

Laborator 1 – Algoritmica Grafurilor/Grafelor *

Informatica Romana

**grafuri/grafe – sunt acceptate ambele forme de plural*

Informatii laborator

1. 6 prezente din 7 posibile

Prezenta la laborator se acorda pe o mini-solutie din tema predata in timpul laboratorului.

2. 5 teme de laborator, 0.2 bonus pe fiecare tema => 1 pct final bonus + nota finala (doar daca nota finala ≥ 5)

Notiuni introductive/recapitulative

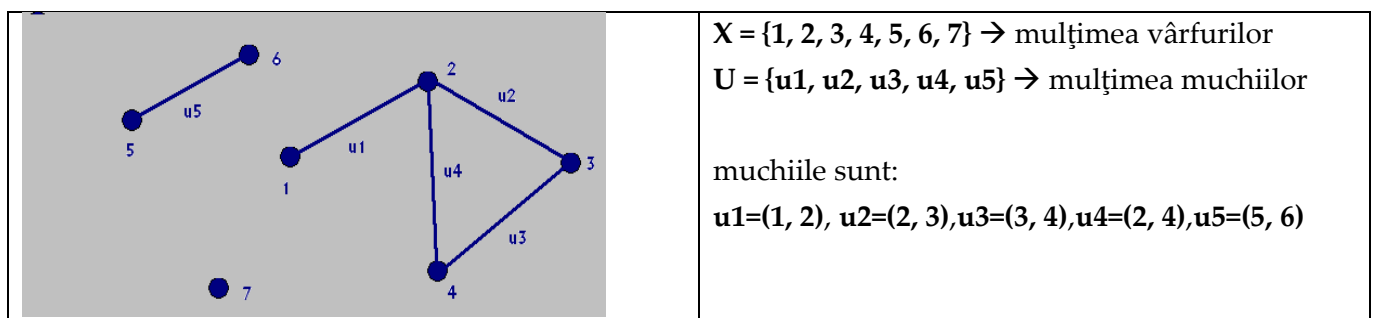
Graf neorientat = pereche ordonată de mulțimi (X, U) astfel:

- * X este o mulțime finită și nevidă de elemente numite *vârfuri* sau *noduri*
- * U este o mulțime de perechi neordonate de câte două elemente din X , numite *muchii* sau *arce*.

Conform literatura de specialitate:

- * pentru grafuri neorientate termenii de *vârf* și *muchie*
- * pentru grafurile orientate termenii de *nod* și *arc*

Exemplu: Pentru graful $G=(X,U)$ de mai jos avem:



Pe caz general, într-un graf neorientat $G=(X, U)$, se noteaza:

- $m \rightarrow$ numărul muchiilor
- $n \rightarrow$ numărul vârfurilor
- $X = \{x_1, x_2, \dots, x_n\} \rightarrow$ mulțimea vârfurilor
- $U = \{u_1, u_2, \dots, u_m\} \rightarrow$ mulțimea muchiilor
- muchia u_k este o pereche neordonată (a,b) alcătuită din două elemente din X

Pentru o muchie $u_k=(a,b)$, vom spune că:

- vârfurile **a** și **b** sunt *adiacente* și se numesc *extremitățile* muchiei u_k
- muchia u_k și vârful **a**, respectiv vârful **b**, sunt *incidente* în graf
- muchia **(a, b)** este totuna cu muchia **(b, a)** (nu există o orientare a muchiei)

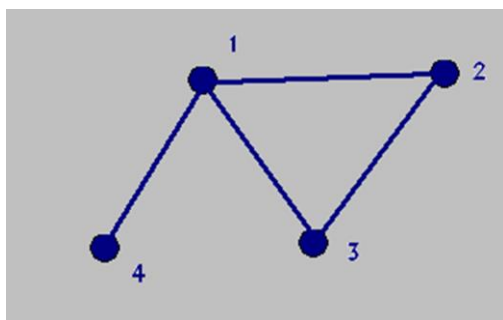
Forme de reprezentare a lui graf neorientat:

Matricea de adiacență

Este o matrice pătratică binară $A_{n \times n}$ în care un element a_{ij} este definit astfel:

$$a[i][j] = \begin{cases} 1, & \text{dacă există muchia } [i,j] \text{ cu } i \neq j \\ 0, & \text{dacă nu există muchia} \end{cases}$$

Exemplu: pentru graful următor, matricea de adiacență este:



		1	2	3	4
liniile	1	0	1	1	1
	2	1	0	1	0
	3	1	1	0	0
	4	1	0	0	0
		coloanele			

Obs:

- * În caz general, $a[i][j]=a[j][i]$ oricare ar fi $i,j \in \{1,2,\dots,n\}$, cu $i \neq j$, adică, pentru orice graf neorientat, matricea de adiacență **a** este *simetrică față de diagonala principală*.
- * Pe diagonala principală a matricii de adiacență vom avea doar valori de 0 deci nu avem muchii de forma **(i,i)** numite **bucle**.
- * Suma elementelor de pe linia **x** sau de pe coloana **x** din matricea de adiacență reprezintă *gradul vârfului x*.
- * Muchiile distincte din graf se regăsesc în jumătatea superioară a matricii de adiacență.
- * Suma tuturor valorilor din matricea de adiacență este număr par egal cu de două ori numărul muchiilor din graf.

Matricea de incidență

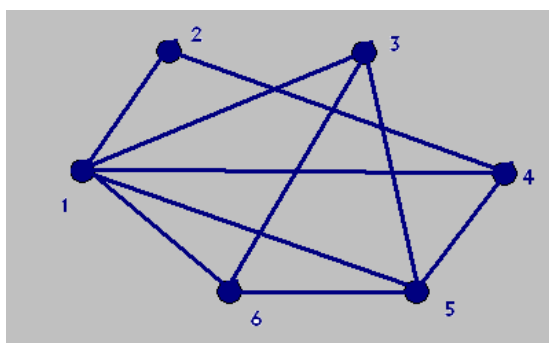
Pentru graful $G=(X, U)$ cu n vârfuri și m muchii, matricea de incidență a are n linii și m coloane și se definește astfel:

$$a[i][j]= \begin{cases} 1, & \text{dacă vârful } x_i \text{ este incident cu muchia } m_j \\ 0, & \text{în caz contrar} \end{cases}$$

Fie graful $G=(X,U)$ din figura alăturată cu

$X=\{1, 2, 3, 4, 5, 6\}$ și

$U=\{[1,2], [1,3], [1,4], [1,5], [1,6], [2,4], [3,5], [3,6], [4,5], [5,6]\}$



Pentru acest graf, asociind fiecăruia dintre vârfuri câte o linie a matricei și fiecărei muchii câte o coloană, se obține matricea de incidență:

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
x_1	1	1	1	1	1	0	0	0	0	0
x_2	1	0	0	0	0	1	0	0	0	0
x_3	0	1	0	0	0	0	1	1	0	0
x_4	0	0	1	0	0	1	0	0	1	0
x_5	0	0	0	1	0	0	1	0	1	1
x_6	0	0	0	0	1	0	0	1	0	1

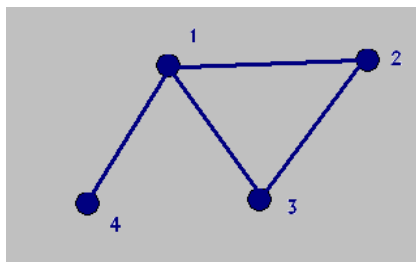
- Fiecare coloană din matricea de incidență conține exact două valori nenule (ambele extrimitati ale muchiei).
- Suma elementelor de pe linia x din matricea de incidență reprezintă gradul vârfului x .

Pentru memorarea în programe a matricei de incidență se folosesc matrici binare $A_{n \times m}$. Matricea de incidență se poate construi prin citirea muchiilor dintr-un fișier text. Muchiile se vor numerota în ordinea citirii din fișier și fiecare muchie este incidentă cu extremitățile ei.

Listele de adiacenta

Pentru fiecare vârf $i \in \{1, 2, \dots, n\}$ formăm lista de adiacenta, adică listele vecinilor lui i . Aceasta cuprinde toate vârfurile care sunt extremități ale muchiilor ce trec prin vârful i .

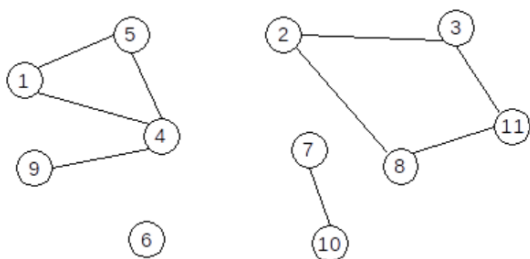
Pentru graful $G=(X, U)$ din figura următoare, lista vecinilor este:



vârful	lista vecinilor
1	2,3,4
2	1,3
3	1,2
4	1

Se observa că fiecare linie i din listele vecinilor conține indicii coloanelor pe care se găsesc valori de 1 în linia i a matricei de adiacență.

Varfuri izolate. Graf regular. Gradul unui vârf



6 este varf izolat

Într-un graf neorientat se numește **grad** al unui vârf numărul de vârfuri adiacente cu acesta (sau numărul de muchii incidente cu acesta).

Gradul unui varf x se notează $d(x)$ (*degree*).

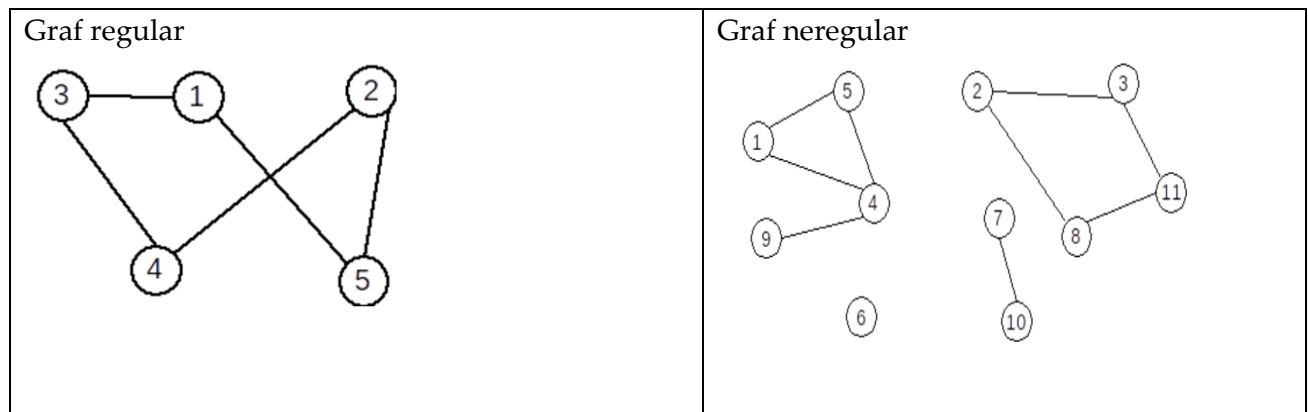
Vârful cu gradul 0 se numește varf **izolat**. În graful de mai sus, vârful 6 este izolat.

Un vârf cu gradul 1 se numește **terminal**. În graful de mai sus, vârful 9 este vârf terminal.

Gradul maxim al unui vârf într-un graf cu n vârfuri este $n-1$.

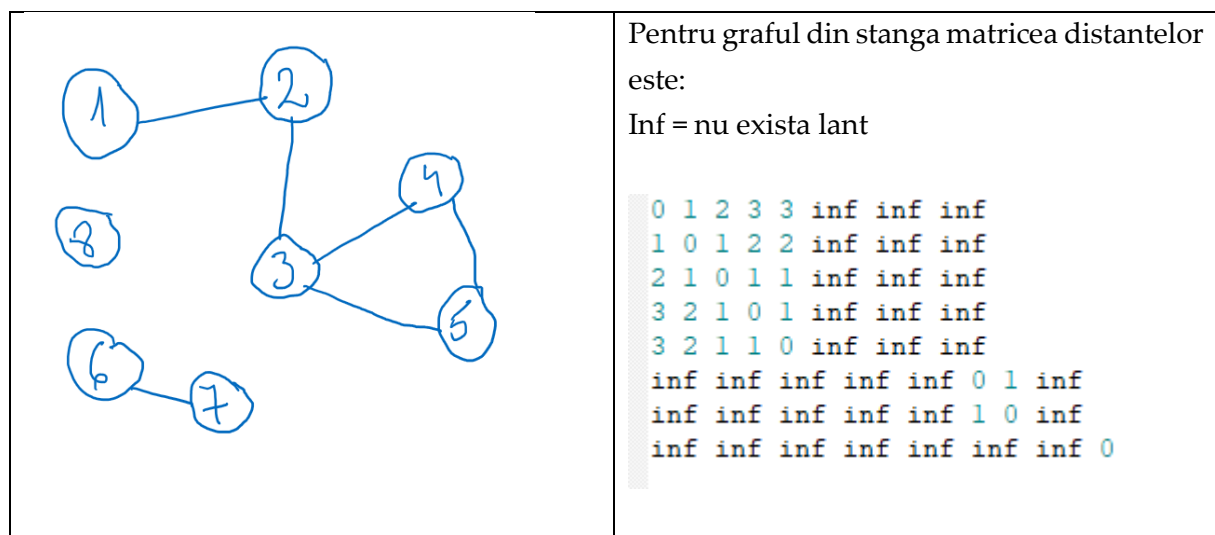
Într-un graf neorientat, suma gradelor tuturor vârfurilor este dublul numărului de muchii.

Un graf în care toate vârfurile au același grad se numește **graf regular**.



Matricea distantelor

Pornind de la un graf neorientat cu n varfuri si m muchii, sa se determine matricea distantelor minime pentru distantele intre oricare 2 varfuri din graf.



Daca nu exista lant intre doua varfuri atunci se va afisa inf (infinit).

Algoritmul Roy-Floyd:

- * pentru oricare pereche de vârfuli i, j trebuie sa se obtina distante mai mici prin vârfuli intermediare k ($k = 1 \dots n$)
- * Se compara "lungimea" lantului $a[i, j]$ cu lungimea lantului care trece prin k si daca:
 $a[i, j] > a[i, k] + a[k, j]$ atunci: $a[i, j] = a[i, k] + a[k, j]$

```

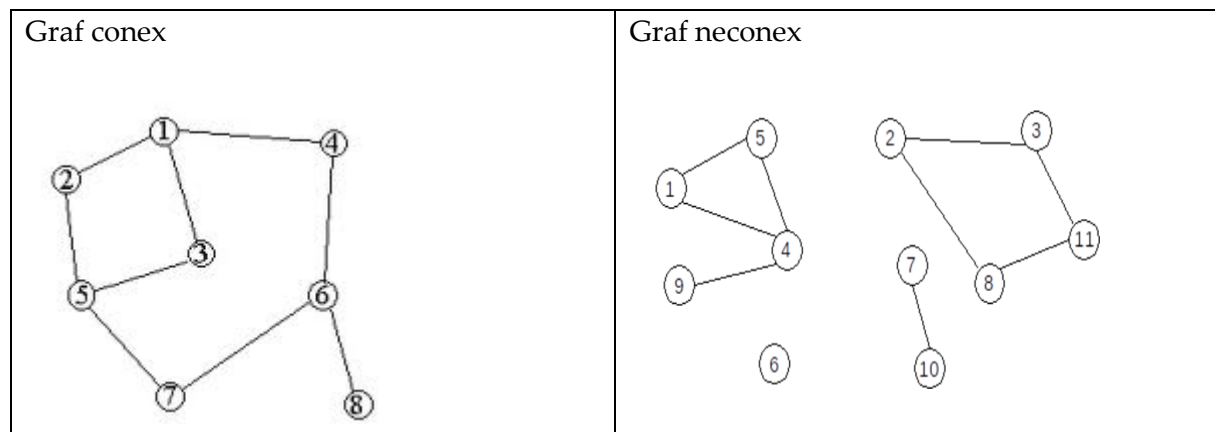
roy_floyd ()
  i, j, k
  pentru k=1,n executa
    pentru i=1,n executa
      pentru j=1,n executa
        daca (A[i][k]<>INF si A[k][j]<>INF) atunci
          daca A[i][j]>A[i][k]+A[k][j] atunci
            A[i][j] ← A[i][k] + A[k][j]

```

Graf neorientat conex

Un graf se numește conex dacă oricare ar fi x și y varfuri din graf există lanț între x și y .

Se numește **lanț** o succesiune de vârfuri cu proprietatea că oricare două vârfuri consecutive sunt adiacente.



Cel mai simplu mod pentru a decide dacă un graf este conex este să parcurgem graful pornind de la un vârf. Dacă se vizitează toate vârfurile atunci graful este conex.

```

void DFS(int x)
{ int i;
  viz[x]=1;
  for(i=1;i<=n;i++)
    if (a[x][i]==1 && viz[i]==0) DFS(i);
}

int Conex()
{ int i;
  DFS(1);
  for(i=1;i<=n;i++)
    if (viz[i]==0) return 0;
  return 1;
}

```

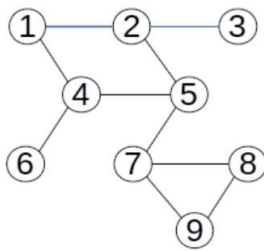
DFS - parcurgerea în adâncime

DFS reprezintă explorarea “naturală” a unui graf neorientat. Este foarte asemănătoare cu modul în care un turist vizitează un oraș în care sunt obiective turistice (vârfurile grafului) și căi de acces între obiective (muchii).

Vizitarea orașului va avea loc din aproape în aproape: se pleacă de la un obiectiv de pornire, se continuă cu un obiectiv învecinat cu acesta, apoi unul învecinat cu al doilea, etc.

Parcurgerea în adâncime se face astfel:

- * Se începe cu un vârf inițial x , care este în acest moment **vârf curent**.
- * Vârful x se vizitează. Se determină primul său vecin nevizitat y al lui x , care devine vârf curent.
- * Apoi se vizitează primul vecin nevizitat al lui y , și așa mai departe, mergând în adâncime, până când ajungem la un vârf care nu mai are vecini nevizitați. Când ajungem într-un astfel de vârf, ne întoarcem la “părintele” acestuia – vârful din care am ajuns în acesta.
- * Dacă acest vârf mai are vecini nevizitați, alegem următorul vecin nevizitat al său și continuăm parcurgerea în același mod.
- * Dacă nici acest vârf nu mai are vecini nevizitați, revenim în vârful său părinte și continuăm în același mod, până când toate vârfurile accesibile din vârful de start sunt vizitate.



- Parcurgerea din nodul 1: 1 2 3 5 4 6 7 8 9
- Parcurgerea din nodul 2: 2 1 4 5 7 8 9 6 3
- Parcurgerea din nodul 9: 9 7 5 2 1 4 6 3 8

Pentru implementarea algoritmului se folosește un vector caracteristic pentru memorarea faptului că un anumit vârf a fost sau nu vizitat, la un anumit moment al parcurgerii:

- $v[i] = 0$, vârful i nu a fost (încă) vizitat
- $v[i] = 1$, vârful i a fost vizitat

Presupunem că graful are n vârfuri și este prezentat prin matricea de adiacență a . Starea unui vârf (vizitat sau nu) este memorată în vectorul caracteristic v .

```
void dfs(int k)
{
    v[k]=1; //vizitam varful curent x
    for(int i=1;i<=n;i++) // determinam vecinii nevizitati ai lui x
        if(a[k][i]==1 && v[i]==0)
        {
            dfs(i); // continuam parcurgerea cu vecinul curent i
        }
}
```

Cerinte L1:

1.

Se da un graf neorientat în fișier sub formă: nr varfuri și număr muchii iar apoi lista de muchii.

Sa se scrie un program care sa realizeze conversia de la o formă de reprezentare la o alta formă de reprezentare.

2.

Fie un graf neorientat reprezentat sub o anumită formă (graful este citit dintr-un fișier).

Sa se rezolve:

- sa se determine varfurile izolate dintr-un graf.
- sa se verifice dacă graful este regulat.
- pentru un graf reprezentat cu matricea de adiacență sa se determine matricea distanțelor.
- pentru un graf reprezentat cu o matrice de adiacență sa se determine dacă este conex.

La final, creați un director de formă (Nr Semigrupa Nume Prenume, ex 217-1 Coroiu Adriana) aici

https://ubbcluj-my.sharepoint.com/:f:/g/personal/adriana_coroiu_ubbcluj_ro/Em4URjojuhxPvaoXNYdVYAIBiYVki_1eVX8DVtLxecJVPw?e=dsuyFR

și încărcați soluțiile (doar cpp) cu numărul laboratorului și nr problemei: ex. L1p2