

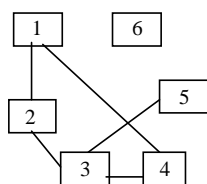
Grafuri neorientate

Graf = orice mulțime finită V prevăzută cu o relație binară internă E . Notăm graful cu $G=(V, E)$.

Graf neorientat = un graf $G=(V, E)$ în care relația binară este simetrică: $(v, w) \in E$ atunci $(w, v) \in E$.

Nod = element al mulțimii V , unde $G=(V, E)$ este un graf neorientat.

Muchie = element al mulțimii E ce descrie o relație existentă între două vârfuri din V , unde $G=(V, E)$ este un graf neorientat;



În figura alăturată:

$V=\{1,2,3,4,5,6\}$ sunt noduri

$E=\{[1,2], [1,4], [2,3],[3,4],[3,5]\}$ sunt muchii

$G=(V, E)=(\{1,2,3,4,5,6\}, \{[1,2], [1,4], [2,3],[3,4],[3,5]\})$

Adiacenta: Într-un graf neorientat existența muchiei (v, w) presupune că w este adiacent cu v și v adiacent cu w .

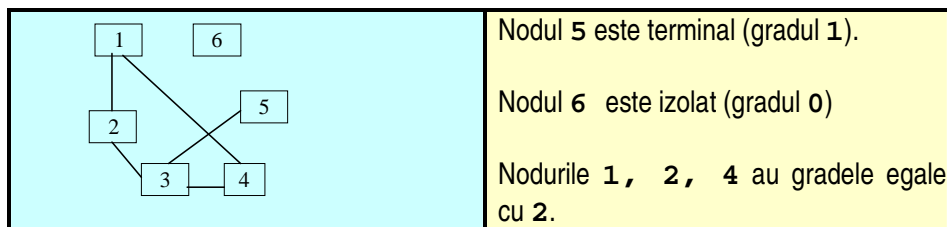
În exemplul din figura de mai sus vârful 1 este adiacent cu 4 dar 1 și 3 nu reprezintă o pereche de vârfuri adiacente.

Incidență = o muchie este incidentă cu un nod dacă îl are pe acesta ca extremitate. Muchia (v, w) este incidentă în nodul v respectiv w .

Grad = Gradul unui nod v , dintr-un graf neorientat, este un număr natural ce reprezintă numărul de noduri adiacente cu acesta (sau numărul de muchii incidente cu nodul respectiv)

Nod izolat = Un nod cu gradul 0.

Nod terminal= un nod cu gradul 1



Lanț = este o secvență de noduri ale unui graf neorientat $G=(V, E)$, cu proprietatea că oricare două noduri consecutive din lanț sunt adiacente:

$L=[w_1, w_2, w_3, \dots, w_n]$ cu proprietatea că $(w_i, w_{i+1}) \in E$ pentru $1 \leq i < n$.

Lungimea unui lanț = numărul de muchii din care este format.

Lanț simplu = lanțul care conține numai **muchi** distincte

Lanț compus = lanțul care **nu este** format numai din muchii distincte

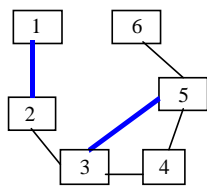
Lanț elementar = lanțul care conține numai **noduri distincte**

Ciclu = Un lanț în care primul nod coincide cu ultimul.

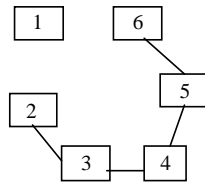
Ciclu este elementar dacă este format doar din noduri distincte, excepție făcând primul și ultimul. Lungimea unui ciclu nu poate fi 2.

	<p>Sucesiunea de vârfuri 2, 3, 5, 6 reprezintă un lanț simplu și elementar de lungime 3. Lanțul 5 3 4 5 6 este simplu dar nu este elementar. Lanțul 5 3 4 5 3 2 este compus și nu este elementar. Lanțul 3 4 5 3 reprezintă un ciclu elementar</p>
--	---

Graf parțial = Dacă dintr-un graf $G=(V,E)$ se suprimă cel puțin o muchie atunci noul graf $G'=(V,E')$, $E' \subset E$ se numește graf parțial al lui G .

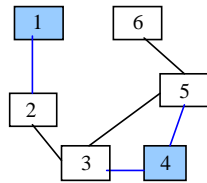


G

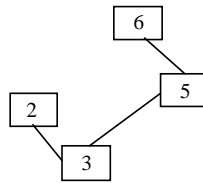


G1 este graf parțial al lui G

Subgraf = Dacă dintr-un graf $G=(V,E)$ se suprimă cel puțin un nod împreună cu muchiile incidente lui, atunci noul graf $G'=(V',E')$, $E' \subset E$ și $V' \subset V$ se numește subgraf al lui G .

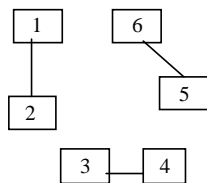


G

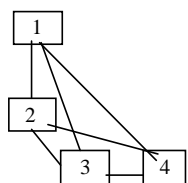


G1 este subgraf al lui G

Graf regulat = graf neorientat în care toate nodurile au același grad;

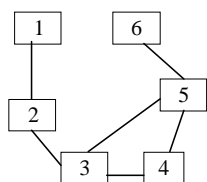


Graf complet = graf neorientat $G=(V, E)$ în care există muchie între oricare două noduri.
 Numărul de muchii ale unui graf complet este: $nr*(nr-1)/2$. Unde nr este numărul de noduri

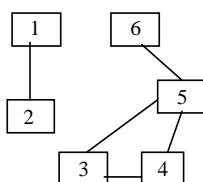


graf complet. Nr de noduri: $4x(4-1)/2 = 6$

Graf conex = graf neorientat $G=(V, E)$ în care pentru orice pereche de noduri (v, w) există un lanț care le unește.

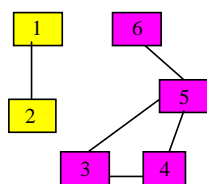


graf conex



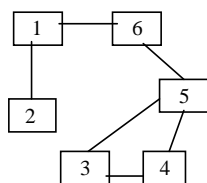
nu este graf conex

Componentă conexă = subgraf al grafului de referință, maximal în raport cu proprietatea de conexitate (între oricare două vârfuri există lanț);



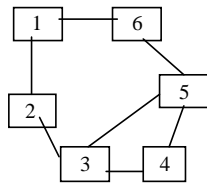
graful nu este conex. Are 2 componente conexe:
 1, 2 și 3, 4, 5, 6

Lanț hamiltonian = un lanț elementar care conține toate nodurile unui graf



L=[2, 1, 6, 5, 4, 3] este lant hamiltonian

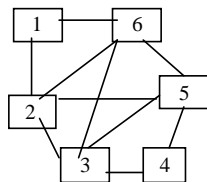
Ciclu hamiltonian = un ciclu elementar care conține toate nodurile grafului



$C=[1,2,3,4,5,6,1]$ este ciclu hamiltonian

Graf hamiltonian = un graf G care conține un ciclu hamiltonian
Graful anterior este graf Hamiltonian.

Lanț eulerian = un lanț simplu care conține toate muchiile unui graf



Lantul: $L=[1.2.3.4.5.3.6.2.5.6]$ este lant eulerian

Ciclu eulerian = un ciclu simplu care conține toate muchiile grafului
Ciclu: $C=[1.2.3.4.5.3.6.2.5.6.1]$ este ciclu eulerian

Graf eulerian = un graf care conține un ciclu eulerian.

Condiție necesară și suficientă: Un graf este eulerian dacă și numai dacă oricare vârf al său are gradul par.

Reprezentarea grafurilor neorientate

Fie $G=(V, E)$ un graf neorientat.

Exista mai multe modalitati de reprezentare pentru un graf neorientat, folosind diverse tipuri de structuri de date. Reprezentarile vor fi utilizate in diversi algoritmi si in programele care implementeaza pe calculator acesti algoritmi.

Matricea de adiacentă matricea booleană

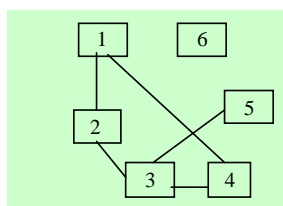
Matricea de adiacentă asociată unui graf neorientat cu n noduri se definește astfel: $A = (a_{ij})_{n \times n}$ cu

$$a_{ij} = \begin{cases} 1, & \text{daca } [i, j] \in E \\ 0, & \text{altfel} \end{cases}$$

Observatii:

- - Matricea de adiacenta asociată unui graf neorientat este o matrice simetrică
- - Suma elementelor de pe linia k reprezintă gradul nodului k
- - Suma elementelor de pe coloana k reprezintă gradul nodului k

Fie graful din figura urmatoare:



$A =$	0	1	0	1	0	0	nodul 1 are gradul 2
	1	0	1	0	0	0	nodul 2 are gradul 2
	0	1	0	1	1	0	nodul 3 are gradul 3
	1	0	1	0	0	0	nodul 4 are gradul 2
	0	0	1	0	0	0	nodul 5 are gradul 1
	0	0	0	0	0	0	nodul 6 are gradul 0

Numarul de noduri este 6 si numarul de muchii este 4

Matricea este simetrica si patratica avand 6 linii si 6 coloane

Diagonala principala contine numai valori nule

Pentru a prelucra graful se citesc:

6- reprezentand n , numarul de noduri

4- reprezentand m , numarul de muchii

4 perechi $x-y$ reprezentand extremitatile celor 4 muchii:

$$1-2 \Rightarrow a[1,2]=a[2,1]=1$$

$$1-4 \Rightarrow a[1,4]=a[4,1]=1$$

$$2-3 \Rightarrow a[2,3]=a[3,2]=1$$

$$3-4 \Rightarrow a[3,4]=a[4,3]=1$$

$$3-5 \Rightarrow a[3,5]=a[5,3]=1$$

Probleme propuse:

Problema 1

Se citeste un graf din fisierul graf.txt: numarul de noduri, numarul de muchii si muchiile.

- a) sa se afiseze matricea de adiacente
- b) Sa se determine gradul unui nod citit
- c) Sa se afiseze pentru un nod citit nodurile adiacente
- d) sa se afiseze nodurile incidente cu cea de a x muchie din matrice
- e) sa se afiseze pentru fiecare nod gradul
- f) sa se afiseze nodul (nodurile) avand cei mai multi vecini
- g) sa se afiseze nodurile izolate

Problema 2

Sa se det daca o matrice citita dintr-un fisier poate fi matricea unui graf neorientat. In caz afirmativ se va determina cate muchii are graf

Problema 3

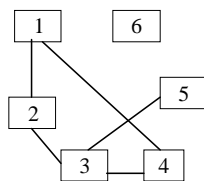
Sa se genereze un graf avand maxim n noduri si maxim m muchii. Sa se afiseze matricea atasata

Observatie: graf

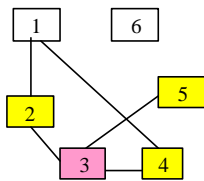
Listele de adiacenta a nodurilor

Reprezentarea in calculator a unui graf se poate face utilizand listele de adiacenta a varfurilor, adica pentru fiecare varf se alcatuieste lista varfurilor adiacente cu el.

Fie graf

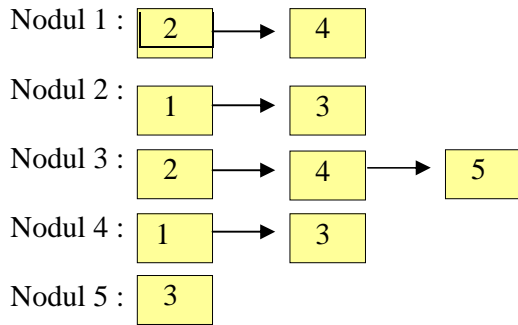


Lista vecinilor nodului 3: 2, 4, 5 (noduri adiacente)



Nodul 6 nu are vecini (este izolat)

Pentru intreg graf



Ordinea nodurilor in cadrul unei liste nu este importanta

Pentru a genera o astfel de lista vom defini tipul nod :

```
struct nod {int nd;
            nod *next;};
```

Toate listele se vor memora utilizand un vector de liste :

```
nod *L[20];
```

Datele de intrare : numarul de noduri si muchiile se vor citi din fisier :

O solutie de implementare este urmatoarea :

```
#include<conio.h>
#include<fstream.h>
struct nod
{ int nd;
  nod *next;};

nod *L[20];

void afisare(int nr_nod) //afiseaza lista vecinilor nodului nr_nod
{ nod *p=L[nr_nod];
  if(p==0)
    cout<<nr_nod<<" este izolat "<<endl;
  else
    { cout<<"lista vecinilor lui "<<nr_nod<<endl;
      nod *c=p;
      while(c)
        { cout<<c->nd<<" ";
          c=c->next;}
      cout<<endl;}
}

void main()
```

```

{fstream f;int i,j,n;
nod *p,*q;

f.open("graf.txt",ios::in); //citirea datelor din fisier
clrscr();
f>>n;
while(f>>i>>j)
{p=new nod; //se adauga j in lista vecinilor lui i
p->nd=j;
p->next=L[i];
L[i]=p;
q=new nod; //se adauga i in lista vecinilor lui j
q->nd=i;
q->next=L[j];
L[j]=q;
}
f.close();

cout<<endl<<"listele de adiacente ";
for(i=1;i<=n;i++)
    afisare(i);
getch();
}

```

Observatie : In exemplul anterior adaugarea unui nou element in lista se face inainte celorlalte din lista corespunzatoare.

Aceste doua moduri de reprezentare (prin matrice de adiacenta si prin liste de vecini) se folosesc dupa natura problemei. Adica, daca in problema se doreste un acces frecvent la muchii, atunci se va folosi matricea de adiacenta; daca numarul de muchii care se reprezinta este mult mai mic dect nxn , este de preferat sa se folosesca listele de adiacenta, pentru a se face economie de memorie.

Probleme propuse :

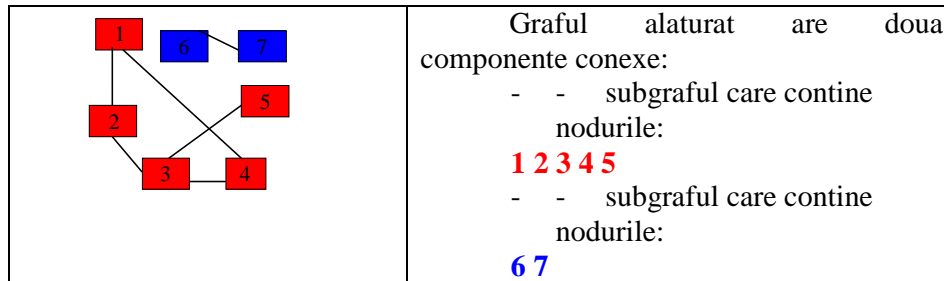
1. 1. Sa se memoreze un graf neorientat utilizand liste de adiacente. Parcurgand listele de adiacenta rezolvati urmatoarele cerinte :
 - a. a. Sa se determine daca graful contine noduri izolate
 - b. b. Sa se determine fradul unui nod citit de la tastatura
 - c. c. Sa se determine nodul cu cel mai mare grad
 - d. d. Sa se determine daca nodurile x si y sunt adiacente
 - e. e. Sa se verifice daca graful este regulat
 - f. f. Sa se determine daca graful este complet

Componente conexe

Fie $G=(V, E)$ un graf neorientat, unde V are n elemente (n noduri) si E are m elemente (m muchii).

Definitie: $G_1=(V_1, E_1)$ este o componenta conexa daca:

- - pentru orice pereche x,y de noduri din V_1 exista un lant de la x la y (implicit si de la y la x)
- - nu exista alt subgraf al lui G , $G_2=(V_2, E_2)$ care sa indeplineasca prima conditie si care sa-l contina pe G_1



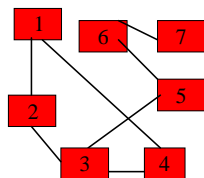
Observatie: subgraful 1, 2, 3, 4 nu este componenta conexa pentru (chiar daca pentru orice pereche x,y de noduri exista un lant de la x la y) deoarece exista subgraful 1, 2, 3, 4, 5 care il contine si indeplineste aceeasi conditie.

Definitie: Un graf $G=(V, E)$ este conex daca pentru orice pereche x,y de noduri din V exista un lant de la x la y (implicit si de la y la x).

Observatii:

- - Un graf este conex daca admite o singura componenta conexa.
- - Graful anterior nu este conex pentru ca admite doua componente conexe

Graful urmator este conex:



Probleme:

1. Fiind dat un graf memorat prin intermediul matricei de adiacenta sa se determine daca graful este conex.
2. Fiind dat un graf memorat prin intermediul matricei de adiacenta si un nod k sa se determine componenta conexa careia ii apartine nodul k
3. Sa se afiseze toate componentele conexe ale unui graf neorientat

Indicatii :

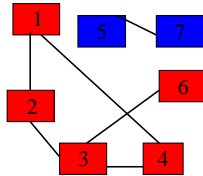
-In vectorul viz se incarca numarul componentei conexe astfel pentru graful urmator, vectorul viz va retine:

viz=[**1,1,1,1,2,1,2**].

-Numarul de componente conexe este 2.

-Se afiseaza componentele cu numarul componentei conexe egal cu 1: **1,2,3,4,6**

-Se afiseaza componentele cu numarul componentei conexe egal cu 2: **5, 7**



-Incarcarea in viz se realizeaza prin parcurgere df pornind de la fiecare nod

-se porneste de la 1, se parcurge df si se incarca in viz valoarea 1 pt nodurile 1, 2, 3, 4, 6. Viz devine:

1,1,1,1,0,1,0

-se porneste in continuare de la primul nod nevizitat, adica 5 si se incarca numarul celei de a doua componente, adica 2

Viz devine: 1,1,1,1,2,1,2

-Nu mai sunt noduri nevizitate si algoritmul se incheie.

Iata o solutie:

```
#include<fstream.h>
#include<conio.h>
int a[20][20],n,m,viz[100],gasit;
int nrc; //pastreaza numarul componentei conexe

void dfmr(int nod)
{ viz[nod]=nrc; //se incarca numarul componentei
  for(int k=1;k<=n;k++)
    if(a[nod][k]==1&&viz[k]==0)
      dfmr(k);
}

void main()
{int x,y,j;
  fstream f;
  clrscr();
  f.open("muchii.txt",ios::in); //memorare matrice de adiacenta
  if(f)
    cout<<"ok!"<<endl;
  else
    cout<<"eroare la deschidere de fisier!";
  f>>n>>m;
  for(int i=1;i<=m;i++)
    {f>>x>>y;
      a[x][y]=a[y][x]=1;}
  cout<<endl<<"matricea de adiacente"<<endl;
  for(i=1;i<=n;i++)
```

```

{ for(j=1;j<=n;j++)
  cout<<a[i][j]<<" ";
cout<<endl;}

for(int nod=1;nod<=n;nod++)
  if(viz[nod]==0) //se incearca parcurgerea numai daca un nod nu a fost deja parcurs
    { nrc++;
      dfmr(nod);}

cout<<endl<<"vectorul viz " <<endl;
for(i=1;i<=n;i++)
  cout<<viz[i]<<" ";

cout<<endl<<"Componentele conexe : " <<endl;
for(i=1;i<=nrc;i++)
  { cout<<endl<<"componenta " <<i<<endl;
    for(j=1;j<=n;j++)
      if(viz[j]==i)
        cout<<j<<" ";
    }
  getch();}

```