GRAFURI NEORIENTATE ȘI ORIENTATE

CURSURI 10, 11- 11.05.2021

Titular: Șef. Lucr. Dr. Mat. Cărbureanu Mădălina

Universitatea Petrol-Gaze din Ploiești

OBIECTIVE:

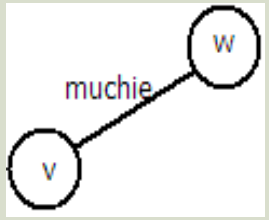
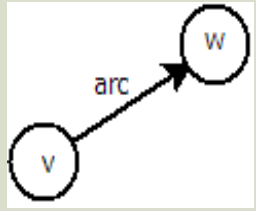
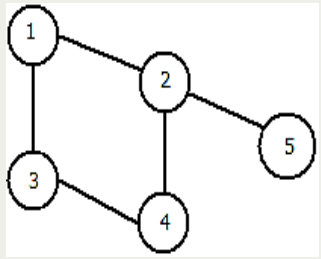
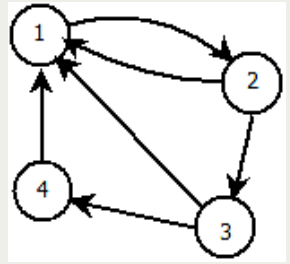
- **NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE;**
- **MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE;**
- **PARCURGERI ALE GRAFURILOR;**
- **ALGORITMUL ROY-WARSHALL;**
- **ALGORITMI PENTRU DETERMINAREA CELOR MAI SCURTE DRUMURI (alg. Roy-Floyd și alg. Dijkstra);**
- **ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM (alg. Kruskal și alg. Prim).**

NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE

- **Graf** ⟹ pereche $G=(V, E)$, unde V este o mulțime finită și nevidă, ce reprezintă mulțimea de vârfuri (noduri), iar $E=\{(v, w) \mid v, w \in V\}$ reprezintă mulțimea de muchii (perechi de vârfuri, arce);
- **Grafuri** ⟹ **grafuri neorientate** (grafuri în care nu contează sensul de parcurgere al unei muchii (v, w), unde $v, w \in V$) ;

  ⟹ **grafuri orientate** (grafuri în care contează sensul de parcurgere a unei muchii (v, w), unde $v, w \in V$).

NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE

Graf neorientat	Graf orientat
	
	

NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE

Tabel 1. Paralelă noțiuni grafuri neorientate și orientate

Graf neorientat	Graf orientat
Muchie	Arc
Lanț	Drum/drum simplu
Ciclu/ciclu simplu	Circuit
Graf conex	Graf tare conex
Grad nod	Grad interior și grad exterior nod

NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE

- **Muchie graf neorientat** - pereche de noduri (v, w), unde $v, w \in V$, în care orientarea muchiei nu este luată în considerare.
- **Lanț** într-un **graf neorientat** - succesiune de muchii de forma $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_{k+1})$.
- **Ciclu** într-un **graf neorientat** - lanț cu proprietatea $v_{k+1}=v_1$.
- **Ciclu simplu graf neorientat** - drum de lungime cel puțin 1, care începe și se sfârșește în același vârf (nod).
- Un **graf neorientat** $G=(V, E)$ este **graf conex**, dacă între oricare două vârfuri (noduri) ale sale există un lanț.

NOȚIUNI SPECIFICE GRAFURILOR NEORIENTATE ȘI ORIENTATE

- **Gradul unui nod** (*grad(nod)*) în cazul unui **graf neorientat**, reprezintă numărul de muchii incidente la nod.
- Un **arc** al unui **graf orientat** este o pereche ordonată de elemente distincte din V.
- Se numeşte **drum** într-un **graf orientat** o secvenţă de forma $(v_1,v_2),(v_2,v_3),\ldots,(v_k,v_{k+1})$.
- Un **drum simplu** este un drum în care niciun vârf nu se repetă, respectiv un drum în care toate vârfurile sale sunt distincte.
- **Circuit** într-un **graf orientat** este un drum cu proprietarea $v_{k+1}=v_1$.
- Un **graf orientat** $G=(V, E)$ este **tare conex,** daca pentru oricare două noduri x şi y din V, există un drum de la x la y şi un drum de la y la x .
- Prin **grad interior** al unui **nod** (*grad_int(nod)*) dintr-un **graf orientat** se înţelege numărul de arce care intră în respectivul nod.
- Prin **grad exterior** al unui **nod** (*grad_ext(nod)*) dintr-un **graf orientat** se înţelege numărul de arce care ies din respectivul nod.
- Un **graf neorientat**, respectiv **orientat** este **complet** dacă oricare două vârfuri (noduri) ale sale sunt adiacente.

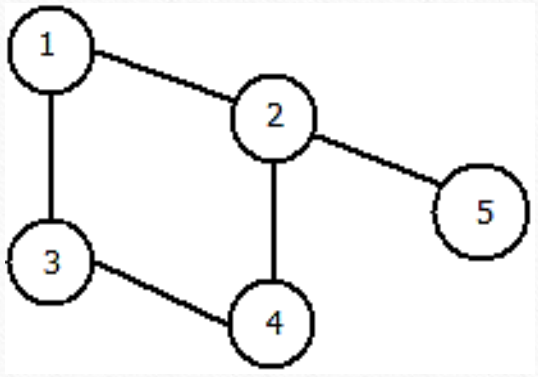
MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Printre cele mai utilizate modalități de **reprezentarea ale grafurilor neorientate în memoria calculatorului**i sunt: matricile de adiacență, listele de adiacență (liste de vecini), matricile de incidență și matricea costurilor.
- În cazul reprezentării **grafurilor neorientate** prin **matricea de adiacență** $\boldsymbol{A(a_{ij})}$ (simetrică față de diagonala principală $a_{ij}=a_{ji}$, pentru $(\forall) i, j=1, \ldots, n$), elementele $\boldsymbol{a_{ij}}$ sunt date de relația: $a_{ij}=\begin{cases}1, \text{ dacă } (i, j)\in E;\\ 0, \text{ dacă } (i,j)\notin E;\end{cases}$

MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Reprezentarea **grafurilor neorientate** prin utilizarea **listelor de adiacență** (**listelor de vecini**) constă în memorarea pentru fiecare nod din graf, a listei cu nodurile adiacente (vecine).

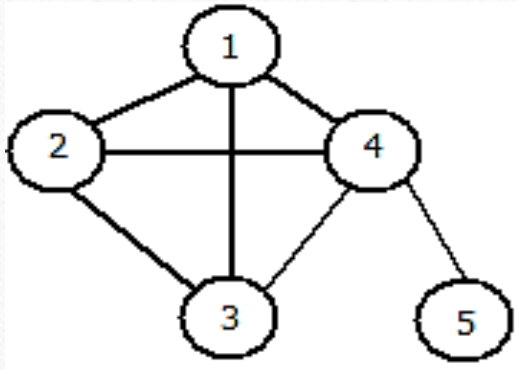
Nod	Listă vecini
1	2, 3
2	1, 4, 5
3	1, 4
4	2, 3
5	2



MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Pentru reprezentarea **grafurilor neorientate** prin utilizarea **matricilor de incidență** $A(a_{ij})$, elementele a_{ij} sunt date de relația: $a_{ij}=\begin{cases}1, & \text{dacă nodul } i \text{ este incident la muchia } (i, j);\\ 0, & \text{dacă nodul } i \text{ nu este incident la muchia } (i, j);\end{cases}$
- Obs: un nod se numește incident la o muchie dacă este capăt al muchiei respective.

Muchie / Nod	(1, 2)	(1, 3)	(2, 4)	(2, 5)	(3, 4)
1	1	1	0	0	0
2	1	0	1	1	0
3	0	1	0	0	1
4	0	0	1	0	1
5	0	0	0	1	0

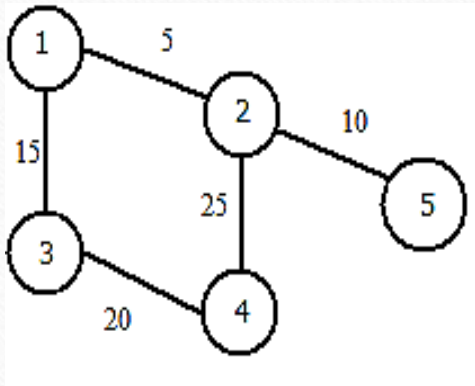


MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Pentru reprezentarea unui **graf neorientat** prin **matricea costurilor** $C(c_{ij})$, elementele c_{ij} sunt date de relaţia:

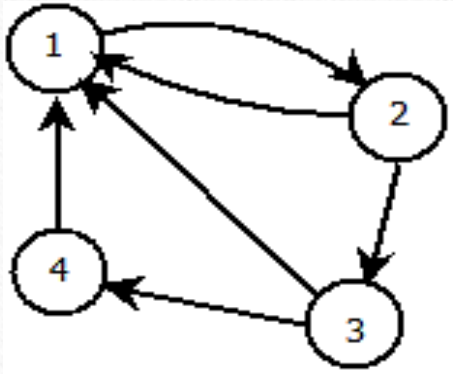
$$c_{ij} = \begin{cases} \text{cost}(i,j), & \text{dacă } (i,j) \in E; \\ 0, & \text{dacă } i=j; \\ \pm\infty, & \text{dacă } (i,j) \notin E; \end{cases}$$

Nod	1	2	3	4	5
1	0	5	15	+∞	+∞
2	5	0	+∞	25	10
3	15	+∞	0	20	+∞
4	+∞	25	20	0	+∞
5	+∞	10	+∞	+∞	0



MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

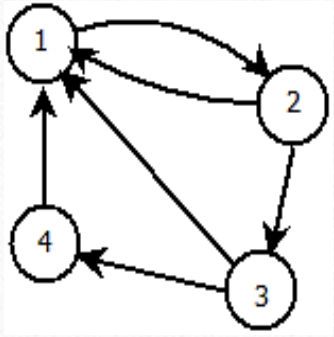
- În ceea ce privește **reprezentarea în memoria calculatorului a grafurilor orientate**, cele mai utilizate modalități de reprezentare a acestora sunt: matricea de adiacență, listele de adiacență, matricea vârfuri-arce, listele de vecini, matricea costurilor și matricea drumurilor.



- **matricea de adiacență *M(m_{ij})*** ***(***nu este simetrică față de diagonala principală***)***, în care elementele m_{ij} sunt date de relația: $m_{ij}=\begin{cases}1, \text{ dacă } (i,j)\in E;\\0, \text{ dacă } (i,j)\notin E;\end{cases}$

MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Folosind **listele de adiacență** pentru un **graf orientat**, pentru fiecare nod al grafului se memorează arcele care pleacă din nodul respectiv.



Nod	Listă adiacență
1	(1, 2)
2	(2, 1), (2, 3)
3	(3, 1), (3, 4)
4	(4, 1)

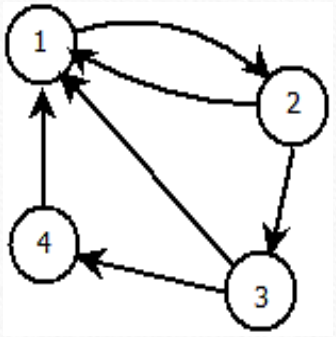
MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Matricea **vârfuri-arce** este o matrice $B(n \times m)$, în care fiecare element bij este dat de:

- $b_{ij} = \begin{cases} 1, & \text{dacă nodul } i \text{ este extremitate inițială a arcului } (i,j); \\ -1, & \text{dacă nodul } i \text{ este extremitate finală a arcului } (i,j); \\ 0, & \text{dacă nodul } i \text{ nu este extremitate a arcului } (i,j); \end{cases}$

- $B(4X6)$, ($|V|=4$ şi $|E|=6$)

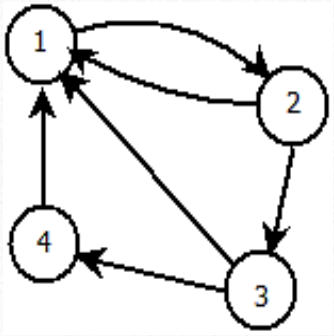
Nod	B(b_{ij})					
1	b_{11} (1)	b_{12} (1)	b_{13} (-1)	b_{14} (-1)	b_{15} (0)	b_{16} (0)
2	b_{21} (-1)	b_{22} (1)	b_{23} (1)	b_{24} (0)	b_{25} (0)	b_{26} (0)
3	b_{31} (1)	b_{32} (-1)	b_{33} (1)	b_{34} (1)	b_{35} (0)	b_{36} (0)
4	b_{41} (1)	b_{42} (0)	b_{43} (-1)	b_{44} (1)	b_{45} (0)	b_{46} (0)



MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

În ceea ce privește **listele de vecini** pentru reprezentarea **grafurilor orientate**, pentru fiecare nod al grafului se construiesc două liste ale vecinilor săi astfel:

- $L^+(x)$ ce reprezintă **lista vecinilor succesori**, adică conține nodurile care sunt extremități finale ale arcelor care ies din nodul x;
- $L^-(x)$ ce reprezintă **lista vecinilor predecesori**, adică conține nodurile care sunt extremități inițiale ale arcelor care intră în nodul x.

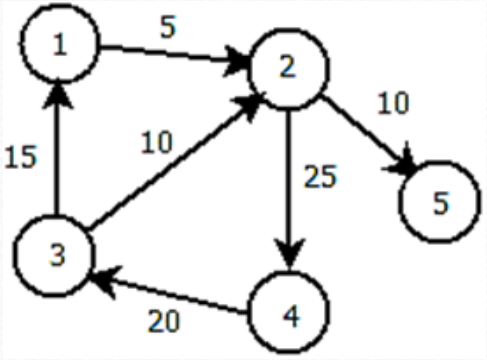


Nod	$L^+(x)$	$L^-(x)$
1	2	2, 3, 4
2	1, 3	1
3	1, 4	2
4	1	3

MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

În reprezentarea unui **graf orientat** prin **matricea costurilor** $C(c_{ij})$, fiecare element c_{ij} al matricei este dat de relația: $c_{ij} = \begin{cases} \text{cost}(i,j), & \text{dacă } (i,j) \in E; \\ 0, & \text{dacă } i=j; \\ \pm\infty, & \text{dacă } (i,j) \notin E; \end{cases}$

Nod	1	2	3	4	5
1	0	5	+∞	+∞	+∞
2	+∞	0	+∞	25	10
3	15	10	0	+∞	+∞
4	+∞	+∞	20	0	+∞



MODALITĂȚI DE REPREZENTARE A GRAFURILOR NEORIENTATE ȘI ORIENTATE ÎN MEMORIE

- Pentru reprezentarea **grafurilor orientate** folosind **matricea drumurilor** $\mathbf{M}^*\left(m_{ij}\right)$, fiecare element m_{ij} este dat de relația:

$$m_{ij}=\begin{cases}1, & \text{dacă există drum de la nodul } i \text{ la nodul } j;\\ 0, & \text{dacă nu există drum de la nodul } i \text{ la nodul } j;\end{cases}$$

PARCURGERI ALE GRAFURILOR

- Parcurgerea grafurilor se realizează folosind două strategii de căutare, și anume:
- **Pacurgerea în lățime (Breadth First Search – BFS);**
- **Parcurgerea în adâncime (Depth First Search – DFS).**
- În cazul **parcurgerii în lățime (BFS)**, parcurgerea se realizează începând cu vârful (nodul) de plecare, apoi se parcurg vecinii vârfului de plecare, apoi se parcurg vecinii nevizitați ai acestora, ș.a.m.d., până când au fost vizitate toate vârfurile grafului;
- Deci, pentru a stabili ordinea de vizitare a vârfurilor se folosește o coadă, iar pentru a stabili dacă un vârf a fost sau nu vizitat se foloseşte un vector viz[j], $j=1,\ldots,n$, definit conform relației: $viz[j]=\begin{cases} 1, \text{ dacă nodul } j \text{ a fost vizitat, pt.} (\forall) j=1,\ldots, n; \\ 0, \text{ în caz contrar;} \end{cases}$

PARCURGERI ALE GRAFURILOR

Pseudocod BFS-grafuri

Intrare: un graf G=(V, E) şi un nod vârf iniţial i∈V;
Postcondiţie: un nod este accesibil din i dacă şi numai dacă este marcat ca fiind vizitat;

(1) fie o coadă C iniţializată (prim=ultim=1);
(2) adaugă vârful iniţial i în coada C (c[1]=i) şi marchează vârful i ca fiind vizitat (viz[i]=1);
(3) atâta timp cât coada C nu este vidă (while(prim<=ultim)) execută
(4) extrage un element (vârf) v din coada C (v=c[prim]) şi determină toţi vecinii j (j=1, …, n) nevizitaţi ai acestuia;
(5) dacă vârful j este vecin cu vârful v (if (a[v][j]==1)) şi vârful j este nevizitat (if (viz[j]==0)) atunci
(6) se vizitează vârful j (viz[j]=1) şi se adaugă vecinul j în coadă (c[ultim]=j);
(7) elimină elementul vizitat din coadă;

- Codul sursă (complet) al parcurgerii BFS a unui graf (graf neorientat) este prezentat în fig. 8.8., pag. 175;
- Pentru construcţia practică a algoritmului BFS s-a utilizat viz[20]şi vectorul c[20] pentru gestiunea unei cozi implementată static, în care prelucrarea unui nod *v* aflat la un capăt al cozii constă în introducerea în celălalt capăt al ei a tuturor nodurilor vecine cu nodul *v*, nevizitate încă.

PARCURGERI ALE GRAFURILOR

- În cazul **parcurgerii în adâncime (Depth First Search-DFS) a unui graf**, se vizitează mai întâi vârful inițial (vârful de start) și apoi se vizitează primul vecin (primul nod adiacent) nevizitat al vârfului de start, nod ce devine nod curent. Apoi se vizitează primul vecin nevizitat al nodului curent (primul vecin al vârfului inițial) și așa mai departe, mergând în adâncime până când se ajunge într-un vârf care nu mai are vecini nevizitați. Când se ajunge la un astfel de vârf, se revine la vârful tată al acestuia (vârful din care acesta a fost vizitat) și dacă acest vârf mai are vecini nevizitați, se alege primul vecin nevizitat al său și se continuă parcurgerea în aceeași manieră. Dacă nici vârful tată nu mai are vecini nevizitați, se revine la vârful său tată și se continuă în același mod, până când nu a mai rămas niciun nod adiacent nevizitat din vârful de start.

PARCURGERI ALE GRAFURILOR

- **Parcurgerea în adâncime (DFS)** pentru grafuri prezintă două variante de implementare, și anume o **variantă iterativă** și o **variantă recursivă** elegantă de implementare;

Varianta iterativă

```
Intrare: un graf G=(V, E) reprezentat folosind matricea de adiacenţă şi un
        nod (vârf) s∈V;
Postcondiţie: un nod este accesibil din s dacă şi numai dacă este marcat ca
        fiind vizitat;
(1) marchează toate nodurile ca fiind nevizitate;
(2) fie o stivă S iniţializată cu nodul s;
(3) cât timp stiva S nu este vidă execută
(4)    scoate nodul v din vârful stivei S;
(5)      dacă nodul nodul v este nevizitat atunci
(6)            marchează nodul v ca fiind vizitat;
(7)            pentru fiecare muchie/arc (v, w) în lista de adiacentă a lui v
execută
(8)             adaugă nodul w prin vârful stivei S;
```

Varianta recursivă

```
Intrare: un graf G=(V, E) reprezentat folosind matricea de
        adiacenţă şi un nod (vârf) s∈V;
Postcondiţie: un nod este accesibil din s dacă şi numai dacă
        este marcat ca fiind vizitat;
(1) marchează nodul s ca fiind vizitat;
(2) pentru fiecare muchie/arc (s, v) din lista de adiacenţă a lui
s execută
(3)     dacă nodul v este nevizitat atunci
(4)       apel DFS(G, v);
```

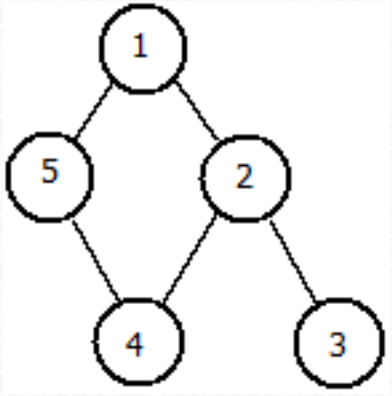
PARCURGERI ALE GRAFURILOR

- Datorită eleganței și a ușurinței în implementare a parcurgerii DFS varianta recursivă de implementare a DFS a fost aleasă pentru parcurgerea în adâncime a unui graf (graf neorientat), codul sursă (complet) fiind prezentat în fig. 8.11, pag. 179.
- Graful neorientat din fig. 8.11 este reprezentat prin matricea de adiacență, iar pentru a reține care dintre noduri au fost vizitate s-a utilizat vectorul viz[20].

PARCURGERI ALE GRAFURILOR

Parcurgere BFS

Nr. de noduri graf: 5
Nr. de muchii/arce graf: 5
Muchia 1=1 5
Muchia 2=1 2
Muchia 3=5 4
Muchia 4 =2 4
Muchia 5=2 3
Matricea de adiacență este:
0 1 0 0 1
1 0 1 1 0
0 1 0 0 0
0 1 0 0 1
1 0 0 1 0



Lista varfurilor(nodurilor) parcurse BFS plecand din vârful 1 este:
1, 2, 5, 3, 4

Pași BFS: pag. 176

Parcurgere DFS

Nr. de noduri graf: 5
Nr. de muchii/arce graf: 5
Muchia 1=1 5
Muchia 2=1 2
Muchia 3=5 4
Muchia 4 =2 4
Muchia 5=2 3
Matricea de adiacență este:
0 1 0 0 1
1 0 1 1 0
0 1 0 0 0
0 1 0 0 1
1 0 0 1 0
Nodul de pornire: 1
Lista varfurilor(nodurilor) parcurse în adâncime DFS este:
1, 2, 3, 4, 5

Pași DFS: pag. 180.

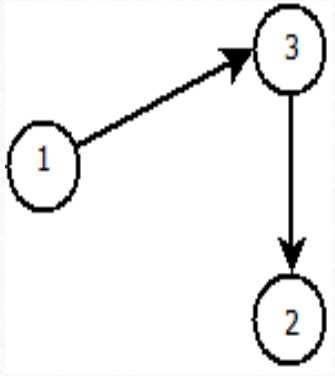
ALGORITMUL ROY-WARSHALL

- **Algoritmul Roy-Warshall** (dezvoltat de către **Bernard Roy** și **Stephen Warshall**) prezentat în cele ce urmează, este utilizat pentru determinarea matricei drumurilor asociată unui graf orientat, plecând de la matricea de adiacență;
- Acest algoritm scoate în evidență aplicabilitatea grafurilor în probleme de accesibilitate (probleme pentru determinarea existenței sau nu a drumurilor) și se poate aplica de asemenea și pentru grafuri neorientate, în cazul în care o muchie este considerată a fi un arc în direcția opusă.
- Se dă o matrice de adiacență $M\left(m_{ij}\right)$ asociată grafului orientat $G=(V, E)$ și se cere a se determina matricea drumurilor (așa-numita matrice de accesibilitate) $M^{*}\left(m_{ij}\right)$ pentru graful G.

ALGORITMUL ROY-WARSHALL

- **Algoritmul Roy-Warshall** pentru determinarea matricei drumurilor $M^*\left(m_{ij}\right)$, pornind de la matricea de adiacentă $M\left(m_{ij}\right)$ asociată unui graf orientat $G=(V, E)$, constă în următorii pași:
  - Se consideră contorul $k=1, k=1, ..., n;$
  - Pentru fiecare $i, j=1, ..., n$ se înlocuiesc toate elementele m_{ij} din matricea de adiacență $M\left(m_{ij}\right)$ care sunt egale cu 0 cu minimul dintre $m_{i,k}$ și $m_{k,j}$ pentru $k \neq i$ și $k \neq$j;
  - Pentru $k=2, ..., n$ se reia algoritmul.
- Un exemplu de aplicare (+pașii necesari) a algoritmului Roy-Warshall este prezentat pentru graful orientat din fig. 8.15, pag. 183;
- Implementarea în limbajul C a algoritmului este prezentată în cadrul aplicației 8.2.1, pag. 197.

ALGORITMUL ROY-WARSHALL



Matricea de adiacență $M(m_{ij})$	Matricea drumurilor $M^*(m_{ij})$
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

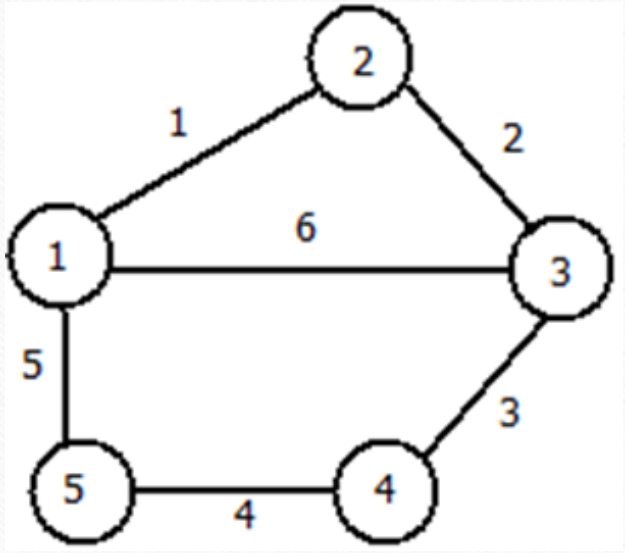
ALGORITMI PENTRU DETERMINAREA CELOR MAI SCURTE DRUMURI

- **Algoritmul Roy-Floyd** (dezvoltat de către **Robert Floyd** și **Bernard Roy**);
- **Algoritmul Dijkstra** (dezvoltat de către **Edsger Dijkstra**);
- **Algoritmul Roy-Floyd** este o generalizare a **algoritmului Roy-Warshall** prezentat în cadrul secțiunii 8.1.4. (*Algoritmul Roy-Warshall pentru determinarea matricei drumurilor*), pag. 181, algoritm care determină cele mai scurte drumuri într-un graf și nu doar existența sau nu a drumurilor, așa cum se întâmplă în cazul algoritmului Roy-Warshall;
- Se utilizează așa-numita matrice a costurilor (a ponderilor) $C(c_{ij})$ asociată grafului orientat;

ALGORITMI PENTRU DETERMINAREA CELOR MAI SCURTE DRUMURI

- **Paşi algoritm Roy-Floyd:**
- Se construieşte matricea costurilor $C(c_{ij})$ pentru respectivul graf neorientat sau orientat pentru care se doreşte a se aplica algoritmul;
- Pentru *k, i, j=1, ..., n* (număr de noduri/vârfuri) se testează dacă *i<>j, j<>k şi k<>i;* dacă condiţia este îndeplinită atunci se testează dacă costul drumului de la nodul *i* la nodul *k* însumat cu costul drumului de la *k* la *j* este mai mic decât costul drumului de la *i* la *j* (*if(c[i][j]>c[i][k]+c[k][j])*), atunci drumul iniţial de la *i* la *j* se înlocuieşte cu drumul indirect *i→k→j*, respectiv cu *c[i][j]=c[i][k]+c[k][j]*;
- Se afişează matricea drumurilor de cost minim obţinută, respectiv C(cij).
- Implementarea în limbajul C a algoritmului Roy-Floyd este prezentată în secţiunea 8.2.2 (*Implementare Roy-Floyd*), pag. 199.

ALGORITM ROY-FLOYD



Conţinutul fişierului GM.TXT:
1 2 1
2 3 2
1 3 6
3 4 3
4 5 4
5 1 5
Matricea costurilor este:
0 1 6 100 5
1 0 2 100 100
6 2 0 3 100
100 100 3 0 4
5 100 100 4 0
Matricea drumurilor de cost minim este:
0 1 3 6 5
1 0 2 5 6
3 2 0 3 7
6 5 3 0 4
5 6 7 4 0

ALGORITMI PENTRU DETERMINAREA CELOR MAI SCURTE DRUMURI

- Un alt algoritm pentru determinarea drumurilor de cost minim în grafurile orientate sau neorientate este **algoritmul Dijkstra**, descoperit de către Edsger W. Dijkstra in 1956. Acesta determină lungimile celor mai scurte drumuri de la un nod (vârf) de start către toate celelalte noduri.
- **algoritmul lui Dijkstra** calculează lungimile drumurilor minime de la un nod (vârf) inițial la toate celelalte noduri ale unui graf dat prin matricea ponderilor (costurilor) și utilizează următoarele elemente:
  - Vectorul *L[30]* ce reprezintăm vectorul lungimilor minime; *L(i)* reprezintă lungimea minimă a unui drum de la vârful *v* la *i*; *tata[30]* ce reține poziția unui vârf anterior tata(*i*) pe un drum minim de la *v* la *i*; *calculat[30]* ce memorează mulțimea de vârfuri pentru care s-a calculat deja drumul minim;
  - *vref* reprezintă vârful de referință, iar *ref_nouă* reprezintă noul vârf de referință;
  - *w[v][w]* reprezintă ponderea muchiei *(v, w)*.

ALGORITMI PENTRU DETERMINAREA CELOR MAI SCURTE DRUMURI

```
(1) pentru k← 1, ... , n execută
(2)     dacă k==v atunci L[k] ←0; calculat[k] ← 1; tata[k] ← k;
(3)         altfel calculat[k] ← 0; L[k] ← ∞;
(4)  vref ← v; nr_calcul←1;
(5)  cât timp dmin<>∞ sau nr_calcul<>n execută
(6)    dmin← ∞;
(7)         pentru k←1,…, n execută
(8)                 dacă calculat[k]==0 atunci
(9)                         dacă L[vref]+w[vref][k]<L[k] atunci
(10)                                tata[k] ←vref; L[k] ← L[vref]+w[vref][k];
(11)                        dacă dmin>L[k] atunci ref_nouă ←k; dmin ←L[k];
(12)                end;
(13)                 dacă dmin< ∞ atunci
(14)                        vref←ref_nouă; calculate[vref] ←1; nr_calcul←nr_calcul+1;
(15)                 end; end;
```

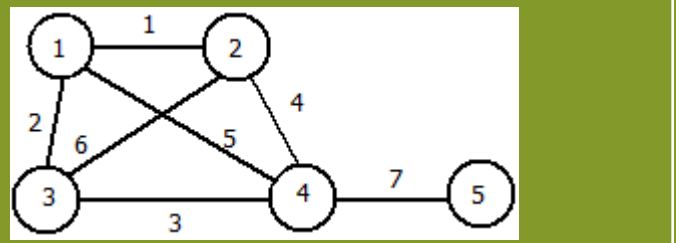
Pseudocod Dijkstra

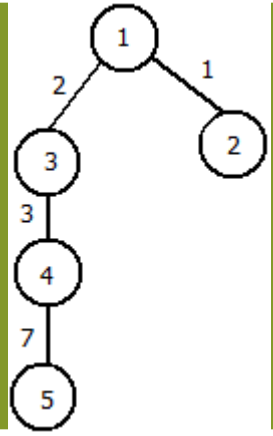
ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

- **algoritmul lui Kruskal** (dezvoltat de către **Joseph Kruskal** în anul 1956);
- **algoritmul lui Prim** (dezvoltat de către **Robert Prim** în anul 1957).
- Astfel, fie *G=(V, E)* un graf neorientat conex (un graf neorientat G=*(V, E)* este **graf conex** dacă între oricare două vârfuri (noduri) ale sale există un lanț) în care fiecare muchie are asociată un cost (o pondere sau lungime) pozitivă. Trebuie să se identifice o submulțime $A \subseteq E$ astfel încât toate vârfurile din *V* să rămână conectate atunci când sunt folosite doar muchii din *A*, iar suma lungimilor muchiilor din *A* să fie minimă. Deci graful parțial $G_1=(V, A)$ este un **arbore parțial de cost minim** al grafului G.
- costul unui arbore este dat de suma lungimilor muchiilor sale;
- un graf *G=(V, E)* conține un arbore parțial dacă și numai dacă este conex;
- un graf poate avea mai mulți arbori parțiali de cost minim;
- o mulțime de muchii constituie soluție dacă formează arbore parțial și nu conține cicluri.

ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

- Folosind **metoda lui Kruskal**, arborele parțial de cost minim se poate construi astfel: se alege mai întâi muchia de cost minim, iar apoi se adaugă în mod repetat muchia de cost minim nealeasă anterior, cu condiția să nu formeze ciclu cu muchiile deja alese.
-





ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

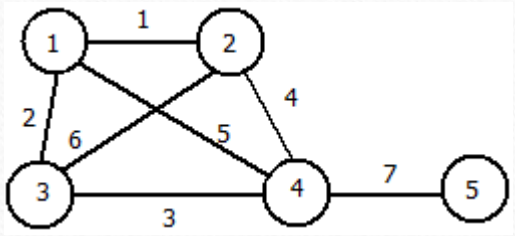
(1) defineşte struct muchie cu datele membru: nod x, nod y şi cost c; muchie m[50],aux; int nr_noduri,nr_muchii,v[100],ax,ay;
(2) citeşte de la tastatură nr_noduri, nr_muchii, capetele x şi y ale muchiile (m[i].x, m[i].y) şi costul asociat muchiei (m[i].c);
(3) sortează crescător muchiile grafului funcţiei de costul asociat acestora prin definirea funcţiei void sortare_muchii();
(4) afişează muchiile grafului în ordinea crescătoare a costurilor prin definirea funcţiei void afisare_muchii_sortate();
(5) defineşte funcţia int Kruskal() după cum urmează:
(5.1) pentru i← 1,..., nr_noduri execută v[i] ← i; //indicele subarborelui din care face parte nodul i
(5.2) s←0; nr (numărul muchiilor adăugate în arbore) ←0;
(5.3) pentru i←1,..., nr_muchii şi dacă nr<nr_noduri execută
(5.4) dacă v[m[i].x]<>v[m[i].y]) atunci //dacă extremităţile unei muchii fac parte din subarbori diferiţi
(5.5) s←s+m[i].c;
(5.6) ax←v[m[i].x]; ay←v[m[i].y]; //reuniunea subarborilor
(5. 7) pentru j←1, ..., nr_noduri execută
(5.8) dacă v[j]==ay atunci v[j]=ax;
(5.9) returnează s;

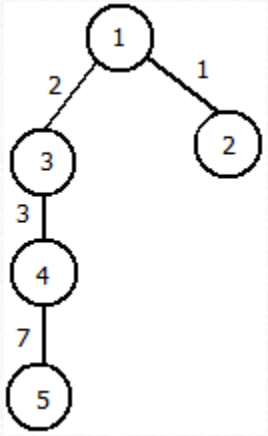
Pseudocod Kruskal

Implementare Kruskal- pag. 202.

ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

Pas	Muchie	Componente conexe ale $G_1=(V, A)$
inițializare	-	{1}, {2}, {3}, {4}, {5}
1	(1, 2)	{1, 2}, {3}, {4}, {5}
2	(1, 3)	{1, 2, 3}, {4}, {5}
3	(3, 4)	{1, 2, 3, 4}, {5}
4	(2, 4)	respinsă (formează ciclu)
5	(1, 4)	respinsă (formează ciclu)
6	(2, 3)	respinsă (formează ciclu)
7	(4, 5)	{1, 2, 3, 4, 5}





ALGORITMI PENTRU DETERMINAREA ARBORELUI PARŢIAL DE COST MINIM

- Altfel, **algoritmul lui Kruskal** presupune:
  - ordonarea crescătoare a muchiilor grafului neorientat *G=(V, E)* în funcţie de costul asociat acestora;
  - pentru fiecare muchie a grafului ordonată crescător, funcţie de cost, se verifică următoarele:
    - dacă extremităţile muchiei analizate fac parte din acelaşi subarbore, atunci muchia se ignoră;
    - dacă extremităţile muchiei analizate fac parte din subarbori diferiţi, atunci arborii se vor reuni, iar muchia considerată va face parte din arborele parţial de cost minim.
  - pentru a verifica dacă extremităţile unei muchii (muchie curentă) aparţin sau nu unui aceluiaşi subarbore, s-a definit vectorul v[i].

ALGORITMI PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

- **Algoritmul lui Prim** construiește arborele parțial de cost minim pornind de la un vârf dat (ce devine rădăcina arborelui) și adăugând la fiecare pas muchia cea mai scurtă (cu cel mai mic cost), care prezintă o extremitate în arborele dezvoltat și cealaltă extremitate printre vârfurile neselectate. Practic, se alege în mod aleatoriu un nod și se identifică muchia adiacentă lui cu costul cel mai mic, muchie care se adaugă în arbore, proces care se continuă evitând ciclurile până când numărul de noduri din arborele parțial este egal cu numărul de noduri din graful *G*.
- Sintetizând, **algoritmul lui Prim** constă în următorii pași:

1. se alege în mod arbitrar un nod, ce va reprezenta rădăcina arborelui parțial;

2. se identifică muchiile incidente nodurilor deja aflate în arbore, respectiv se alege un nod neadăugat încă în arbore pentru care muchia dintre el și un nod din arbore are cost minim;

3. se adaugă în arborele parțial de cost minim muchiile cu cel mai mic cost.

ALGORITMI PENTRU DETERMINAREA ARBORELUI PARŢIAL DE COST MINIM

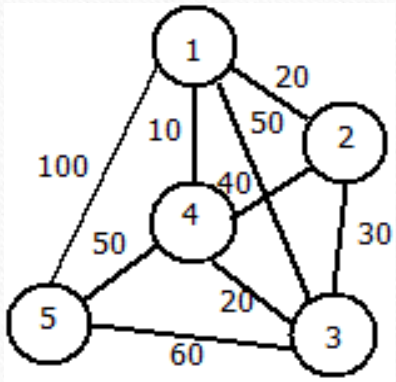
```
(1) iniţializare A← ∅; { A va conţine muchiile arborelui parţial de cost minim }
(2) U←{un vârf oarecare din V};
(3) cât timp U<>V execută
(4)     identifică muchia (u, v) de cost minim astfel încât u∈V\U şi v∈ U;
(5)     A ← A ∪ {(u, v)};
(6)     U ← U ∪ {u};
(7) returnează A;
```

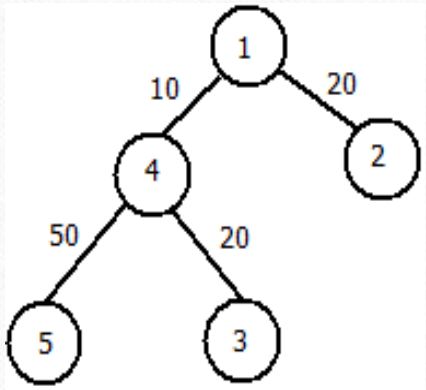
Pseudocod algoritmul lui Prim

Pentru implementarea **algoritmului lui Prim** se folosesc următoarele elemente:

- mulţimea vârfurilor V ce conţine n elemente;
- mulţimea costurilor $C(c_{ij})$;
- două tablouri unidimensionale U şi V;
- pentru fiecare $i \in V \setminus U$, *vecin[i]* conţine vârful din U care este conectat de i printr-o muchie de cost minim, tabloul *mincost[i]* furnizând acest cost;
- pentru $i \in U$, *mincost[i]=-1;*
- mulţimea U este iniţializată cu {1}.

ALGORITMI PENTRU DETERMINAREA ARBORELUI PARŢIAL DE COST MINIM





Graf neorientat şi arborele parţial de cost minim asociat

- Testarea aplicațiilor 8.2.1, 8.2.2 și 8.2.3, pag. 197-203;
- Aplicații propuse: aplicațiile din cadrul lucrării de laborator nr. 12, pag. 204;
- **Obs: se va utiliza cartea "*Elemente de proiectarea algoritmilor. Ghid teoretic și practic*", Șef lucr. dr. mat. Cărbureanu Mădălina, Editura Universității Petrol-Gaze din Ploiești, 2021.**

Să vă fie de folos și spor la lucru!