

3

GRAFURI. IMPLEMENTĂRI ÎN LIMBAJUL PASCAL

Grafurile sunt structuri de date cu aplicații în multe domenii ale prelucrării automate a datelor, algoritmi pentru reprezentarea și prelucrarea grafurilor fiind considerați fundamentali în acest domeniu.

În cadrul secțiunii 3.1 sunt prezentate principalele caracteristici ale grafurilor, precum și modalitățile uzuale de reprezentare a structurii de graf. În continuare sunt descrise tehnicile de parcurgere a grafurilor în lățime și în adâncime. Verificarea conexității și calculul drumurilor în grafuri sunt tratate în secțiunea 3.3

3.1 Definiții, caracteristici și reprezentări ale grafurilor

Definiția 3.1.1. Un *graf* (sau un *graf neorientat*) este o structură $G=(V,E)$, unde V este o mulțime nevidă, iar E este o submulțime (posibil vidă) a mulțimii perechilor neordonate cu componente distincte din V .

Obiectele mulțimii V se numesc *vârfuri*, iar obiectele mulțimii E se numesc *muchii*. Dacă $e \in E$, $e = (u, v) = uv$, se spune că muchia e are ca extremități u, v sau că muchia e este determinată de vârfurile u și v . Dacă $e=uv \in E$ se spune că vârfurile u, v sunt incidente cu muchia e .

Definiția 3.1.2. Fie $G = (V,E)$ graf. Vârfurile u, v sunt *adiacente* în G dacă $uv \in E$.

Definiția 3.1.3. Graful $G = (V,E)$ este *finit*, dacă V este o mulțime finită.

În cadrul acestui capitol vor fi considerate în exclusivitate grafurile finite, chiar dacă acest lucru nu va fi precizat în mod explicit.

Definiția 3.1.4. Fie $G_i = (V_i, E_i)$, $i=1,2$ grafuri. G_2 este un *subgraf* al grafului G_1 dacă $V_2 \subseteq V_1$ și $E_2 \subseteq E_1$. Dacă G_2 este un subgraf al lui G_1 , G_2 este un *graf parțial* al lui G_1 dacă $V_2=V_1$.

Definiția 3.1.5. Un *digraf* este o structură $D = (V, E)$, unde V este o mulțime nevidă de obiecte numite convențional vârfuri, iar E este o mulțime (posibil vidă) de perechi ordonate cu componente elemente distincte din V . Convențional, elementele mulțimii E sunt numite *arce* sau *muchii ordonate*.

Terminologia utilizată relativ la digrafuri este similară celei corespunzătoare grafurilor.

Definiția 3.1.6. Se numește *graf ponderat* o structură (V, E, W) , unde $G = (V, E)$ este graf, W funcție, $W : E \rightarrow (0, \infty)$. Funcția W este numită *pondere* și ea asociază fiecărei muchii a grafului un cost/câștig al parcurgerii ei.

Definiția 3.1.7. Fie $G=(V, E)$ un graf, $u, v \in V$.

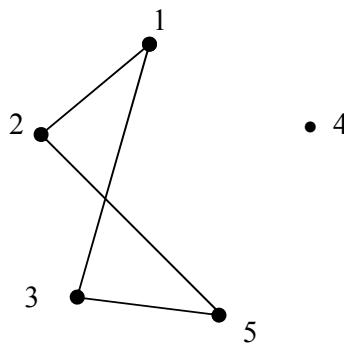
Secvența de vârfuri $\Gamma: u_0, u_1, \dots, u_n$ este un *u-v drum* dacă $u_0=u$, $u_n=v$, $u_i u_{i+1} \in E$ pentru toți i , $0 \leq i \leq n$.

Moduri de reprezentare a grafurilor

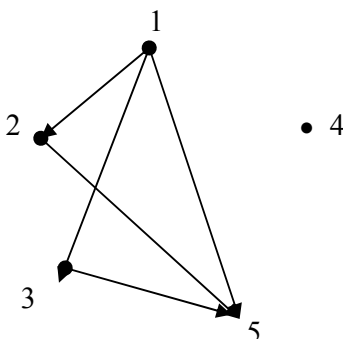
Cea mai simplă reprezentare a unui graf este cea intuitivă, *grafică*; fiecare vârf este figurat printr-un punct, iar muchiile sunt reprezentate prin segmentele de dreaptă, orientate (în cazul digrafurilor) sau nu și etichetate (în cazul grafurilor ponderate) sau nu, având ca extremități punctele corespunzătoare vârfurilor care le determină.

Exemple:

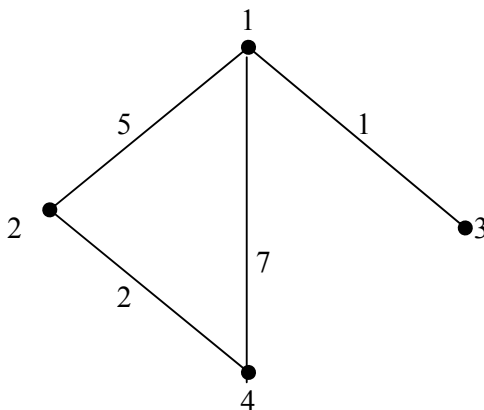
3.1. Fie $G = (V, E)$ graf, cu $V = \{1, 2, 3, 4, 5\}$, $E = \{(1,2), (1,3), (2,5), (3,5)\}$. O posibilă reprezentare grafică este:



3.2. Fie $D = (V, E)$ digraf, cu $V = \{1, 2, 3, 4, 5\}$, $E = \{(1,2), (1,3), (2,5), (3,5), (1,5)\}$. Digraful poate fi reprezentat grafic astfel:



3.3. Fie $G = (V, E, W)$ graf ponderat, cu $V = \{1, 2, 3, 4\}$, $E = \{(1,2), (1,3), (2,4), (3,4)\}$, $W((1,2))=5$, $W((1,3))=1$, $W((2,4))=2$, $W((3,4))=7$. O posibilă reprezentare grafică este:



Deși acest mod de reprezentare este foarte comod și sugestiv, în special în cazul grafurilor cu număr mic de vârfuri, pentru prelucrări cu ajutorul calculatorului sunt necesare reprezentări prin intermediul structurilor de date.

O modalitate de reprezentare este cea prin *matrice de adiacență*. Dacă $G=(V,E)$ este graf sau digraf cu $|V| = n$, atunci matricea de adiacență $A \in M_{n \times n}(\{0,1\})$ are componentele:

$$a_{ij} = \begin{cases} 1, & \text{dacă } (v_i, v_j) \in E \\ 0, & \text{altfel} \end{cases},$$

unde v_i, v_j reprezintă cel de-al i -lea, respectiv cel de-al j -lea nod din V . Se observă că, în cazul unui graf neorientat, matricea de adiacență este simetrică - $\forall i, j = \overline{1, n}, a_{ij} = a_{ji}$ (perechile de vârfuri ce caracterizează muchiile sunt neordonate, deci dacă $uv \in E$, atunci și $vu \in E$), în timp ce, în cazul unui digraf, este posibil ca $(v_i, v_j) \in E, (v_j, v_i) \notin E$, deci $a_{ij} \neq a_{ji}$.

Exemplu:

3.4. Graful din exemplul 3.1 și digraful din exemplul 3.2 sunt reprezentate prin matricele de adiacență:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \text{ pentru 3.1, } A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ pentru 3.2}$$

În cazul grafurilor ponderate, reprezentarea matriceală este asemănătoare celei prezentate anterior. *Matricea ponderilor* unui graf ponderat $G = (V, E, W)$, $|V| = n$, $W \in M_{n \times n}((0, \infty))$ are componentele:

$$w_{i,j} = \begin{cases} W((v_i, v_j)), & \text{dacă } (v_i, v_j) \in E \\ \alpha, & \text{altfel} \end{cases},$$

unde v_i, v_j reprezintă cel de-al i -lea, respectiv cel de-al j -lea nod din V , $\alpha = 0$ dacă ponderea are semnificația de câștig, respectiv $\alpha = \infty$ în cazul în care se dorește reprezentarea costurilor ca ponderi ale grafului.

Exemplu:

3.5. Presupunând că ponderile reprezintă costuri, matricea de reprezentare a grafului din exemplul 3.3. este:

$$W = \begin{pmatrix} \infty & 5 & 1 & 7 \\ 5 & \infty & \infty & 2 \\ 1 & \infty & \infty & \infty \\ 7 & 2 & \infty & \infty \end{pmatrix}.$$

Reținând numai “informația utilă”, și anume existența unei muchii între două vârfuri și eventual valoarea ponderii ei, se obține reprezentarea *tabelară*, mai economică din punctul de vedere al spațiului de memorare. În cazul în care există vârfuri izolate în graf (ce nu sunt incidente cu nici o muchie), atunci este necesară păstrarea acestora într-un vector suplimentar. Mulțimea muchiilor se memorează

într-o matrice cu $|E|$ linii și 2 coloane dacă graful nu este ponderat, respectiv cu 3 coloane, dacă graful este ponderat. În primele două coloane se scriu perechile de vârfuri ce determină muchiile, în cazul grafurilor ponderate cea de-a treia coloană conține valoarea ponderii muchiei respective.

Exemple:

3.6. Graful din exemplul 3.1 poate fi reprezentat astfel: deoarece 4 este vârf izolat, vectorul suplimentar este $VS = (4)$, pentru reprezentarea muchiilor fiind

utilizată matricea $A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 5 \\ 3 & 5 \end{pmatrix}$

3.7. Digraful din exemplul 3.2 poate fi reprezentat astfel: deoarece 4 este vârf izolat, vectorul suplimentar este $VS = (4)$, arcele fiind reprezentate prin

$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \\ 2 & 5 \\ 3 & 5 \end{pmatrix}$

O altă reprezentare este prin intermediul *listelor*. Reprezentarea permite utilizarea economică a spațiului de memorare și, în anumite cazuri, implementări mai eficiente pentru anumite clase de algoritmi. Vârfurile grafului se memorează într-o listă, fiecare celulă a listei având o legătură către lista vecinilor aceluia vârf (vârfurile din graf adiacente cu vârfurile corespunzătoare acelei celule și indicat ca informație utilă).

În situația în care graful nu este ponderat, el se reprezintă printr-o listă de liste, și anume: nodurile grafului se trec într-o listă L_nod , fiecare celulă având structura

informație	legătură vecini	legătură nod următor
------------	-----------------	----------------------

unde:

- câmpul *informație* conține identificatorul nodului;
- *legătură vecini* reprezintă pointer către capul listei vecinilor;
- *legătură nod următor* conține adresa următoarei celule din lista L_nod .

Un graf ponderat poate fi reprezentat în mod similar, cu diferența că fiecare celulă din lista vecinilor conține și ponderea muchiei respective (muchia care are ca extremități vârfurile referite prin identificatorul de nod din lista vecinilor și respectiv vârfurile indicate de informația acelei celule din L_nod ce conține adresa primului element al listei vecinilor).

3.2 Modalități de parcurgere a grafurilor

Parcurgerea unui graf reprezintă o modalitate de vizitare a tuturor vârfurilor grafului, fiecare vârf fiind vizitat o singură dată. În acest paragraf sunt prezentate două modalități de parcurgere a grafurilor neorientate.

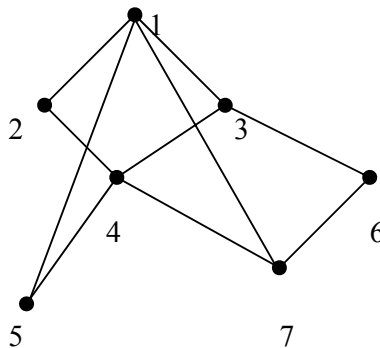
Ambele metode de parcurgere presupun selectarea unui vârf inițial v_0 . Prin aplicarea acestor metode sunt identificate numai vârfurile grafului cu proprietatea că există cel puțin un drum de la vârfurile inițiale către acel vârf. Grafurile cu proprietatea că oricare două vârfuri sunt conectate printr-un drum se numesc conex și sunt prezentate în § 3.3. Dacă graful este conex, atunci prin aplicarea metodelor de parcurgere vor fi identificate toate vârfurile grafului.

3.2.1 Metoda de parcurgere BF (Breadth First)

Ideea traversării BF este de parcurgere *în lățime* a grafului, în sensul că vârfurile grafului sunt prelucrate în ordinea crescătoare a “distanțelor” la vârfurile inițiale. Prin distanță se înțelege numărul de muchii din drumul identificat la acel moment de la vârfurile inițiale către acel vârf. La momentul inițial, vârfurile curente sunt v_0 . Deoarece vârfurile curente la fiecare moment trebuie să fie unul aflat la distanță minimă de v_0 , se poate proceda în modul următor: inițial lui v_0 i se asociază valoarea 0 și fiecărui vârf diferit de v_0 i se asociază valoarea -1 . Dacă valoarea asociată vârfurilor curente este m , atunci fiecăruia dintre vecinii acestuia de valoare -1 i se asociază valoarea $m+1$. Se observă că dacă după ce toate vârfurile de valoare m au fost considerate și nici unui vârf nu i-a fost recalculată valoarea, atunci toate vârfurile conectate cu v_0 au fost găsite, deci calculul se încheie.

Exemple:

3.8 Fie graful:



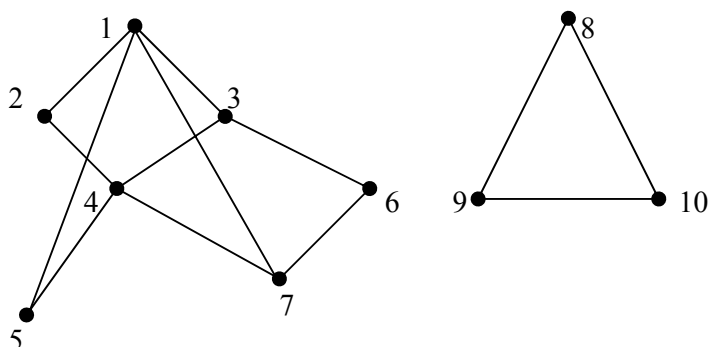
și $v_0=1$.

Valorile calculate prin aplicarea metodei prezentate sunt:

vârf \ m	1	2	3	4	5	6	7
0	0	-1	-1	-1	-1	-1	-1
1	0	1	1	-1	1	-1	1
2	0	1	1	2	1	2	1
	0	1	1	2	1	2	1

Ordinea de vizitare a vârfurilor: 1,2,3,5,7,4,6.

3.9. Fie graful:



și $v_0=1$.

Se observă că vârfurile 8, 9 și 10 nu sunt conectate cu vârful inițial.

Valorile rezultate prin aplicarea metodei sunt:

vârf \ m	1	2	3	4	5	6	7	8	9	10
0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	0	1	1	-1	1	-1	1	-1	-1	-1
2	0	1	1	2	1	2	1	-1	-1	-1
	0	1	1	2	1	2	1	-1	-1	-1

Ordinea de vizitare a vârfurilor este 1,2,3,5,7,4,6.

Se observă că valorile lui m calculate în final reprezintă numărul de muchii corespunzător celui mai scurt drum care conectează vârful inițial cu vârful respectiv, pentru vârfurile neconectate cu v_0 valoarea lui m rezultată la terminarea calculului este -1 .

O implementare diferită, dar urmând aceeași idee, rezultă prin utilizarea următoarelor structuri de date:

- A matricea de adiacență a grafului;
- o structură de tip coadă, C , în care sunt introduse vârfurile ce urmează a fi vizitate și procesate (în sensul cercetării vecinilor lor);

- Un vector c cu n componente, unde:

$$c_i = \begin{cases} 1, & \text{dacă } i \text{ a fost introdus în coadă} \\ 0, & \text{altfel} \end{cases}$$

și n este numărul vârfurilor grafului.

Componentele vectorului c vor fi inițializate cu valoarea 0.

Descrierea metodei este:

- se inițializează coada C cu vârful v_0 ;
- cât timp coada este nevidă, se extrage un vârful i din coadă, se vizitează, apoi se introduc în coadă numai vecinii acestuia care nu au fost deja introduși (adică toate vârfurile k având proprietatea că $c[k]=0$ și $a[i,k]=1$). Vârfurile i ce au fost introduse în coadă sunt marcate: $\forall i$ introdus în coadă, $c[i]=1$.

În continuare este prezentat un program Pascal pentru parcurgerea în lățime a unui graf. Vârfurile grafului sunt numerotate de la 1 până la n , parcurgerea începând cu vârful numerotat cu 1. Graful este reprezentat prin matricea de adiacență. Structura de coadă este implementată ca o listă simplu înlănțuită, procedurile *push* și *pop* realizând introducerea, respectiv extragerea unui element din coadă.

```
uses crt;
type
  ptcoada=^coada;
  coada=record
    inf:byte;
    leg:ptcoada;
  end;

var
  prim,ultim:ptcoada;
  c:array[1..20] of byte;
  a:array[1..20,1..20] of byte;
  i,j,k,n:byte;

procedure push(var p,u:ptcoada;i:byte);
var a:ptcoada;
begin
  new(a); a^.inf:=i; a^.leg:=nil;
  if p=nil then
    begin
      p:=a; u:=a;
    end
  else begin
    u^.leg:=a;u:=a;
  end;
end;

procedure pop(var p,u:ptcoada;var i:byte);
var a:ptcoada;
begin
  if p<>nil then
    begin
      i:=p^.inf; a:=p; p:=p^.leg;
      dispose(a);
      if p=nil then u:=nil;
    end;
```



```
end;

begin {program principal}
clrscr;
write('Numarul de varfuri:');
readln(n);
writeln('Matricea de adiacenta');
for i:=1 to n do
  for j:=1 to n do
    begin
      write('a[' ,i ,', ' ,j ,']=');
      readln(a[i,j]);
    end;
  for i:=2 to n do c[i]:=0;
  readln;
  clrscr;
  writeln('Incepem parcurgerea de la nodul 1');
  c[1]:=1;
  prim:=nil;ultim:=nil;
  push(prim,ultim,1);
  while prim<>nil do
    begin
      pop(prim,ultim,i);
      write(i,' ');
      for k:=1 to n do
        if (a[i,k]=1)and(c[k]=0) then
          begin
            c[k]:=1;
            push(prim,ultim,k);
          end;
      end;
    end;
  readln;
end.
```

3.2.2 Metoda de parcurgere DF (Depth First)

Ideea metodei DF revine la parcurgerea *în adâncime* a grafurilor, în sensul că, la fiecare moment, dacă M este mulțimea vârfurilor vizitate de procedură, pentru vizitarea vecinilor este considerat unul din vârfurile din M cu proprietatea că lungimea drumului calculat până la vârful inițial v_0 este maximă.

Implementarea metodei poate fi realizată în mai multe moduri, pentru menținerea mulțimii vârfurilor grafului disponibilizate până la momentul curent fiind utilizată o structură de date de tip stivă, S .

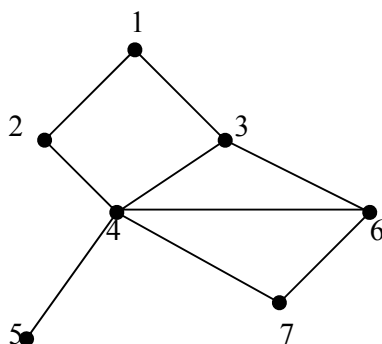
La momentul inițial se introduce în stivă v_0 . La fiecare pas, se preia cu ștergere ca vârf curent vârful stivei S și se introduc în stivă vecinii încă nevizitați ai vârfului curent. Un vârf se marchează ca vizitat în momentul introducerii lui în S . Calculul continuă până când este efectuat un acces de preluare din stivă și se constată că S este vidă. Pentru gestiunea vârfurilor vizitate, se utilizează un vector c cu n componente, unde n reprezintă numărul vârfurilor grafului și, la fiecare moment, componentele sunt:

$$c_i = \begin{cases} 1, & \text{dacă } i \text{ a fost vizitat} \\ 0, & \text{altfel} \end{cases}$$

Componentele vectorului c vor fi inițializate cu valoarea 0.

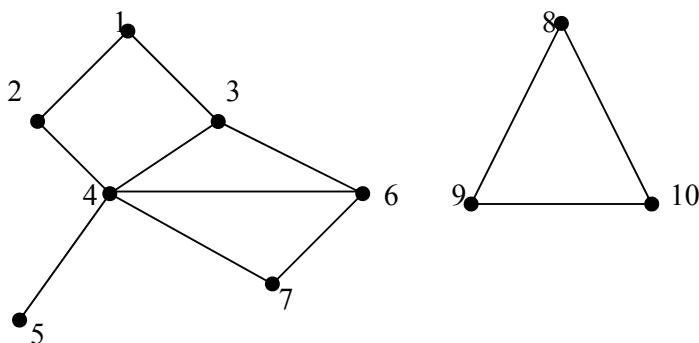
Example:

3.10. Pentru graful:



și $v_0 = 1$, ordinea în care sunt vizitate vârfurile este: 1, 2, 3, 4, 6, 7, 5.

3.11. Pentru graful



și $v_0 = 1$, ordinea în care sunt vizitate vârfurile este: 1, 2, 3, 4, 6, 7, 5. Vârfurile 8, 9 și 10 nu sunt vizitate de procedură, pentru că nu sunt conectate de vârful inițial selectat.

O variantă de implementare a metodei DF rezultă prin gestionarea stivei S după cum urmează. Inițial vârful v_0 este unicul component al lui S . La fiecare etapă se preia, fără ștergere, ca vârf curent vârful stivei. Se introduce în stivă unul dintre vecinii vârfului curent încă nevizitat. Vizitarea unui vârf revine la introducerea lui în S . Dacă vârful curent nu are vecini încă nevizitați, atunci este eliminat din stivă și

este efectuat un nou acces de preluare a noului vârf al stivei ca vârf curent. Calculul se încheie în momentul în care este efectuat un acces de preluare a vârfului stivei ca vârf curent și se constată că S este vidă. Evident, nici în cazul acestei variante nu vor fi vizitate vârfurile care nu sunt conectate cu vârful ales inițial.

3.3 Drumuri în grafuri. Conexitate

3.3.1 Drumuri; definiții

Una dintre cele mai importante proprietăți ale grafurilor o constituie posibilitatea de accesare, prin intermediul unei secvențe de muchii (arce), dintr-un vârf dat a oricărui alt vârf al grafului, proprietate cunoscută sub numele de *conexitate* sau *conexiune*. Așa după cum a rezultat în §3.2., dacă $G=(V,E)$ este un graf conex, atunci pentru orice vârf inițial v_0 considerat, metodele BF și DF permit vizitarea tuturor vârfurilor din V .

Definiția 3.3.1. Fie $G=(V,E)$ un graf, $u,v \in V$. Secvența de vârfuri $\Gamma: u_0, u_1, \dots, u_n$ este un u - v drum dacă $u_0=u$, $u_n=v$, $u_i u_{i+1} \in E$ pentru toți i , $0 \leq i \leq n$. Lungimea drumului, notată $l(\Gamma)$, este egală cu n . Convențional, se numește *trivial*, un drum Γ cu $l(\Gamma)=0$.

Definiția 3.3.2. Fie $\Gamma: u_0, u_1, \dots, u_n$ un drum în graful $G=(V,E)$. Γ este un *drum închis* dacă $u_0=u_n$; în caz contrar, Γ este *deschis*. Drumul Γ este *elementar* dacă oricare două vârfuri din Γ sunt distincte, cu excepția, eventual, a extremităților. Drumul Γ este *proces* dacă, pentru orice $0 \leq i \neq j \leq n-1$, $u_i u_{i+1} \neq u_j u_{j+1}$.

Evident, orice drum elementar este un proces.

Definiția 3.3.3. Fie $\Gamma: u_0, u_1, \dots, u_n$ un drum în graful $G=(V,E)$. Γ' : v_0, v_1, \dots, v_m este un subdrum al lui Γ dacă Γ' este un drum și pentru orice j , $0 \leq j \leq m$, există i , $0 \leq i \leq n$, astfel încât $u_i = v_j$.

Evident, orice drum cu lungime cel puțin 1 conține cel puțin un drum elementar cu aceleași extremități.

Într-adevăr, dacă $\Gamma: u_0, u_1, \dots, u_n$ nu este elementar, atunci există $0 \leq i < j \leq n$ și $i \neq 0$ sau $j \neq n$, astfel încât $u_i = u_j$.

Atunci drumul

$$\Gamma' : \begin{cases} u_j u_{j+1} \dots u_n, & \text{dacă } i = 0 \\ u_0 u_1 \dots u_i, & \text{dacă } j = 0 \\ u_0 u_1 \dots u_i u_{j+1} \dots u_n, & \text{dacă } i \neq 0, j \neq n \end{cases}$$

este, de asemenea, un u_0 - u_n drum. Aplicând în continuare eliminarea duplicatelor vârfurilor în modul descris, rezultă în final un u_0 - u_n drum elementar.

3.3.2 Matricea existenței drumurilor; algoritmul Roy-Warshall

Fie $G=(V,E)$ un graf, $|V|=n$. Dacă A este matricea de adiacență asociată grafului, atunci, pentru orice $p \geq 1$, $a_{ij}^{(p)}$ este numărul v_i - v_j drumurilor distincte de lungime p din graful G , unde $A^p = (a_{ij}^{(p)})$.

Definiția 3.3.4. Fie $M_n(\{0,1\})$ mulțimea matricelor de dimensiuni $n \times n$, componentele fiind elemente din mulțimea $\{0,1\}$. Pe $M_n(\{0,1\})$ se definesc operațiile binare, notate \oplus și \otimes , astfel: pentru orice $A=(a_{ij})$, $B=(b_{ij})$ din $M_n(\{0,1\})$, $A \oplus B=(c_{ij})$, $A \otimes B=(d_{ij})$, unde

$$1 \leq i, j \leq n,$$

$$c_{ij} = \max\{a_{ij}, b_{ij}\}$$

$$d_{ij} = \max\{\min\{a_{ik}, b_{kj}\}, 1 \leq k \leq n\}.$$

Dacă $A=(a_{ij}) \in M_n(\{0,1\})$, se notează $\{\bar{A}^k = (\bar{a}_{ij}^{(k)}), k \geq 1\}$ secvența de matrice definită prin:

$$\bar{A}^{(1)} = A, \bar{A}^k = A \otimes \bar{A}^{(k-1)}, \quad \forall k \geq 2.$$

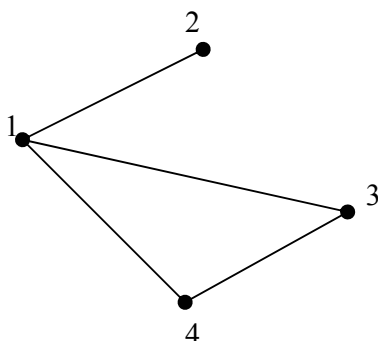
Dacă A este matricea de adiacență a unui graf $G=(V,E)$, atunci pentru fiecare k , $1 \leq k \leq n-1$, $\bar{a}_{ij}^{(k)} = \begin{cases} 1, & \text{dacă există drum de la } i \text{ la } j \text{ de lungime } k \\ 0, & \text{altfel} \end{cases}$

$M = \bar{A}^{(1)} \oplus \bar{A}^{(2)} \oplus \dots \oplus \bar{A}^{(n-1)}$ se numește *matricea existenței drumurilor* în graful G . Semnificația componentelor matricei M este:

$$\forall 1 \leq i, j \leq n, \quad m_{ij} = \begin{cases} 0, & \text{dacă nu există } v_i - v_j \text{ drum în } G \\ 1, & \text{altfel} \end{cases}$$

Exemplu:

3.12. Pentru graful:



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \bar{A}^2 = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \bar{A}^3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Calculul matricei existenței drumurilor permite verificarea faptului că un graf dat este conex: graful este conex dacă și numai dacă toate componentele matricei M sunt egale cu 1.

Algoritmul Roy-Warshall calculează matricea existenței drumurilor într-un graf G cu n vârfuri.

```

procedure Roy_Warshall (a,n,m);
i,j,k:integer;
do-for i=1,n,1
  do-for j=1,n,1
    mij=aij;
do-for j=1,n,1
  do-for i=1,n,1
    if mij=1 then
      do-for k=1,n,1
        if mik<mkj then
          mik=mkj;
        endif;
      enddo;
    endif;
  enddo;
enddo;
end;
  
```

Datele de intrare sunt: n , numărul de noduri și A , matricea de adiacență corespunzătoare grafului. Matricea M calculată de algoritm constituie ieșirea și este matricea existenței drumurilor în graful G .

3.3.3 Componente conexe ale unui graf

Definiția 3.3.5. Fie $G=(V,E)$ graf netrivial. Vârfurile $u,v \in V$ sunt *conectate* dacă există un u - v drum în G .

Definiția 3.3.6. Dacă G este un graf, atunci o componentă conexă a lui G este un subgraf conex al lui G , maximal în raport cu proprietatea de conexitate.

Evident, un graf este conex dacă și numai dacă numărul componentelor sale conexe este 1. Mulțimile vârfurilor corespunzătoare oricăror două componente conexe distincte sunt disjuncte. Mulțimile vârfurilor corespunzătoare componentelor conexe ale unui graf formează o partiție a mulțimii vârfurilor grafului.

Multe aplicații modelate în termeni de grafuri impun determinarea componentelor conexe corespunzătoare unui graf dat. Problema poate fi rezolvată în modul următor: se selectează un vârf al grafului, se determină componenta conexă care-l conține; dacă există vârfuri care nu aparțin componentei conexe determinate, se alege unul dintre acele vârfuri căruia i se determină componenta conexă care-l conține; în continuare, se repetă procedeul până când au fost găsite toate componentele conexe ale grafului.

Pentru $G=(V,E)$, $|V|=n$, $n \geq 1$ și $v_0 \in V$, pașii algoritmului pentru determinarea componentei conexe care conține un vârf v_0 dat sunt:

Pasul 1: $V_0=\{v_0\}$; $E_0=\Phi$; $i=0$;

Pasul 2: repetă *Pas 3* până când $V_i=V_{i-1}$ și $E_i=E_{i-1}$

Pasul 3: $i=i+1$;

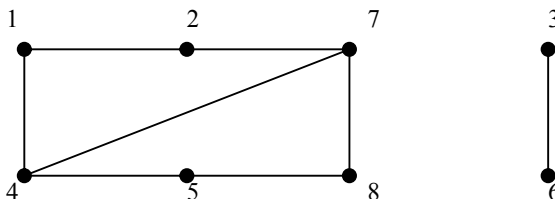
$$V_i = V_{i-1} \cup \{v / v \in V, \exists u \in V_{i-1}, uv \in E\};$$

$$E_i = E_{i-1} \cup \{e / e \in E, \exists u \in V_{i-1}, u \text{ incident cu } e\};$$

Ieșirea este $G_1=(V_i,E_i)$, componenta conexă din care face parte v_0 .

Exemplu:

3.13. Pentru graful



Aplicarea algoritmului descris, pentru $v_0=1$, determină următoarea evoluție:

i	V_i	E_i
i=0	{1}	\emptyset
i=1	{1,2,4}	{(1,2),(1,4)}
i=2	{1,2,4,7,5}	{(1,2),(1,4),(2,7),(4,5),(4,7)}
i=3	{1,2,4,7,5,8}	{(1,2),(1,4),(2,7),(4,5),(4,7),(5,8),(7,8)}
i=4	{1,2,4,7,5,8}	{(1,2),(1,4),(2,7),(4,5),(4,7),(5,8),(7,8)}

3.3.4 Drumuri de cost minim

Definiția 3.3.7. Fie $G=(V, E, w)$ un graf ponderat. Costul drumului $\Gamma: u_1, u_2, \dots, u_n$, notat $L(\Gamma)$, este definit prin:

$$L(\Gamma) = \sum_{i=1}^{n-1} w(u_i, u_{i+1}).$$

Pentru orice u și v vârfuri conectate în G , $u \neq v$, w -distanța între u și v , notată $D(u,v)$, este definită prin:

$D(u, v) = \min\{L(\Gamma), \Gamma \in D_{uv}\}$, unde D_{uv} desemnează mulțimea tuturor u - v drumurilor elementare din G . Dacă $\Gamma \in D_{uv}$ este astfel încât $D(u,v)=L(\Gamma)$, drumul Γ se numește *de cost minim*.

Cu toate că este utilizat termenul de w -distanță, în general D nu este o distanță în sensul matematic al cuvântului.

În particular, dacă funcția pondere asociază valoarea 1 fiecărei muchii a grafului, atunci, pentru fiecare pereche de vârfuri distincte ale grafului, costul $D(u,v)$ este lungimea celui mai scurt drum între cele două vârfuri. În acest caz, D este o distanță pe mulțimea vârfurilor.

Dat fiind interesul pentru determinarea w -distanțelor și a drumurilor de cost minim în cadrul aplicațiilor modelate pe grafuri, în continuare vor fi prezentați algoritmi pentru rezolvarea acestor probleme.

Algoritmul Dijkstra

Algoritmul a fost propus de E. W. Dijkstra pentru determinarea w -distanțelor $D(u_0,v)$ și a câte unui u_0 - v drum de cost minim pentru fiecare vârf $v \neq u_0$ într-un graf ponderat, unde u_0 este prestabilit.

Fie (V,E,w) graf conex ponderat, $u_0 \in V$, $S \subset V$, $u_0 \in S$. Se notează $\bar{S} = V \setminus S$ și $D(u_0, \bar{S}) = \min\{D(u_0, x), x \in \bar{S}\}$. Fie $v \in \bar{S}$ astfel încât $D(u_0,v)=D(u_0, \bar{S})$, $\Gamma: u_0, u_1, \dots, u_p, v$ este un u_0 - v drum de cost minim. Evident, $\forall 0 \leq i \leq p$ $u_i \in S$ și $\Gamma': u_0, u_1, \dots, u_p$, un u_0 - u_p drum de cost minim. De asemenea,

$$D(u_0, \bar{S}) = \min\{D(u_0, u) + w(uv); u \in S, v \in \bar{S}, uv \in E\}.$$

Dacă $x \in S$, $y \in \bar{S}$ astfel încât $D(u_0, \bar{S}) = D(u_0, x) + w(xy)$, rezultă $D(u_0, y) = D(u_0, x) + w(xy)$. Pentru determinarea unui drum u_0-v , cel mai ieftin, algoritmul consideră o etichetare dinamică a vârfurilor grafului. Eticheta vârfului v este $(L(v), u)$, unde $L(v)$ este lungimea celui mai ieftin drum u_0-v determinat până la momentul respectiv și u este predecesorul lui v pe un astfel de drum.

Pentru (V, E, w) graf conex ponderat, $|V| = n$ și $u_0 \in V$, calculul implicat de algoritmul Dijkstra poate fi descris astfel:

Pasul 1: $i=0$; $S_0=\{u_0\}$; $L(u_0)=0$, $L(v)=\infty$ pentru toți $v \in V$, $v \neq u_0$. Dacă $n=1$ atunci stop

Pasul 2: Pentru toți $v \in \bar{S}_i$, dacă $L(v) > L(u_i) + w(u_i, v)$, atunci $L(v) = L(u_i) + w(u_i, v)$ și etichetează v cu $(L(v), u_i)$.

Pasul 3: Se determină $d = \min\{L(v), v \in \bar{S}_i\}$ și se alege $u_{i+1} \in \bar{S}_i$ astfel încât $L(u_{i+1}) = d$.

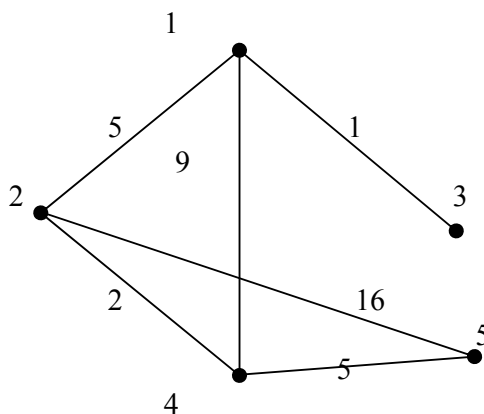
Pasul 4: $S_{i+1} = S_i \cup \{u_{i+1}\}$

Pasul 5: $i=i+1$. Dacă $i=n-1$, atunci stop. Altfel, reia Pasul 2.

Evident, dacă (V, E, w) este graf ponderat neconex, atunci, pentru $u_0 \in V$, algoritmul lui Dijkstra permite determinarea w -distanțelor $D(u_0, v)$ și a câte unui u_0-v drum de cost minim pentru toate vârfurile v din componenta conexă căreia îi aparține u_0 .

Exemplu:

3.14. Fie graful ponderat



Considerând $u_0=1$, etapele în aplicarea algoritmului Dijkstra sunt:

P1: $i=0$; $S_0=\{1\}$; $L(1)=0$, $L(i)=\infty$ pentru toți $i = \overline{2,5}$.

P2: $\bar{S}_0=\{2,3,4,5\}$, $u_0=1$

$L(2)=\infty > L(1)+5=5 \Rightarrow L(2)=5$, etichetează 2 cu 1

$L(3)=\infty > L(1)+1=1 \Rightarrow L(3)=1$, etichetează 3 cu 1

$L(4)=\infty > L(1)+9=9 \Rightarrow L(4)=9$, etichetează 4 cu 1

$L(5)=\infty$, $w(1,5)=\infty$, deci $L(5)$ nu se modifică

P3: selectează $u_1=3$, $L(3)=1$, cea mai mică dintre w-distanțele calculate la P2

P4: $S_1=\{1,3\}$

P5: $i=i+1=1 \neq 4$, reia P2

P2: $\bar{S}_1=\{2,4,5\}$, $u_1=3$

Nu se modifică nici o etichetă și nici o w-distanță ($w(3,i)=\infty$, pentru toți i din \bar{S}_1)

P3: selectează $u_2=2$, $L(2)=5$, cea mai mică dintre w-distanțele calculate la P2

P4: $S_2=\{1,3,2\}$

P5: $i=i+1=2 \neq 4$, reia P2

P2: $\bar{S}_2=\{4,5\}$, $u_2=2$

$L(4)=9 > L(2)+2=7 \Rightarrow L(4)=7$, etichetează 4 cu 2

$L(5)=\infty > L(2)+16=21$, etichetează 5 cu 2

P3: selectează $u_3=4$, $L(4)=7$, cea mai mică dintre w-distanțele calculate la P2

P4: $S_3=\{1,3,2,4\}$

P5: $i=i+1=3 \neq 4$, reia P2

P2: $\bar{S}_3=\{5\}$, $u_3=4$

$L(5)=21 > L(4)+5=12$, etichetează 5 cu 4

P3: selectează $u_4=5$, $L(5)=12$, cea mai mică dintre w-distanțele calculate la P2

P4: $S_3=\{1,3,2,4,5\}$

P5: $i=i+1=4$, stop.

Algoritmul calculează următoarele rezultate:

Vârful v până la care se calculează w-distanța	1	2	3	4	5
$D(1,v)$, eticheta lui v	0, 1	5, 1	1, 1	7, 2	12, 4

Drumurile de cost minim de la vârful 1 la fiecare dintre vârfurile grafului se stabilesc pe baza sistemului de etichete astfel: drumul de la 1 la un vârf v este dat de: v_1 , eticheta lui v , v_2 eticheta lui v_1 ș.a.m.d., până se ajunge la eticheta 1.

Astfel, v_0 -drumurile de cost minim sunt:

până la 2: 2,1;

până la 3: 3,1;

până la 4: 4,2,1;

până la 5: 5,4,2,1.

Programul Pascal pentru algoritmul Dijkstra, graful ponderat fiind reprezentat sub formă tabelară, este prezentat în continuare.

Variabilele folosite în program au următoarele semnificații:

- v: numărul de vârfuri ale grafului;
- m: numărul de muchii din graf;
- s: mulțimea S_i la momentul curent;
- t: mulțimea \bar{S}_i la momentul curent;
- l: vectorul componentelor L din etichetele atașate vârfurilor;
- dr: vectorul componentelor de tip vârf din etichetele atașate vârfurilor.

```
uses crt;
var a:array[1..200,1..3] of integer;
    s,dr,t,l:array[1..20] of integer;
    i,j,k,p,v1,u,c,d:integer;
    m,v:integer;
begin
  clrscr;
  readln(v);
  readln(m);
  for i:=1 to m do
    for j:=1 to 3 do
      begin
        write('a[' ,i ,',',j ,']=');
        readln(a[i,j]);
      end;
  i:=1;s[1]:=1;v1:=1;
  for j:=2 to v do
    begin
      t[j-1]:=j; l[j]:=maxint;
    end;
  l[1]:=0;
  for k:=1 to v-1 do
    begin
      for j:=1 to v-i do
        begin
          u:=1;
          while((t[j]<>a[u,1])or(v1<>a[u,2]))and
            ((t[j]<>a[u,2])or(v1<>a[u,1]))and(u<=m)
          do inc(u);
          if(u<=m) and (l[t[j]]>l[v1]+a[u,3]) then
            begin
              l[t[j]]:=l[v1]+a[u,3];
              dr[t[j]]:=v1;
            end;
          end;
        d:=l[t[1]];v1:=t[1];
        for j:=2 to v-i do
          if l[t[j]]<d then begin
            d:=l[t[j]];
```

```
        v1:=t[j];
    end;
    inc(i);
    s[i]:=v1;
    u:=1;
    while(u<=v-i+1)and(t[u]<>v1) do inc(u);
    for j:=u to v-i do t[j]:=t[j+1];
    end;
clrscr;
for i:=1 to v do
    writeln(i,'--->',l[i]);
writeln('Drumurile minime de la fiecare varf la 1:');
for j:=2 to v do
    begin
        k:=j;write(k,' ');
        while k<>1 do
            begin
                write(dr[k],' '); k:=dr[k];
            end;
        writeln;
    end;
readln;
end.
```

În anumite cazuri se dorește determinarea numai a w -distanțelor $D(v_0, v)$, pentru toți $v \in V$. În acest caz, algoritmul Roy-Floyd permite o rezolvare a acestei probleme mai simplu de implementat decât algoritmul Dijkstra.

Algoritmul Roy-Floyd

Pentru (V, E, w) graf ponderat, $|V| = n$ și W matricea ponderilor, sistemul de w -distanțe $D(v_0, v)$, $v \in V$, poate fi calculat pe baza următoarei proceduri (similară algoritmului Roy-Warshall):

```
procedure Roy_Floyd (w,n,d);
i,j,k:integer;
do-for i=1,n,1
    do-for j=1,n,1
        dij=wij;
    do-for j=1,n,1
        do-for i=1,n,1
            if dij ≠ ∞ then
                do-for k=1,n,1
                    if dik>dij+ djk then
                        dik=dij+ djk;
                    endif;
                enddo;
            endif;
        enddo;
    enddo;
enddo;
```

end;

Matricea D calculată de algoritm este a w -distanțelor $D(u,v)$ în graful ponderat conex (V,E,w) ; pentru orice $1 \leq i, j \leq n$

$$d_{ij} = \begin{cases} D(v_i, v_j), & v_i, v_j \text{ sunt conectate} \\ \infty, & \text{altfel} \end{cases}$$

Într-adevăr, procedura calculează dinamic w -distanța între oricare două vârfuri i și k , astfel: dacă există un i - k drum ce trece prin j ($1 \leq j \leq n$), cu costul corespunzător ($d_{ij} + d_{jk}$) inferior costului curent (d_{ik}), atunci noul drum de la i la k via j este de cost mai mic decât cel al drumului vechi, deci w -distanța între i și k trebuie reactualizată la $d_{ij} + d_{jk}$.

Algoritmul Yen

Pentru calculul tuturor w -distanțelor într-un graf ponderat, algoritmul propus de Yen rezolvă problema într-o manieră mai eficientă decât algoritmul Roy-Floyd, din punctul de vedere al volumului operațiilor. Fie (V,E,w) un graf ponderat, W matricea ponderilor. Pentru determinarea w -distanțelor de la vârful v_k fixat la celelalte vârfuri ale grafului, algoritmul Yen inițiază următoarele operații:

Pasul 1: $D = W$

Pasul 2: $i = 1$; $\lambda(k) = 0$, $b(k) = 0$; $\lambda(j) = 0$, pentru toți $1 \leq j \leq n$, $j \neq k$

Pasul 3: Calculează $\min\{d_{kj}; 1 \leq j \leq n, \lambda(j) = 1\}$; determină j_0 astfel încât

$$\lambda(j_0) = 1 \text{ și } d_{kj_0} = \min\{d_{kj}; 1 \leq j \leq n, \lambda(j) = 1\}$$

$$B(j_0) = d_{kj_0}, \lambda(j_0) = 0$$

$$d[k,j] = \min\{d[k,j], d[k,j_0] + d[j_0,j]\}, \text{ pentru toți } j, 1 \leq j \leq n$$

$$i = i + 1$$

Pasul 4: Dacă $i < n$, reia Pasul 3, altfel stop.

La terminarea algoritmului, componentele vectorului B sunt egale cu w -distanța de la vârful v_k la orice alt vârf al grafului: $B(j) = D(v_k, v_j)$, $1 \leq j \leq n$. Într-adevăr, componentele egale cu 1 ale vectorului λ indică, la fiecare reluare a pasului 3, vârfurile grafului pentru care nu s-a calculat încă w -distanța la vârful v_k . După fiecare efectuare a etapei 3, dacă j_0 a fost selectat, atunci $B(j_0) = D(v_k, v_{j_0})$.