\_nume(s :string);*

*begin*

*nume := s;*

*end;*

*procedure graf.parcurgere\_in\_latime(x :integer);*

*var i, p :integer;*

*procedure prelucrare(nod :integer);*

*{ prelucram nodul <nod>, momentan doar in afisam }*

*begin*

*write(nod, ' ');*

*end;*

*begin*

*{ initializam coada }*

*coada.init;*

*{ initializam vectorul vizitate }*

*vizitate.init(n);*

*{ adaugam nodul <x> in coada }*

*coada.push(x);*

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

*writeln('Ii gasim urmatoarele noduri :');*

*while not coada.vida do*

*begin*

*p := coada.acces;*

*if vizitate.este\_neviz(p) then { nodul din varful stivei trebuie procesat }*

*begin*

*prelucrare(p);*

*vizitate.vizitat(p);*

*end*

*else*

*coada.pop;*

*for i := 1 to n do*

*if mat\_adiacenta[p, i] AND vizitate.este\_neviz(i) then*

*coada.push(i);*

*end;*

*writeln;*

*end;*

*procedure graf.parcurgere\_in\_adancime(x :integer);*

*var i, p :integer;*

*ok :boolean;*

*procedure prelucrare(nod :integer);*

*{ prelucram nodul <nod>, momentan doar in afisam }*

*begin*

*write(nod, ' ');*

*end;*

*begin*

*{ initializam stiva }*

*stiva.init;*

*{ initializam vectorul vizitate }*

*vizitate.init(n);*

*{ adaugam nodul <x> in stiva }*

*stiva.push(x);*

*(\* vizitate.vizitat(stiva.acces); \*)*

*writeln('Graful cu numele <', nume, '>, parcurs adancime de la nodul : ', x);*

*writeln('Ii gasim urmatoarele noduri :');*

*while not stiva.vida do*

*begin*

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

*if vizitate.este\_neviz(p) then { nodul din varful stivei trebuie procesat }*

*begin*

*prelucrare(p);*

*vizitate.vizitat(p);*

*end;*

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

*for i := 1 to n do*

*if mat\_adiacenta[p, i] AND vizitate.este\_neviz(i) then*

*begin*

*stiva.push(i); { adaugam nodul gasit in stiva }*

*ok := true; { memoram faptul ca am gasit un nod de la <p> }*

*break;*

*end;*

*if not ok then { daca nu am gasit nici un nod, eliminam nodul din stiva }*

*stiva.pop;*

*end;*

*writeln;*

*end;*

*procedure graf.set\_file(s :string);*

*{ setam un fisier de citire }*

*begin*

*fisier := s;*

*end;*

*procedure graf.afis\_mat\_adiacenta;*

*{ afisam matricea de adiacenta }*

*var i, j :integer;*

*begin*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*if mat\_adiacenta[i, j] then*

*write(' \* ')*

*else*

*write(' x ');*

*writeln;*

*end;*

*end;*

*procedure graf.afis\_mat\_costuri;*

*{ afisam matricea costurilor }*

*var i, j :integer;*

*begin*

*for i := 1 to n do*

*begin*

*for j := 1 to n do*

*write(' ',inf\_or\_nr(mat\_costuri[i, j]):8, ' ');*

*writeln;*

*end;*

*end;*

*procedure graf.afis\_lista\_noduri;*

*var i, j :integer;*

*begin*

*writeln('Graful cu numele ', nume, ' are urmatoarele noduri :');*

*for i := 1 to n do*

*for j := 1 to n do*

*if mat\_adiacenta[i, j] then*

*writeln(i, ' - ', j);*

*end;*

*procedure graf.afis\_lista\_noduri\_cost;*

*var i, j :integer;*

*begin*

*writeln('Graful cu numele ', nume, ' are urmatoarele noduri :');*

*for i := 1 to n do*

*for j := 1 to n do*

*if mat\_adiacenta[i, j] then*

*writeln(i, ' - ', j, ' : ', mat\_costuri[i, j]:4:2);*

*end;*

*procedure graf.roy\_floyd\_min;*

*var i, j, k :integer;*

*a, b, c :string;*

*begin*

*{ initializam matricea de costuri }*

*config\_mat\_costuri(p\_inf);*

*mat\_costuri\_roy := mat\_costuri;*

*{ initializam matricea de drumuri }*

*init\_mat\_drumuri\_min;*

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

*writeln;*

*writeln;*

*for i := 1 to n do*

*for j := 1 to n do*

*for k := 1 to n do*

*if (k <> i) AND (k <> j) then*

*begin*

*a := inf\_or\_nr(mat\_costuri\_roy[i, j]);*

*b := inf\_or\_nr(mat\_costuri\_roy[i, k]);*

*c := inf\_or\_nr(mat\_costuri\_roy[k, j]);*

*writeln('Vedem daca drumul direct de la ', i, ' la ', j, ' cu costul : ', a);*

*writeln('Este mai mic decat suma drumului de la ', i, ' la ', k, ' cu costul :', b);*

*writeln('cu drumul de la ', k, ' la ', j, ' cu costul :', c);*

*if (mat\_costuri\_roy[i, j] > mat\_costuri\_roy[i, k] + mat\_costuri\_roy[k, j]) then*

*begin*

*writeln('Costul fiind mai mic, modificam matricea drumurilor si a costurilir !');*

*mat\_costuri\_roy[i, j] := mat\_costuri\_roy[i, k] + mat\_costuri\_roy[k, j];*

*mat\_drum[i, j] := concat\_drum(mat\_drum[i, k], mat\_drum[k, j]);*

*end;*

*writeln;*

*end;*

*writeln;*

*writeln;*

*writeln('Afisam drumurile minime obtinute folosind algoritmul lui Roy-Floyd :');*

*afis\_drum\_roy('min');*

*cauta\_roy := 'min';*

*end;*

*procedure graf.roy\_floyd\_max;*

*var i, j, k :integer;*

*a, b, c :string;*

*begin*

*{ initializam matricea de costuri }*

*config\_mat\_costuri(m\_inf);*

*mat\_costuri\_roy := mat\_costuri;*

*{ initializam matricea de drumuri }*

*init\_mat\_drumuri\_max;*

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

*writeln;*

*writeln;*

*for i := 1 to n do*

*for j := 1 to n do*

*for k := 1 to n do*

*begin*

*if (k <> i) AND (k <> j) AND (i <> j) AND (mat\_costuri\_roy[i, k] > m\_inf) AND (mat\_costuri\_roy[k, j] > m\_inf) then*

*begin*

*a := inf\_or\_nr(mat\_costuri\_roy[i, j]);*

*b := inf\_or\_nr(mat\_costuri\_roy[i, k]);*

*c := inf\_or\_nr(mat\_costuri\_roy[k, j]);*

*writeln('Vedem daca drumul direct de la ', i, ' la ', j, ' cu costul : ', a);*

*writeln('Este mai mare decat suma drumului de la ', i, ' la ', k, ' cu costul :', b);*

*writeln('cu drumul de la ', k, ' la ', j, ' cu costul :', c);*

*if (mat\_costuri\_roy[i, j] < mat\_costuri\_roy[i, k] + mat\_costuri\_roy[k, j]) then*

*begin*

*writeln('Costul fiind mai maxim, modificam matricea drumurilor si a costurilir !');*

*mat\_costuri\_roy[i, j] := mat\_costuri\_roy[i, k] + mat\_costuri\_roy[k, j];*

*mat\_drum[i, j] := concat\_drum(mat\_drum[i, k], mat\_drum[k, j]);*

*end;*

*writeln;*

*end;*

*end;*

*writeln;*

*writeln;*

*writeln('Afisam drumurile maxime obtinute folosind algoritmul lui Roy-Floyd :');*

*afis\_drum\_roy('max');*

*cauta\_roy := 'max';*

*end;*

*procedure graf.dijkstra\_min(x :integer);*

*var p, i, count :integer;*

*min :real;*

*s :string;*

*begin*

*count := 0;*

*config\_mat\_costuri(p\_inf);*

*mat\_costuri\_dj := mat\_costuri; { duplicam matricea costurilor }*

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

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

*init\_vector\_drum(x); { initializam vectorul drumrilor pentru nodul x }*

*while vizitate.exista\_neviz() AND ( count < n) do*

*begin*

*min := p\_inf; { pe care il stim momentan }*

*p := 0;*

*for i := 1 to n do { cautam nodul la care putem ajunge cel mai usor de la nodul de pornire }*

*if vizitate.este\_neviz(i) AND (mat\_costuri\_dj[x, i] < min ) then*

*begin*

*min := mat\_costuri\_dj[x ,i];*

*p := i;*

*end;*

*vizitate.vizitat(p);*

*for i := 1 to n do*

*if vizitate.este\_neviz(i) AND mat\_adiacenta[p, i] then*

*if mat\_costuri\_dj[x, i] > mat\_costuri\_dj[x, p] + mat\_costuri[p, i] then*

*begin*

*mat\_costuri\_dj[x, i] := mat\_costuri\_dj[x, p] + mat\_costuri[p, i];*

*matrice\_drum\_dj[x, i] := p;*

*end;*

*count := count + 1;*

*end;*

*writeln(' Drumurile minime de la nodul ', x, ' la restul nodurilor din graf :');*

*afis\_drum\_dj(x);*

*str(x, s);*

*cauta\_dj := cauta\_dj + s;*

*end;*

*procedure graf.afis\_drum\_roy(s :string);*

*var i, j, k :integer;*

*aux :drum;*

*begin*

*if s = 'min' then*

*begin*

*for i := 1 to n do*

*for j := 1 to n do*

*if i <> j then*

*if (mat\_costuri\_roy[i, j] <> p\_inf) then*

*begin*

*writeln('Intre nodurile ', i, ' si ', j, ' drumul are costul : ', mat\_costuri\_roy[i, j]:4:2);*

*writeln('Drumul este :');*

*aux := mat\_drum[i, j];*

*write(' ':5);*

*for k := 1 to aux.n do*

*write(aux.v[k], ' ');*

*writeln;*

*writeln;*

*end*

*else*

*writeln('Intre nodurile ', i, ' si ', j, ' nu exista drum !');*

*end*

*else*

*begin*

*for i := 1 to n do*

*for j := 1 to n do*

*if i <> j then*

*if (mat\_costuri\_roy[i, j] <> m\_inf) then*

*begin*

*writeln('Intre nodurile ', i, ' si ', j, ' drumul are costul : ', mat\_costuri\_roy[i, j]:4:2);*

*writeln('Drumul este :');*

*aux := mat\_drum[i, j];*

*write(' ':5);*

*for k := 1 to aux.n do*

*write(aux.v[k], ' ');*

*writeln;*

*writeln;*

*end*

*else*

*writeln('Intre nodurile ', i, ' si ', j, ' nu exista drum !');*

*end;*

*end;*

*procedure graf.afis\_drum\_dj(x :integer);*

*var i :integer;*

*begin*

*for i := 1 to n do*

*if i <> x then*

*if (matrice\_drum\_dj[x, i] <> 0) then*

*begin*

*writeln('Drumul de la ', x, ' la ', i ,' are costul :', mat\_costuri\_dj[x, i]:4:2);*

*writeln('Si este compus din :');*

*write(' ':7);*

*afis\_dr(x, i);*

*writeln;*

*end*

*else*

*writeln('Nu exista drum de la ', x, ' la ', i, ' !');*

*end;*

*procedure graf.drum\_min(a, b :integer);*

*var i :integer;*

*s :string;*

*begin*

*str(a, s);*

*if cauta\_roy = 'min' then*

*begin*

*writeln('Drumul minim a fost folosit deja folosind algoritmul roy-floyd');*

*writeln('Drumul minim este :');*

*if mat\_costuri\_roy[a, b] < p\_inf then*

*for i := 1 to mat\_drum[a, b].n do*

*write(mat\_drum[a, b].v[i], ' ')*

*else*

*writeln(' Din pacate nu se poate ajunge de la ', a, ' la ', b);*

*writeln;*

*end*

*else*

*if pos(s, cauta\_dj) <> 0 then*

*begin*

*writeln('Drumul minim a fost folosit deja folosind algoritmul dijkstra ');*

*if (mat\_costuri\_dj[a, b] <> 0) AND (mat\_costuri\_dj[a, b] < p\_inf) then*

*begin*

*writeln('Drumul de la ', a, ' la ', b ,' are costul :', mat\_costuri\_dj[a, b]:4:2);*

*writeln('Si este compus din :');*

*write(' ':7);*

*afis\_dr(a, b);*

*end*

*else*

*writeln('Nu exista drum de la ', a, ' la ', b, ' !');*

*end*

*else*

*begin*

*writeln('Drumul minim nu a fost deja calculat asa ca il vom calcula folosind aglgoritmul dikstra ');*

*dijkstra\_min(a);*

*writeln;*

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

*writeln;*

*if (mat\_costuri\_dj[a, b] <> 0) AND (mat\_costuri\_dj[a, b] < p\_inf) then*

*begin*

*writeln('Drumul de la ', a, ' la ', b ,' are costul :', mat\_costuri\_dj[a, b]:4:2);*

*writeln('Si este compus din :');*

*write(' ':7);*

*afis\_dr(a, b);*

*end*

*else*

*writeln('Nu exista drum de la ', a, ' la ', b, ' !');*

*end;*

*writeln;*

*end;*

*procedure graf.drum\_max(a, b :integer);*

*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 aglgoritmul 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 citit, matricea de adiacenta,');*

*writeln(' sau matricea costurilor grafului.');*

*writeln(' Exemplu de utilizare:');*

*writeln(' set\_file(graf.txt)');*

*writeln(' set\_file(costuri.in)');*

*end;*

*'citire\_mat\_cost' :begin*

*writeln(' citire\_mat\_cost');*

*writeln(' Cu ajutorul acestei comenzi puteti citi matricea costurilor din fiseirul');*

*writeln(' setat cu ajutorul comenzi <set\_file(arg)> !');*

*writeln(' Exemplu de utilizare:');*

*writeln(' citire\_mat\_cost');*

*end;*

*'citire\_mat\_adj' :begin*

*writeln(' citire\_mat\_adj');*

*writeln(' Cu ajutorul acestei comenzi puteti citi matricea de adiacenta din fiseirul');*

*writeln(' setat cu ajutorul comenzi <set\_file(arg)> !');*

*writeln(' Exemplu de utilizare:');*

*writeln(' citire\_mat\_adj');*

*end;*

*'parcurge\_adancime':begin*

*writeln(' parcurgere\_adancime(arg)');*

*writeln(' Cu ajutorul acestei comenzi puteti parcurge graful in adancime,' );*

*writeln(' graful, pornind de la nodul specificat ca si argument .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' parcurge\_adancime(2)');*

*writeln(' parcurge\_adancime(33)');*

*end;*

*'parcurge\_latime':begin*

*writeln(' parcurgere\_latime(arg)');*

*writeln(' Cu ajutorul acestei comenzi puteti parcurge graful in latime,' );*

*writeln(' graful, pornind de la nodul specificat ca si argument .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' parcurge\_latime(2)');*

*writeln(' parcurge\_latime(33)');*

*end;*

*'afisare\_muchii':begin*

*writeln(' afisare\_muchii');*

*writeln(' Cu ajutorul acestei comenzi putem afisa muchiile grafului stocat momentan .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' afisare\_muchii');*

*end;*

*'afisare\_muchii\_cost':begin*

*writeln(' afisare\_muchii\_cost');*

*writeln(' Cu ajutorul acestei comenzi putem afisa muchiile grafului stocat momentan si ');*

*writeln(' costul asociat fiecarei muchii .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' afisare\_muchii');*

*end;*

*'afis\_mat\_cost':begin*

*writeln(' afis\_mat\_cost');*

*writeln(' Cu ajutorul acestei comenzi putem afisa matricea costurilor asociata grafului. ');*

*writeln(' Exemplu de utilizare:');*

*writeln(' afis\_mat\_adj');*

*end;*

*'afis\_mat\_adj':begin*

*writeln(' afis\_mat\_adj');*

*writeln(' Cu ajutorul acestei comenzi putem afisa matreia de adiacenta a graului .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' afis\_mat\_adj');*

*end;*

*'roy\_floyd\_max':begin*

*writeln(' roy\_floyd\_max');*

*writeln(' Cu ajutorul acestei comenzi cautam drumurile maxiem de la orce nod din graf la restul nodurilor');*

*writeln(' cu ajutorul algoritmului lui Roy-Floyd.');*

*writeln(' Exemplu de utilizare:');*

*writeln(' roy\_floyd\_max');*

*end;*

*'roy\_floyd\_min': begin*

*writeln(' roy\_floyd\_min');*

*writeln(' Cu ajutorul acestei comenzi cautam drumurile minime de la orce nod din graf la restul nodurilor');*

*writeln(' cu ajutorul algoritmului lui Roy-Floyd.');*

*writeln(' Exemplu de utilizare:');*

*writeln(' roy\_floyd\_min');*

*end;*

*'dijkstra\_max':begin*

*writeln(' dijkstra\_max(arg)');*

*writeln(' Cu ajutorul acestei comenzi aflam drumurile maxime de la nodul primit ca si argument ');*

*writeln(' si restul nodurilor din graf .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' dijkstra\_max(1)');*

*writeln(' dijkstra\_max(4)');*

*end;*

*'dijkstra\_min':begin*

*writeln(' dijkstra\_min(arg)');*

*writeln(' Cu ajutorul acestei comenzi aflam drumurile minime de la nodul primit ca si argument ');*

*writeln(' si restul nodurilor din graf .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' dijkstra\_min(1)');*

*writeln(' dijkstra\_min(4)');*

*end;*

*'drum\_max':begin*

*writeln(' drum\_max(arg, arg)');*

*writeln(' Cu ajutorl acestei comenzi vom afisa drumul maxim de la nodul primit ca prim argument ');*

*writeln(' pana la nodul primit ca al doilea argument. Programul va verifica daca cautarea nu a fost deja ');*

*writeln(' facuta. Daca nu a fost facuta, va cauta drumul folosit algoritmul 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 ajutorl acestei comenzi vom afisa drumul minin de la nodul primit ca prim argument ');*

*writeln(' pana la nodul primit ca al doilea argument. Programul va verifica daca cautarea nu a fost deja ');*

*writeln(' facuta. Daca nu a fost facuta, va cauta drumul folosit algoritmul Dijkstra .');*

*writeln(' Exemplu de utilizare:');*

*writeln(' drum\_min(1, 2);');*

*writeln(' drum\_min(3, 6);');*

*end;*

*'credit':begin*

*writeln(' credit');*

*writeln(' Aceasta comanda va afisa informatii utile despre autorul programului, utilizare, menire si licenta ');*

*end;*

*end;*

*writeln;*

*end;*

*procedure info;*

*begin*

*writeln;*

*writeln(' Drumuri Minime si Maxime in Grafuri orientate ');*

*writeln(' ');*

*writeln(' ');*

*writeln(' Accesati comanda <help> pentru mai multe informatii !');*

*writeln(' ');*

*writeln(' ');*

*writeln;*

*end;*

*procedure clear;*

*{ curatam ecranul }*

*begin*

*clrscr;*

*end;*

*procedure set\_file(s :string);*

*{ setam un fisier de citire }*

*var a, b :string;*

*ok :boolean;*

*begin*

*split\_args(s, a, b, ok);*

*if ok then*

*begin*

*writeln('Am setat fisierul sursa ca fiind :', b);*

*gf.set\_file(b);*

*end;*

*end;*

*procedure set\_nume(s :string);*

*{ setam un numele grafului }*

*var a, b :string;*

*ok :boolean;*

*begin*

*split\_args(s, a, b, ok);*

*if ok then*

*begin*

*writeln('Am setat numele grafului ca fiind :', b);*

*gf.set\_nume(b);*

*end;*

*end;*

*procedure citire\_mat\_adiacenta;*

*begin*

*writeln;*

*if gf.fisier <> '' then*

*begin*

*writeln('Citim matricea de adiacenta din fisierul : ', gf.fisier);*

*gf.citire\_mat\_adiacenta;*

*gf.citit := true;*

*writeln('Citirea a fost facuta cu succes !');*

*end*

*else*

*begin*

*writeln('ERROARE:');*

*writeln('Nu ati selectat un fisier de unde putem citit matricea de adiacenta !');*

*writeln('Folositi comanda <set\_file(arg)> pentru a seta unul .');*

*writeln('Pentru mai multe informatii folosit comanda <help> .');*

*end;*

*writeln;*

*end;*

*procedure citire\_mat\_costuri;*

*begin*

*writeln;*

*if gf.fisier <> '' then*

*begin*

*writeln('Citim matricea costurilor din fisierul : ', gf.fisier);*

*gf.citire\_mat\_costuri;*

*gf.citit := true;*

*writeln('Citirea a fost facuta cu succes !');*

*end*

*else*

*begin*

*writeln('ERROARE:');*

*writeln('Nu ati selectat un fisier de unde putem citit matricea costurilor !');*

*writeln('Folositi comanda <set\_file(arg)> pentru a seta unul .');*

*writeln('Pentru mai multe informatii folosit comanda <help> .');*

*end;*

*writeln;*

*end;*

*procedure parcurge\_latime(s :string);*

*{ parcurgem in latime graful }*

*var a, b :string;*

*ok :boolean;*

*i :integer;*

*begin*

*split\_args(s, a, b, ok); { impartim argumentele }*

*i := to\_int(b); { transformam argumentu in integer }*

*writeln;*

*if i > 0 then*

*begin*

*writeln;*

*gf.parcurgere\_in\_latime(i);*

*writeln;*

*writeln('Am terminat de parcurs !');*

*end*

*else*

*begin*

*writeln('ERROARE:');*

*writeln(' A aparut o eroare incercand sa parcurgem graful in latime. ');*

*writeln(' Folotisi comanda <help> pentru a vede mai multe informatii .');*

*end;*

*writeln;*

*end;*

*procedure parcurge\_adancime(s :string);*

*{ parcurgem in latime graful }*

*var a, b :string;*

*ok :boolean;*

*i :integer;*

*begin*

*split\_args(s, a, b, ok); { impartim argumentele }*

*i := to\_int(b); { transformam argumentu in integer }*

*writeln;*

*if i > 0 then*

*begin*

*writeln;*

*gf.parcurgere\_in\_adancime(i);*

*writeln;*

*writeln('Am terminat de parcurs !');*

*end*

*else*

*begin*

*writeln('ERROARE:');*

*writeln(' A aparut o eroare incercand sa parcurgem graful in adancime. ');*

*writeln(' Folotisi comanda <help> pentru a vede mai multe informatii .');*

*end;*

*writeln;*

*end;*

*procedure afisare\_muchii(inp :string);*

*begin*

*if (gf.n > 0) then*

*gf.afis\_lista\_noduri*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosnutlati comanda <help> ')*

*end;*

*end;*

*procedure afisare\_muchii\_cost(inp :string);*

*begin*

*if (gf.n > 0) then*

*gf.afis\_lista\_noduri\_cost*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosnutlati comanda <help> ')*

*end;*

*end;*

*procedure afis\_mat\_adj(inp :string);*

*begin*

*if gf.citit then*

*gf.afis\_mat\_adiacenta*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure afis\_mat\_cost(inp :string);*

*begin*

*if gf.citit then*

*gf.afis\_mat\_costuri*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure roy\_floyd\_min(inp :string);*

*var a, b :string;*

*ok :boolean;*

*begin*

*split\_args(inp, a, b, ok);*

*if gf.citit then*

*begin*

*gf.roy\_floyd\_min;*

*writeln;*

*writeln(' Am terminat de aflat drumurile minime . . . ');*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure roy\_floyd\_max(inp :string);*

*var a, b :string;*

*ok :boolean;*

*begin*

*split\_args(inp, a, b, ok);*

*if gf.n > 0 then*

*begin*

*gf.roy\_floyd\_max;*

*writeln;*

*writeln(' Am terminat de aflat drumurile maxime. . . ');*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure dijkstra\_min(inp :string);*

*var a, b :string;*

*ok :boolean;*

*n :integer;*

*begin*

*split\_args(inp, a, b, ok);*

*n := to\_int(b);*

*if gf.n > 0 then*

*if n > 0 then*

*begin*

*gf.dijkstra\_min(n);*

*writeln(' Am terminat de aflat drumurile minime de la nodul ', n, ' la restul nodurilor . . . ');*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Parametrul transmis functiei este incorect, folositi comadna <help> ');*

*writeln(' pentru mai multe informatii .');*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure drum\_max(inp :string);*

*var a, b, c, d :string;*

*x, y :integer;*

*ok1, ok2 :boolean;*

*begin*

*split\_args(inp, a, b, ok1);*

*split\_args2(b, c, d, ok2);*

*x := to\_int(c);*

*y := to\_int(d);*

*if gf.citit then*

*begin*

*if ok2 AND (x > 0) AND (x <= gf.n) AND (y > 0) AND (y <= gf.n) then*

*begin*

*writeln('Cautam drumul maxim de la ', x, ' la ', y, ' :');*

*gf.drum\_max(x, y);*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu ati transmis argumente corect sau sunt invalide , cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure drum\_min(inp :string);*

*var a, b, c, d :string;*

*x, y :integer;*

*ok1, ok2 :boolean;*

*begin*

*split\_args(inp, a, b, ok1);*

*split\_args2(b, c, d, ok2);*

*x := to\_int(c);*

*y := to\_int(d);*

*if gf.citit then*

*begin*

*if ok2 AND (x > 0) AND (x <= gf.n) AND (y > 0) AND (y <= gf.n) then*

*begin*

*writeln('Cautam drumul minim de la ', x, ' la ', y, ' :');*

*gf.drum\_min(x, y);*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu ati transmis argumente corect sau sunt invalide , cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end*

*else*

*begin*

*writeln(' ERROARE:');*

*writeln(' Nu avem nici un graf citit, cosultati comanda <help> ');*

*writeln(' pentru mai multe informatii .');*

*end;*

*end;*

*procedure credit(inp :string);*

*begin*

*writeln('');*

*writeln(' Drumuri Minime si Maxime in grafuri Orientate ');*

*writeln('');*

*writeln(' Versiunea: 3.0.1');*

*writeln('');*

*writeln(' Autor : Micu Matei-Marius');*

*writeln(' Email : matei10@yahoo.com');*

*writeln(' Gmail : micumatei@gmail.com');*

*writeln(' GitHub : matei10');*

*writeln('');*

*writeln(' Licenta : MIT ');*

*writeln(' O copie a licente MIT ar trebui sa fie distribuita o data cu programul ');*

*writeln('');*

*writeln(' Descriere :');*

*writeln(' Programul aduna o serie de algoritmi intr-un singur loc pentru o utilizare mai usoara ');*

*writeln('');*

*writeln(' Algoritmi :');*

*writeln(' - parcurgerea in adancime a unui graf orientat');*

*writeln(' - parcurgerea in latime a unui graf orientat ');*

*writeln(' - algoritmul Roy-Floyd');*

*writeln(' - algoritmul Dijkstra');*

*writeln(' - o serie de algoritmi auxiliari pentru cititre si afisare a matricilor de adiacenta si cea a costurilor !');*

*writeln('');*

*writeln('');*

*end;*

*{===================================================================================}*

*{ Metode de managereriere }*

*procedure start;*

*{ ascultam pentru inputul utilizatorului }*

*var inp :string;*

*ok :boolean;*

*begin*

*write('>> ');*

*readln(inp);*

*while not check\_close(inp) do*

*begin*

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

*{ verificam daca utilizatorul cere informatii }*

*if check\_info(inp) then*

*begin*

*info;*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa curatam ecranul }*

*if check\_clear(inp) then*

*begin*

*clear;*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa seteze un fisier de citire }*

*if check\_set\_file(inp) then*

*begin*

*set\_file(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste ajutor }*

*if check\_help(inp) then*

*begin*

*help(inp);*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa citeasca matricea de adiacenta }*

*if check\_citire\_adj(inp) then*

*begin*

*citire\_mat\_adiacenta;*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa citeasca matricea de costuri }*

*if check\_citire\_cost(inp) then*

*begin*

*citire\_mat\_costuri;*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa parcurga graful in latime }*

*if check\_parcurge\_latime(inp) then*

*begin*

*parcurge\_latime(inp);*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa parcurga graful in adancime }*

*if check\_parcurge\_adancime(inp) then*

*begin*

*parcurge\_adancime(inp);*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa afisam muchiile grafului }*

*if check\_afisare\_muchii(inp) then*

*begin*

*afisare\_muchii(inp);*

*ok := true;*

*end;*

*{ verificam daca userul vrea sa afisam muchiile grafului si costul asociat }*

*if check\_afisare\_muchii\_cost(inp) then*

*begin*

*afisare\_muchii\_cost(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa afisam matricea de adiacenta }*

*if check\_afis\_mat\_adj(inp) then*

*begin*

*afis\_mat\_adj(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa afisam matricea costurilor asociate }*

*if check\_afis\_mat\_cost(inp) then*

*begin*

*afis\_mat\_cost(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa foloseasca algoritmul roy-floyd pentru drumuri minime }*

*if check\_roy\_floyd\_min(inp) then*

*begin*

*roy\_floyd\_min(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa foloseasca algoritmul roy-floyd pentru drumuri maxime}*

*if check\_roy\_floyd\_max(inp) then*

*begin*

*roy\_floyd\_max(inp);*

*ok := true;*

*end;*

*{ verificam daca userul doreste sa afle durmurile minime de la un nod anume folosid algoritmul dijkstra }*

*if check\_dijkstra\_min(inp) then*

*begin*

*dijkstra\_min(inp);*

*ok := true;*

*end;*

*{ verificam daca utilizatorul doreste sa afisam drumul minim dintre doua noduri }*

*if check\_drum\_min(inp) then*

*begin*

*drum\_min(inp);*

*ok := true;*

*end;*

*{ verificam daca utilizatorul doreste sa afisam drumul maxi dintre doua noduri }*

*if check\_drum\_max(inp) then*

*begin*

*drum\_max(inp);*

*ok := true;*

*end;*

*{ daca se doreste afisarea creditelor }*

*if check\_credit(inp) then*

*begin*

*credit(inp);*

*ok := true;*

*end;*

*{ daca s-a introdus o linie goala }*

*if check\_blank(inp) then*

*begin*

*ok := true;*

*end;*

*{ 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](#h.q5lcbk450pkn) 1

[B. Metode de reprezentare a grafului orientat](#h.woqjxsuyk4q2) 2

[C. Noțiunea de graf parțial](#h.amsjj4dpbkdx) 3

[D. Noțiunea de subgraf](#h.aha9umerat72) 5

[E. Gradul unui nod](#h.2h2yeyf3nu4e) 7

[G. Metode de parcurgere a grafurilor orientate](#h.whpja3qs6ssi) 17

[H. Drumuri minime și maxime](#h.s74e1d8aubm9) 25

[Algoritmul Roy-Floyd](#h.bqqcqzfi9d27) 27

[Algoritmul Dijkstra](#h.sugkda6rdjn) 35

[Program auxiliar](#h.5r8t7to3v1nz) 41

[I. Program Complex](#h.nhi28w6q7dro) 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**](http://www.draw.io)

**Siteuri utilizare:**

[**http://stackoverflow.com/**](http://stackoverflow.com/)

[**https://github.com/**](https://github.com/)

[**www.draw.io**](http://www.draw.io)

[**https://docs.google.com/**](https://docs.google.com/)

**Toata lucrarea și programele pot fi găsite pe :**

[**https://github.com/matei10/atestat-2015**](https://github.com/matei10/atestat-2015)

**Autor : Micu Matei-Marius**

**Email :** [**matei10@yahoo.com**](mailto:matei10@yahoo.com)

**Gmail :** [**micumatei@gmail.com**](mailto:micumatei@gmail.com)

**GitHub : https://github.com/matei10**

**StackOverflow : http://stackoverflow.com/users/4311994/matei**