Laborator 1 – Algoritmica Grafurilor/Grafelor *
Informatica Romana

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

Informatii laborator

- 6 prezente din 7 posibile: Prezenta la laborator se acorda pe o mini-solutie din tematica alocata fiecarui laborator.
- 5 teme de laborator, 0.2 bonus pe fiecare tema => 1 pct final bonus + nota finala (doar daca nota finala >=5)
- 2 teste practice:
  - in sapt. 7/8
  - in sapt. 13/14

Notiuni introductive/recapitulative

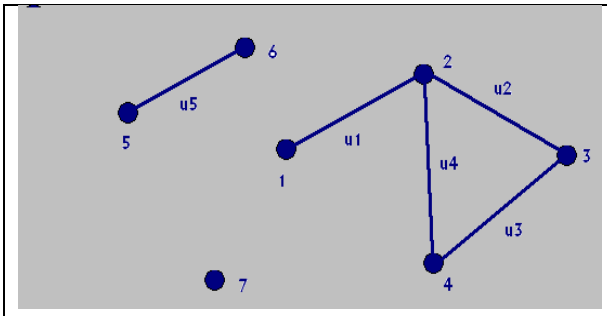
Graf neorientat = pereche ordonată de mulţimi **(X, U)** astfel:

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

Conform literatura de specialitate:

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

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



X = {1, 2, 3, 4, 5, 6, 7} → mulţimea vârfurilor
U = {u1, u2, u3, u4, u5} → mulţimea muchiilor

muchiile sunt:
u1=(1, 2), u2=(2, 3),u3=(3, 4),u4=(2, 4),u5=(5, 6)

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

- **m**→ numărul muchiilor
- **n**→ numărul vârfurilor
- **X= {x_1, x_2,……., x_n}** → mulţimea vârfurilor

- **U={u_1, u_2,……, u_m}→** mulţimea 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** şi **b** sunt ***adiacente*** şi se numesc ***extremităţile*** muchiei **u_k**
- muchia **u_k** şi vârful **a**, respectiv vârful **b** , sunt ***incidente*** în graf
- muchia **(a, b)** este totuna cu muchia **(b, a)** (nu există o orientare a muchiei)

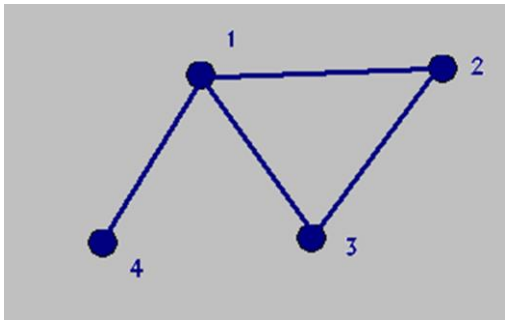
Forme de reprezentare a lui graf neorientat:

Matricea de adiacenţă

Este o matrice pătratică binară A_{nxn} în care un element a_{ij} este definit astfel:

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

Exemplu: pentru graful următor, matricea de adiacenţă este:



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

coloanele

Obs:

- In caz general, **a[i][j]=a[j][i]** oricare ar fi **i,j∈{1,2,……..,n}** , cu **i≠j**, adică, pentru orice graf neorientat, matricea de adiacenţă **a** este *simetrică faţă de diagonala principală*.

- Pe diagonala principală a matricii de adiacenţă 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 adiacenţă reprezintă *gradul vârfului* **x**.

- Muchiile distincte din graf se regăsesc în jumătatea superioară a matricei de adiacenţă.

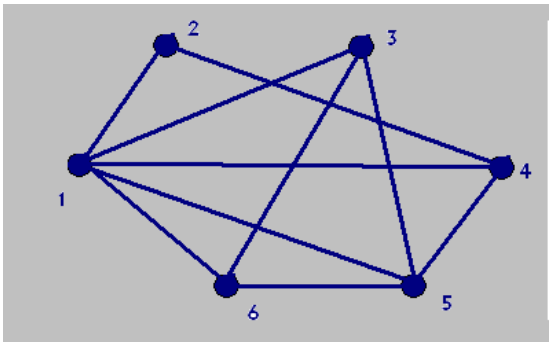
- Suma tuturor valorilor din matricea de adiacenţă este număr par egal cu de două ori numărul muchiilor din graf.

Matricea de incidenţă

Pentru graful **G=(X, U)** cu **n** vârfuri şi **m** muchii, matricea de incidenţă **a** are **n** linii şi **m** coloane şi se defineşte astfel:

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

Fie graful **G=(X,U)** din figura alăturată cu
X={1, 2, 3, 4, 5, 6} şi
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 şi fiecărei muchii câte o coloană, se obţine matricea de incidenţă:

	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

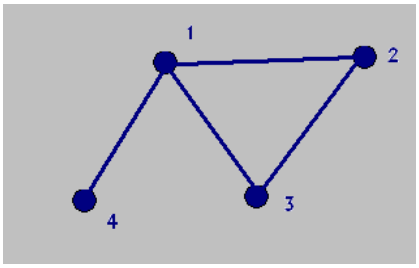
- Fiecare coloană din matricea de incidenţă conţine exact două valori nenule (ambele extrimitati ale muchiei).
- Suma elementelor de pe linia **x** din matricea de incidenţă reprezintă gradul vârfului **x**.

Pentru memorarea în programe a matricei de incidenţă se folosesc matrici binare $\mathbf{A}_{nxm}$. Matricea de incidenţă se poate construi prin citirea muchiilor dintr-un fişier text. Muchiile se vor numerota în ordinea citirii din fişier şi fiecare muchie este incidentă cu extremităţile ei.

Listele de adiacenta

Pentru fiecare vârf **i∈{1, 2,……, n}** formăm lista de adicenta, adica liestele vecinilor lui **i**. Aceasta cuprinde toate vârfurile care sunt extremităţi ale muchiilor ce trec prin vârful **i**.

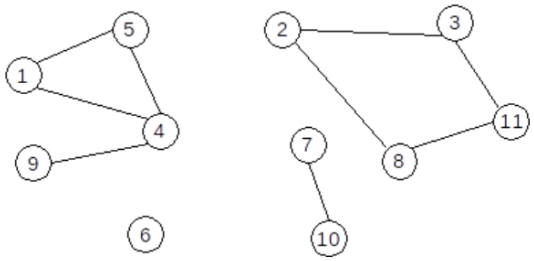
Pentru graful **G=(X, U)** din figura următoare, lista vecinilor este:



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

Se observa ca că fiecare linie **i** din listele vecinilor conţine indicii coloanelor pe care se găsesc valori de **1** în linia **i** a matricei de adiacenţă.

Varfuri izolate. Graf regular. Gradul unui vârf



6 este varf izolat

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

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

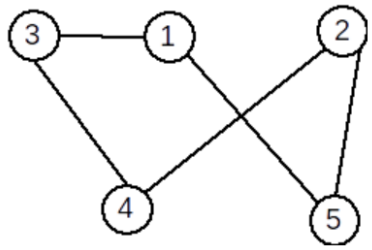
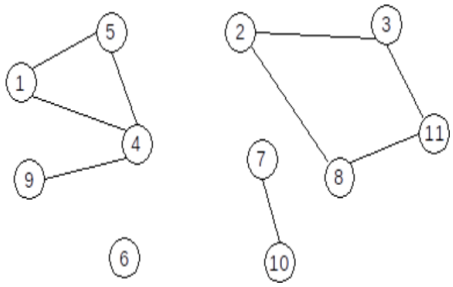
Vârful cu gradul 0 se numește varf **izolat**. În graful de mai sus, vârful 6 este izolat.

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.

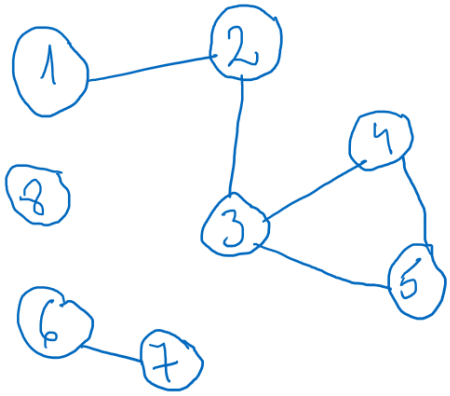
Într-un graf neorientat, suma gradelor tuturor vârfurilor este dublul numărului de muchii.

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

Graf regular	Graf neregular
	

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.

	Pentru graful din stanga matricea distantelor este: Inf = nu exista lant 0 1 2 3 3 inf inf inf 1 0 1 2 2 inf inf inf 2 1 0 1 1 inf inf inf 3 2 1 0 1 inf inf inf 3 2 1 1 0 inf inf inf inf inf inf inf inf 0 1 inf inf inf inf inf inf 1 0 inf inf inf inf inf inf inf inf 0

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

Algoritmul Roy-Floyd:

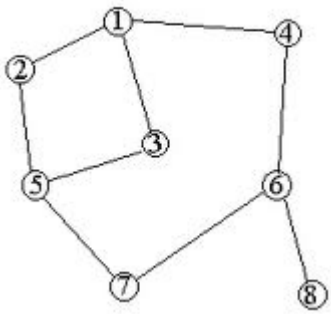
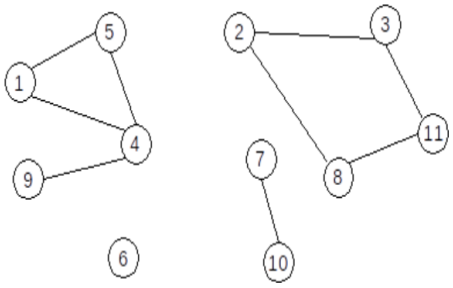
- pentru oricare pereche de vârfuri i,j trebuie sa se obtina distante mai mici prin vârfuri intermediare k (k =1...n)
- Se compara "lungimea" lantului a[i,j] cu lungimea lantului care trece prin k si daca:
  a[i,j] > a[i,k]+a[k,j] atunci: a[i,j] = a[i,j] > a[i,k]+a[k,j]

```
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 numeste conex daca oricare ar fi x si y varfuri din graf exista lant intre x si y.

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

Graf conex	Graf neconex
	

Cel mai simplu mod pentru a decide daca un graf este conex este sa parcurgem graful pornind de la un vârf. Daca se viziteaza toate vârfurile atunci graful este conex.

Se poate folosi DFS pentru verificarea conexitatii:

```
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;
}
```

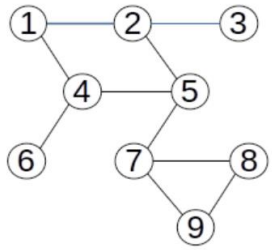
DFS - parcurgerea în adâncime

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 (muchiile).

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, şi aşa 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 continuam 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 foloseşte un vector caracteristic pentru memorarea faptului că un anume 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
    }
}
```