Liceul Teoretic “Henri Coandă”, Craiova

**PROIECT ATESTAT**

**Profesor indrumator:**

**Ciucă Aida**

**Elev:**

**Stancu Alexandru Razvan**

**Clasa a XII-a D**

* **2022 -**

**PROIECT ATESTAT**

**GRAFURI NEORIENTATE**

**Modalități de reprezentare și parcurgere**

**Cuprins**

1. **Definirea Grafurilor Neorientate** ...................... **4**
2. **Reprezentarea grafurilor neorientate**

**(Matricea de adiacenta) ................................. 10**

1. **Reprezentarea grafurilor neorientate**

**(Liste de adiacenta)** **............................................ 12**

1. **Parcurgerea grafurilor .................................... 14**
2. **Aplicatie (Rezolvare) ..................................... 19**
3. **Bibliografie .................................................... 24**

**1 -** **Definirea Grafurilor Neorientate**

**Definitie:** Se numeşte graf neorientat o pereche ordonată de multimi notată G=(V, E) unde:

V : este o multime finită şi nevidă, ale cărei elemente se numesc noduri sau vârfuri;

E : este o multime, de perechi neordonate de elemente distincte din V, ale cărei elemente se numesc muchii.

**• Exemplu de graf neorientat**: G=(V, E) unde: V={1,2,3,4} E={{1,2}, {2,3},{1,4}t}

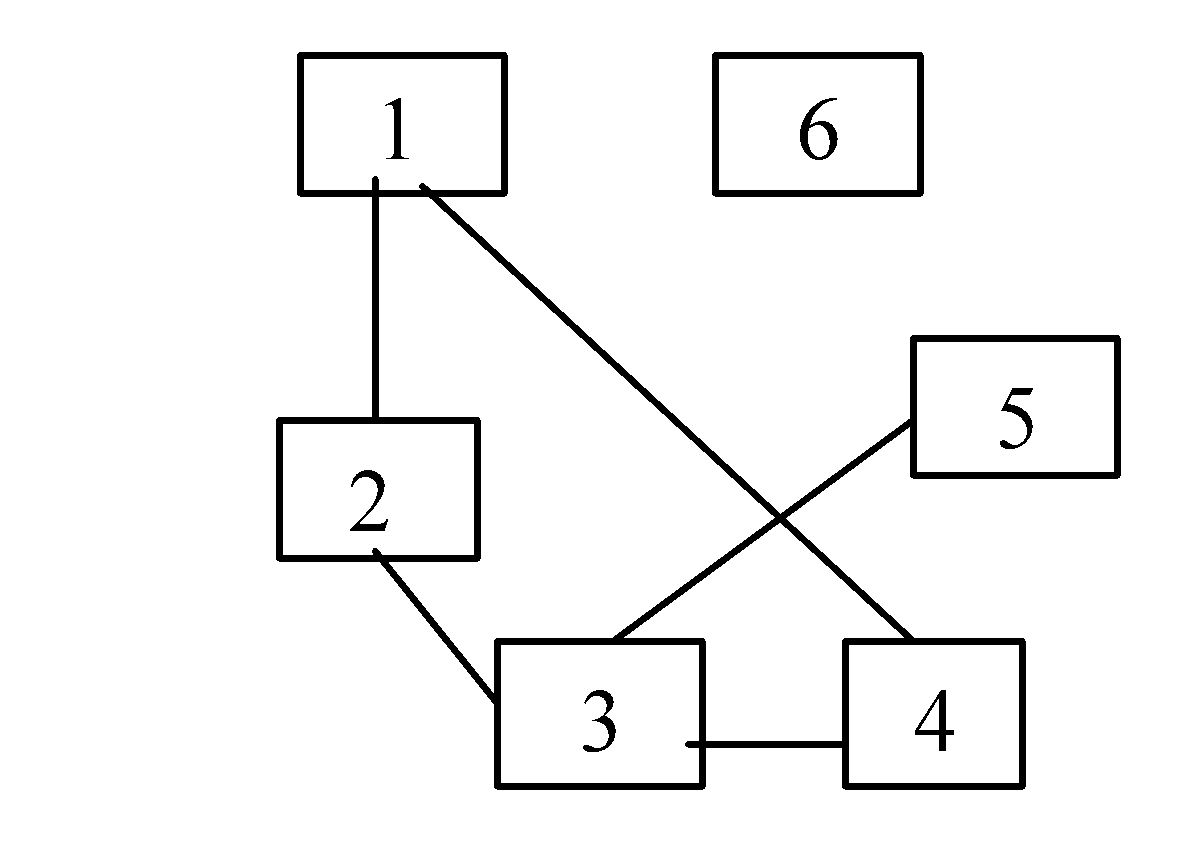
Demonstratie: Perechea G este graf neorientat deoarece respectă definitia prezentată mai sus, adică: V : este finită şi nevidă;

E: este o multime de perechi neordonate (submultimi cu două elemente) de elemente din V.

Intr-un un graf **G=(V, E) neorientat** relaţia binară este simetrică: **(v,w)∈E** atunci **(w,v) ∈E**.

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

**Muchie** = element al mulţimii **E** ce descrie o relaţie existentă între două noduri din **V**, unde **G=(V, E)** este un graf neorientat;



In figura alaturata:

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]})

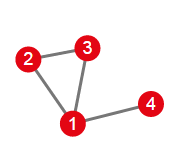
**Adiacenta**: Într-un graf neorientat existenţa muchiei **(v,w)** presupune că **w** este adiacent cu **v** şi **v** adiacent cu **w**.

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

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

**Gradul unui** **vârf**: Fie G=(V, E) un graf neorientat şi x un nod al său. Se numeşte grad al nodului x, numărul muchiilor incidente cu x, notat d(x).

•Exemplu: Fie graful neorientat: G=(V, E) unde: V= {1,2,3,4} E={[l,2], [2,3], [1,4], [1,3]} reprezentat grafic astfel:



Gradul nodului 1 este d(1) şi d(1)=3 (în graf sunt trei muchii incidente cu 1 )

Gradul nodului 2 este d(2) şi d(2)=2 (în graf sunt două muchii incidente cu 2 )

Gradul nodului 3 este d(3) şi d(3)=2 (în graf sunt două muchii incidente cu 3)

Gradul nodului 4 este d(4) şi d(4)=1 (în graf este o singură muchie incidentă cu 4)

**Observatii:** **1**. Dacă gradul unui vârf este 0, vârful respectiv se numeşte **vârf izolat**.

2.Dacă gradul unui vârf este l, vârful respectiv se numeşte **vârf terminal**.

**Notiunea de graf partial**

Definitie. Fie G=(V, M) un graf neorientat. Se numeste **graf partial**, al grafului G, graful neorientat

G1=(V, M1) unde M1 ÍM.

*Concluzie:*

Un graf partial al unui graf neorientat G=(V, M) are aceeasi multime de vârfuri ca si G iar multimea

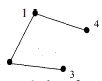
muchiilor este o submultime a lui M sau chiar M.

**Exemplu**: Fie graful neorientat:

G=(V, M) unde: V={ 1,2,3,4}

M={[1,2], [1,4], [2,3]}

reprezentat grafic astfel:

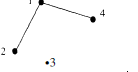


1. Un exemplu de graf partial al grafului G este graful neorientat:

G1=(V, M,) unde: V={1,2,3,4}

M1={[1,2],[1,4]} (s-a eliminat muchia [2,3])

reprezentat grafic astfel:



*Observatie.* Fie G=(V, M) un graf neorientat. Un graf partial, al grafului G, se obtine păstrând vârfurile si eliminând eventual niste muchii (se pot elimina si toate muchiile, sau chiar nici una).

Daca un graf neorientat are m muchii atunci suma gradelor tuturor nodurilor este 2m

In orice graf G exista un numar par de noduri de grad impar

**Lanţ** = este o secvenţă de noduri ale unui graf neorientat **G=(V,E)**, cu proprietatea că oricare două noduri consecutive din secventa lant sunt adiacente:

**L=[w1, w2, w3,. . ,wn]**  cu proprietatea că **(wi, wi+1)∈E** pentru **1≤i<n.**

**Lungimea unui lanţ** = numărul de muchii din care este format.

**Lanţ simplu** = lanţul care conţine numai **muchii distincte**

**Lanţ compus**= lanţul care **nu este** format numai din muchii distincte

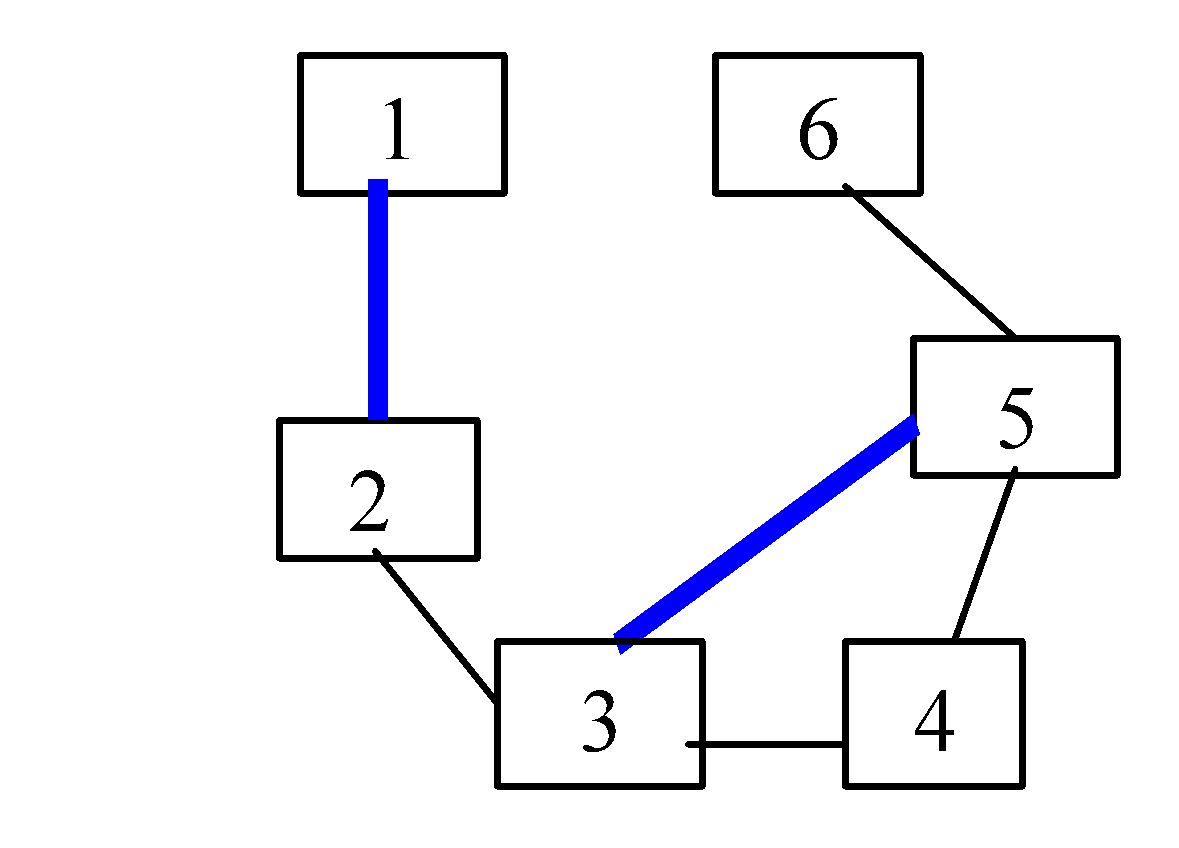
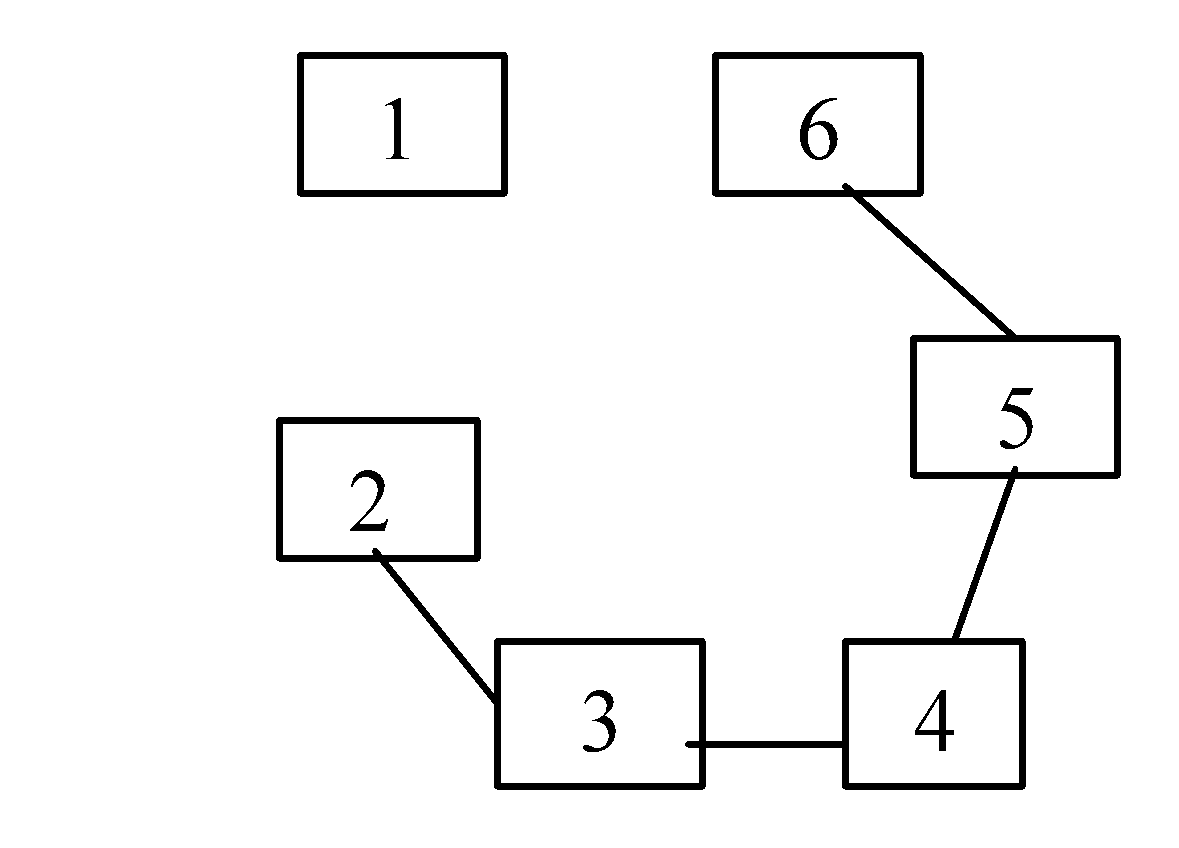
**Lanţ elementar** = lanţul care conţine numai **noduri distincte**

**Ciclu** = Un lanţ în care primul nod coincide cu ultimul.

**Ciclul este elementar** dacă este format doar din noduri distincte, excepţie făcând primul şi ultimul. Lungimea unui ciclu nu poate fi 2.

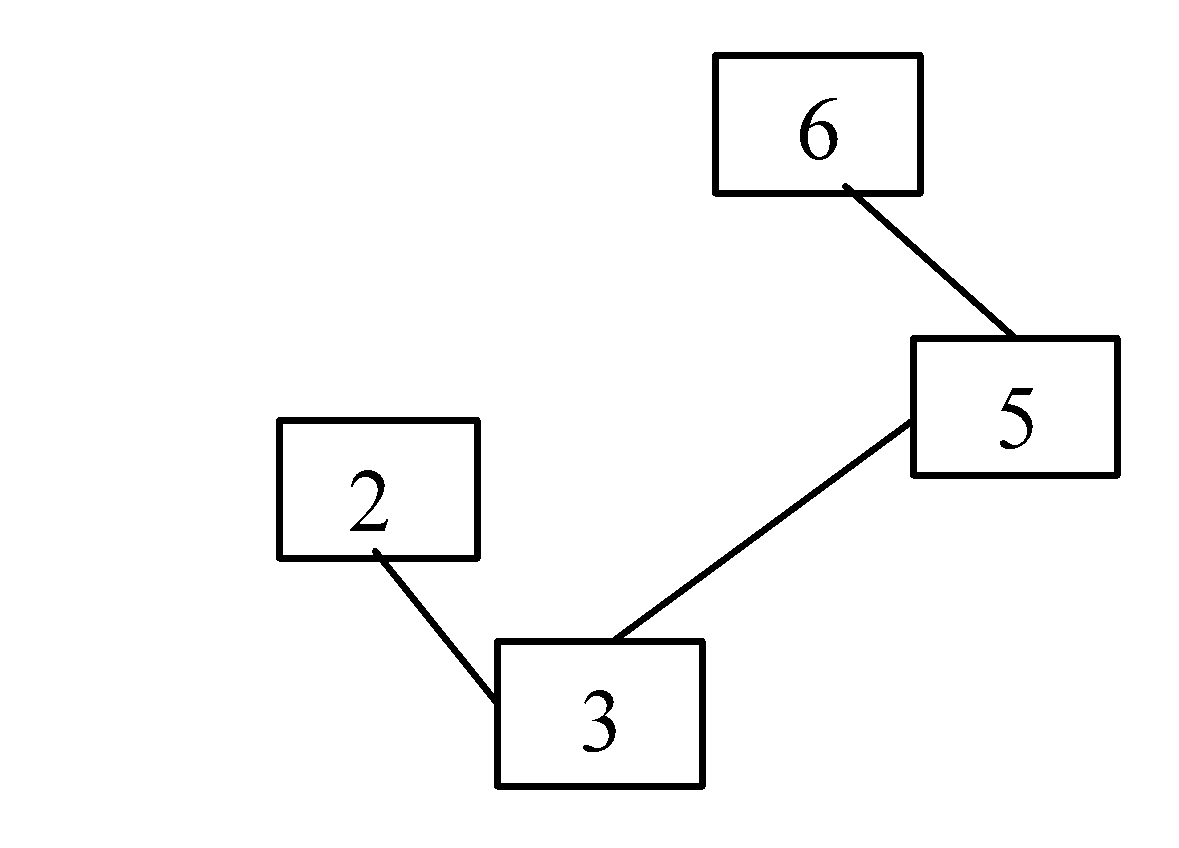
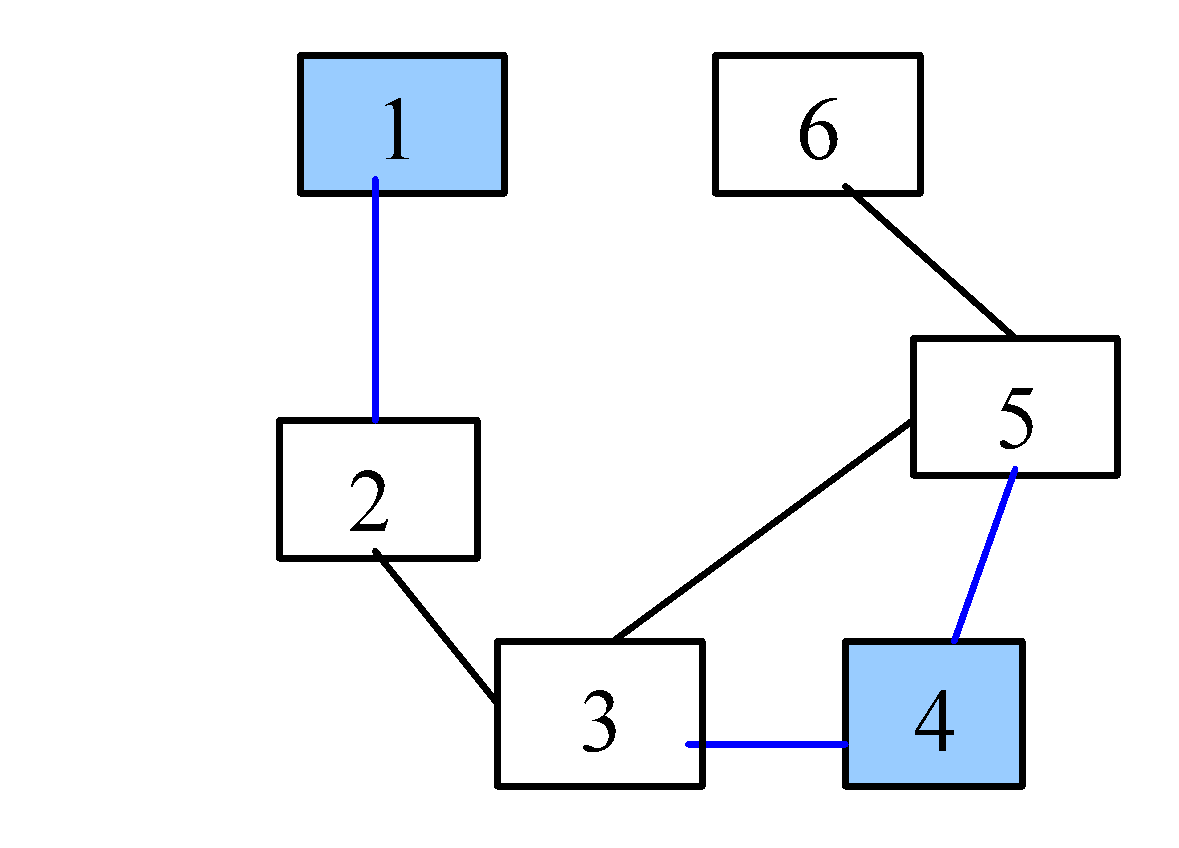
|  | Succesiunea de vârfuri **2, 3, 5, 6** reprezintă un lanţ simplu şi elementar de lungime **3**.  Lanţul **5 3 4 5 6** este simplu dar nu este elementar.  Lanţul **5 3 4 5 3 2** este compus şi nu este elementar.  Lanţul **3 4 5 3** reprezintă un ciclu elementar |
| --- | --- |

**Graf partial** = Dacă dintr-un graf **G=(V,E)** se suprimă cel puţin o muchie atunci noul graf **G’=(V,E’)**, **E’⊂ E** se numeşte graf parţial al lui **G** (are aceleasi noduri si o parte din muchii).



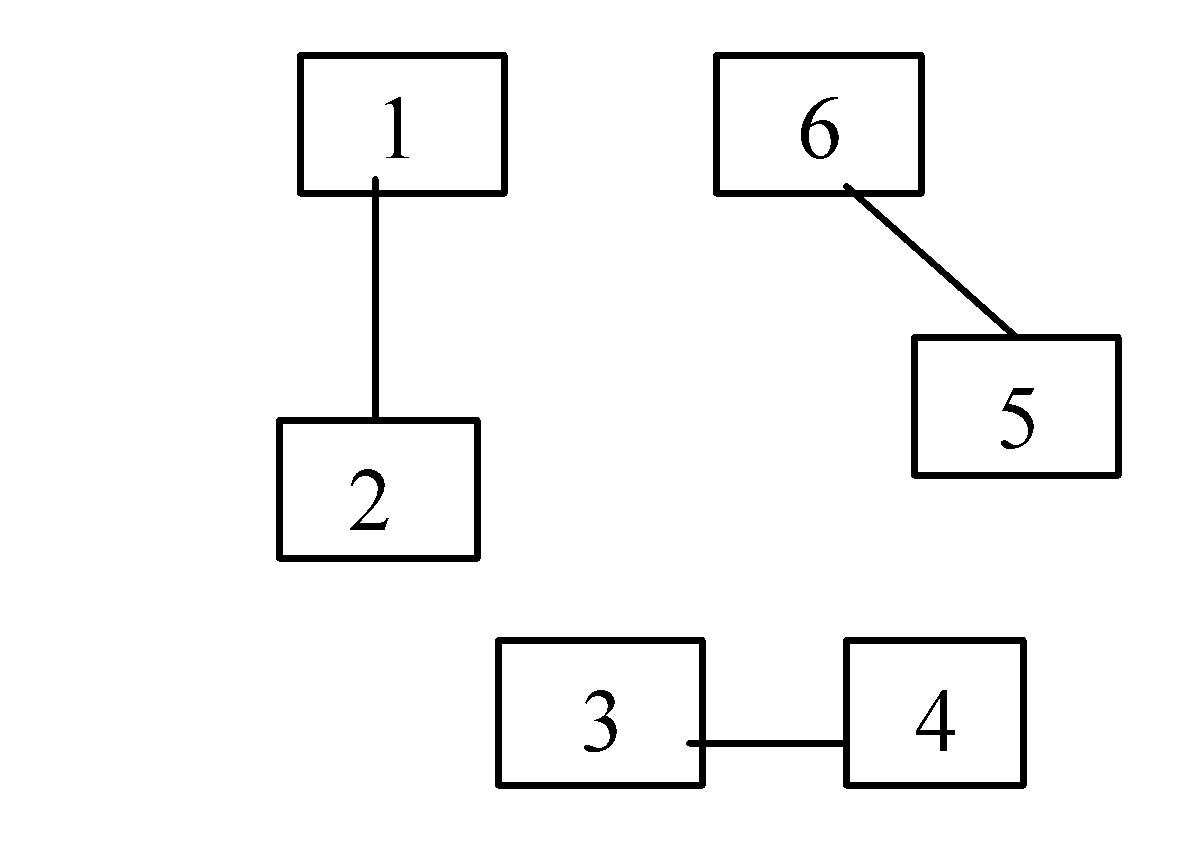
**G G1** este graf partial al lui G

**Subgraf =** Dacă dintr-un graf **G=(V,E)** se suprimă cel puţin un nod împreună cu muchiile incidente lui, atunci noul graf **G’=(V’,E’)**, **E’⊂ E si**  **V’⊂V** se numeşte subgraf al lui **G**.



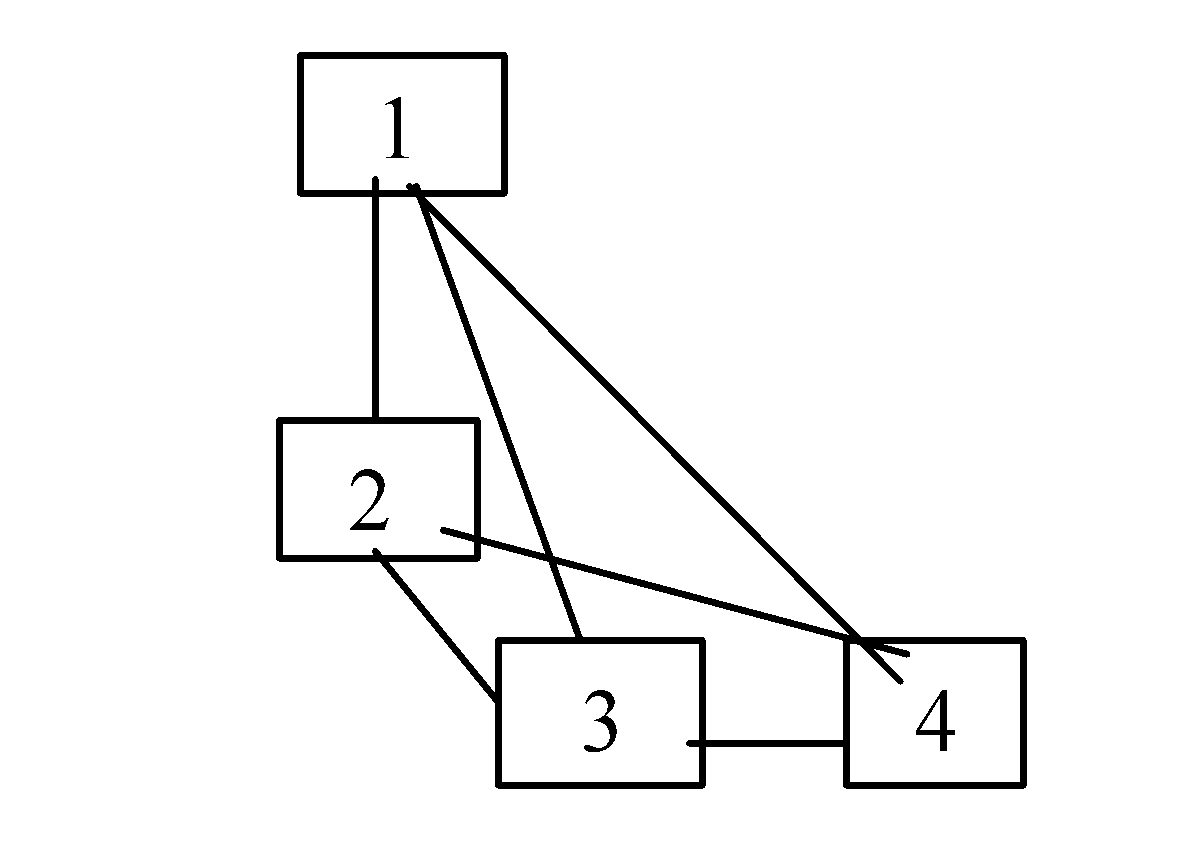
**G G1** este subgraf al lui G

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

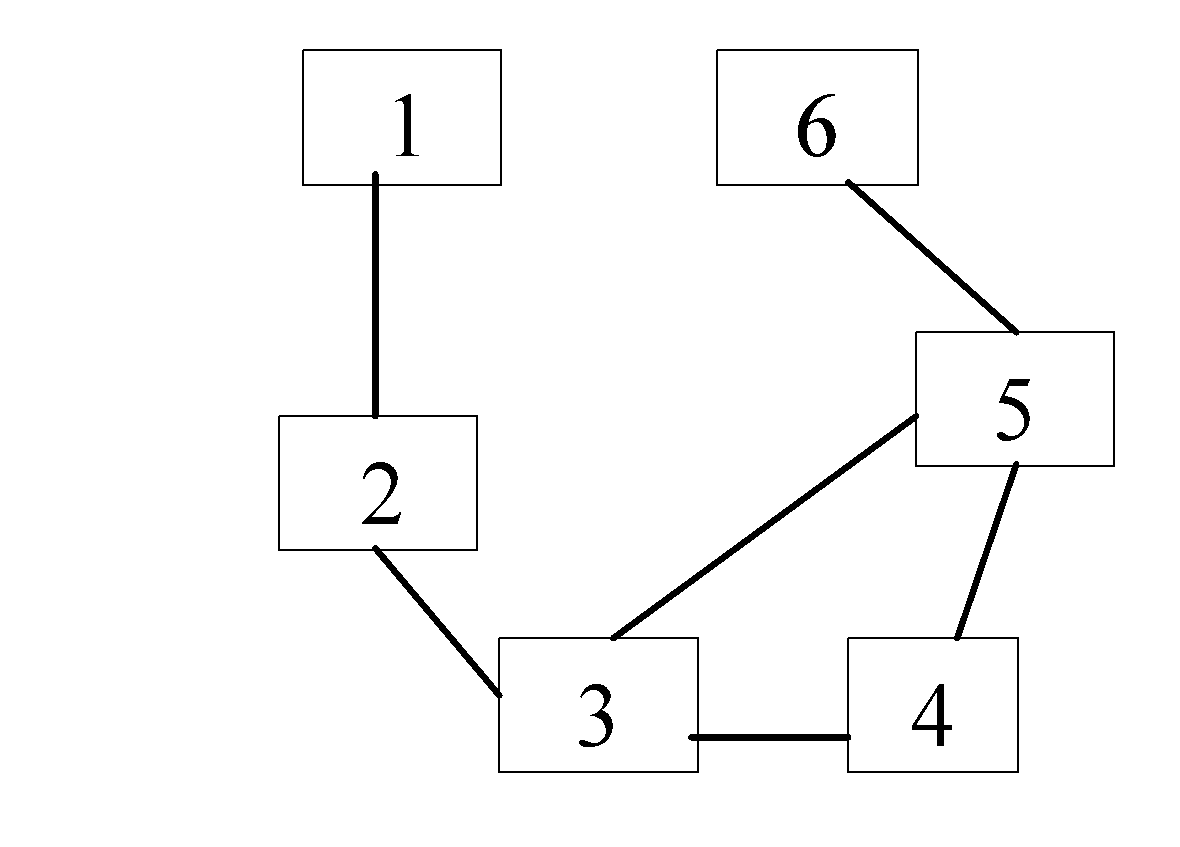
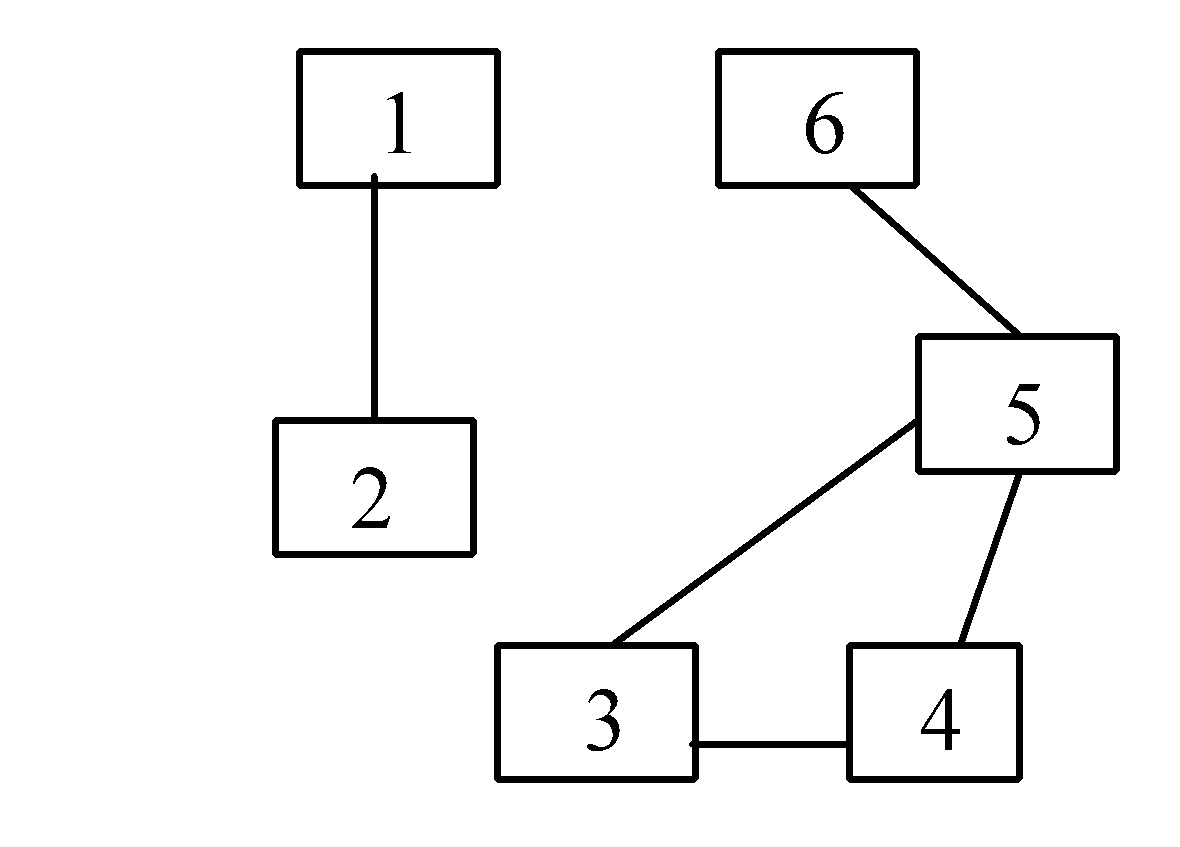
****

**Graf complet** = graf neorientat **G=(V,E)** în care există muchie între oricare două noduri.

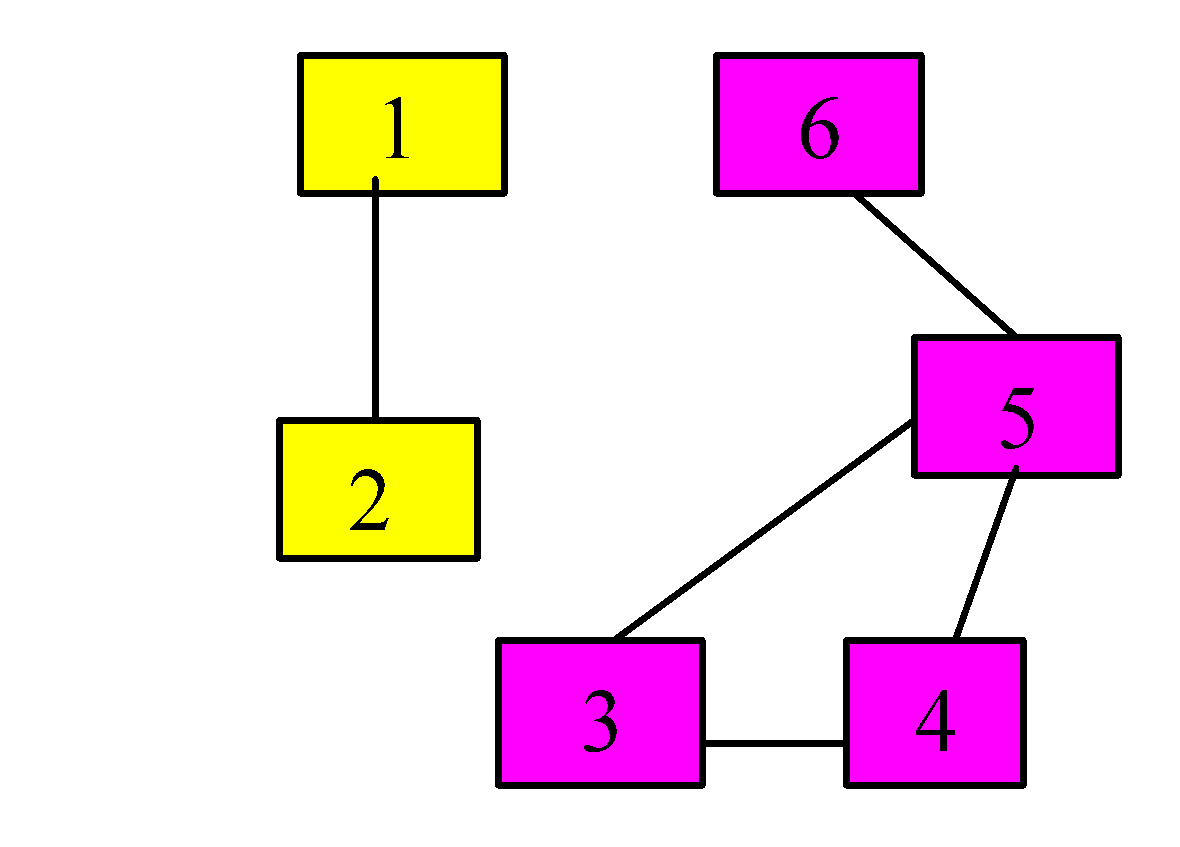
Numărul de muchii ale unui graf complet este: **nr\*(nr-1)/2**.Unde nr este numarul de noduri

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

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

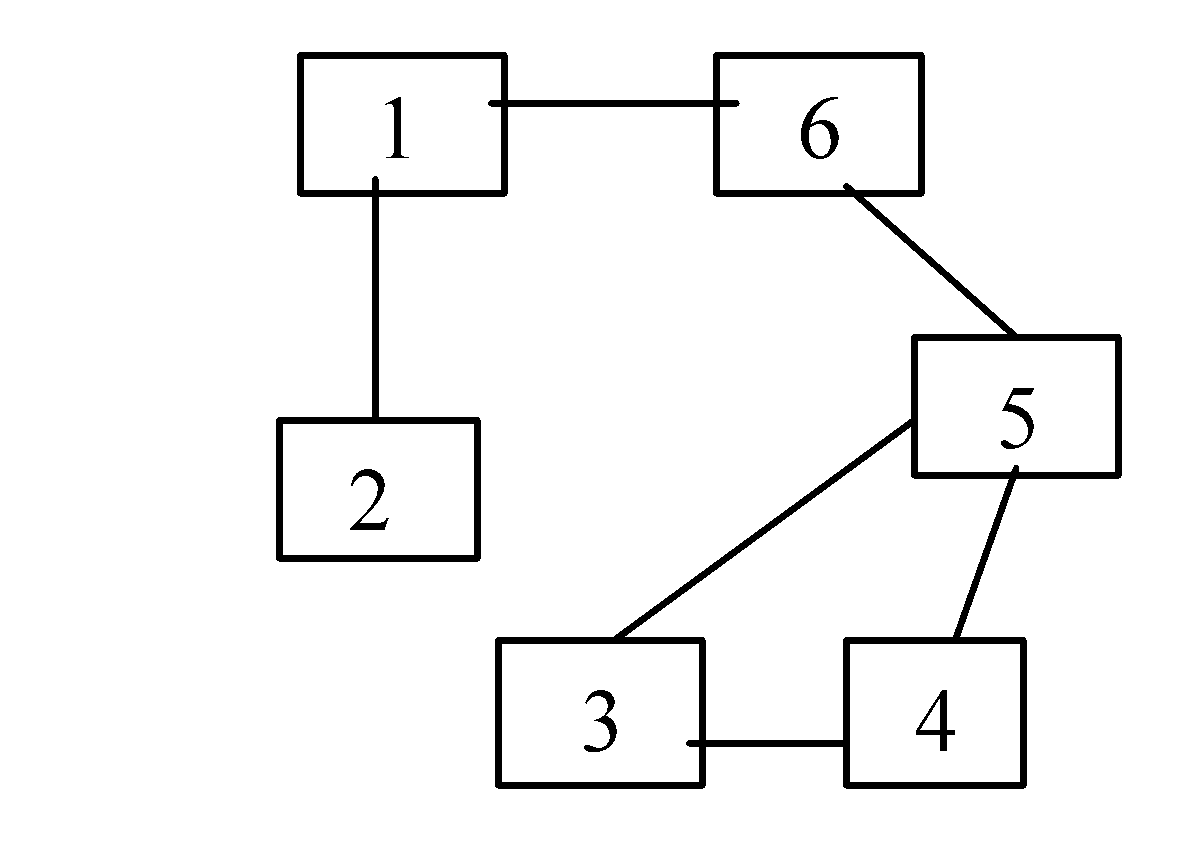
**graf conexnu este graf conex**

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

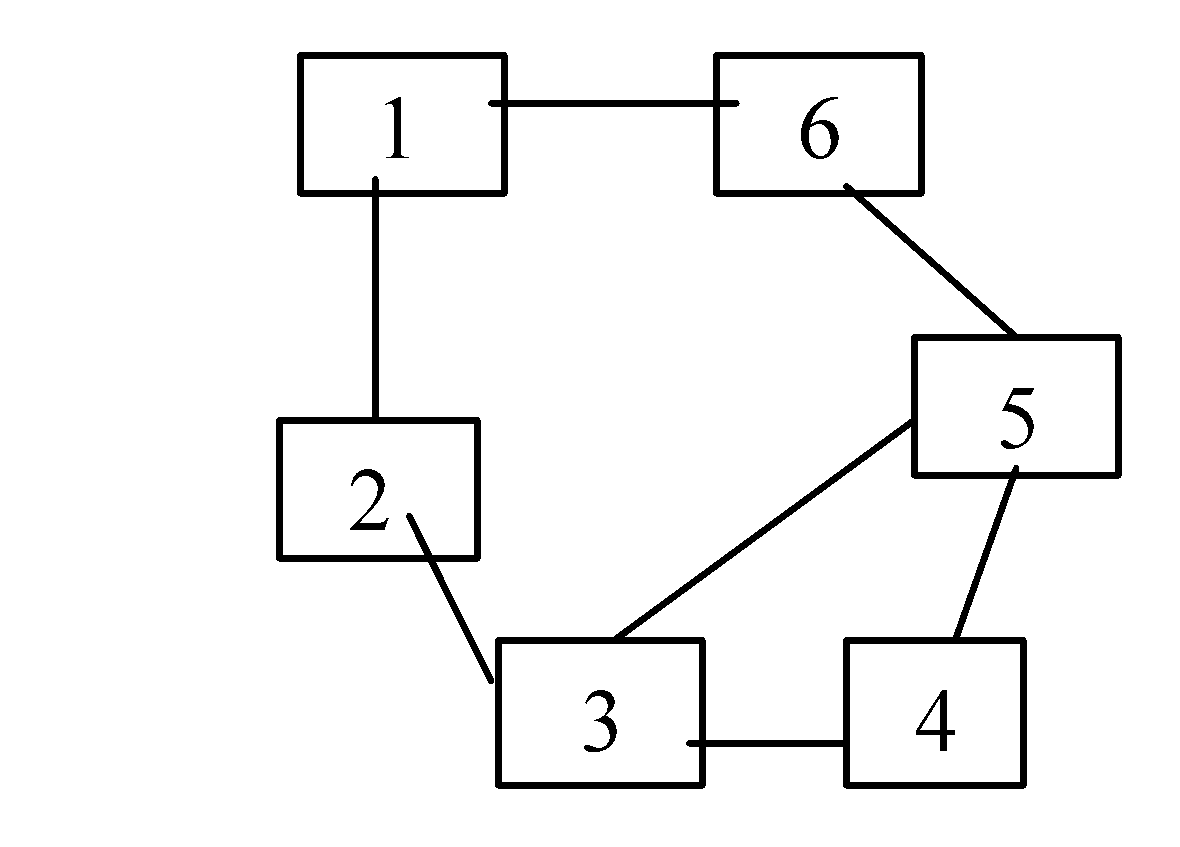
**graful nu este conex.** Are 2 componente conexe:

1, 2 si 3, 4, 5, 6

**Lanţ hamiltonian** = un lanţ elementar care conţine toate nodurile unui graf

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

**Ciclu hamiltonian** = un ciclu elementar care conţine toate nodurile grafului

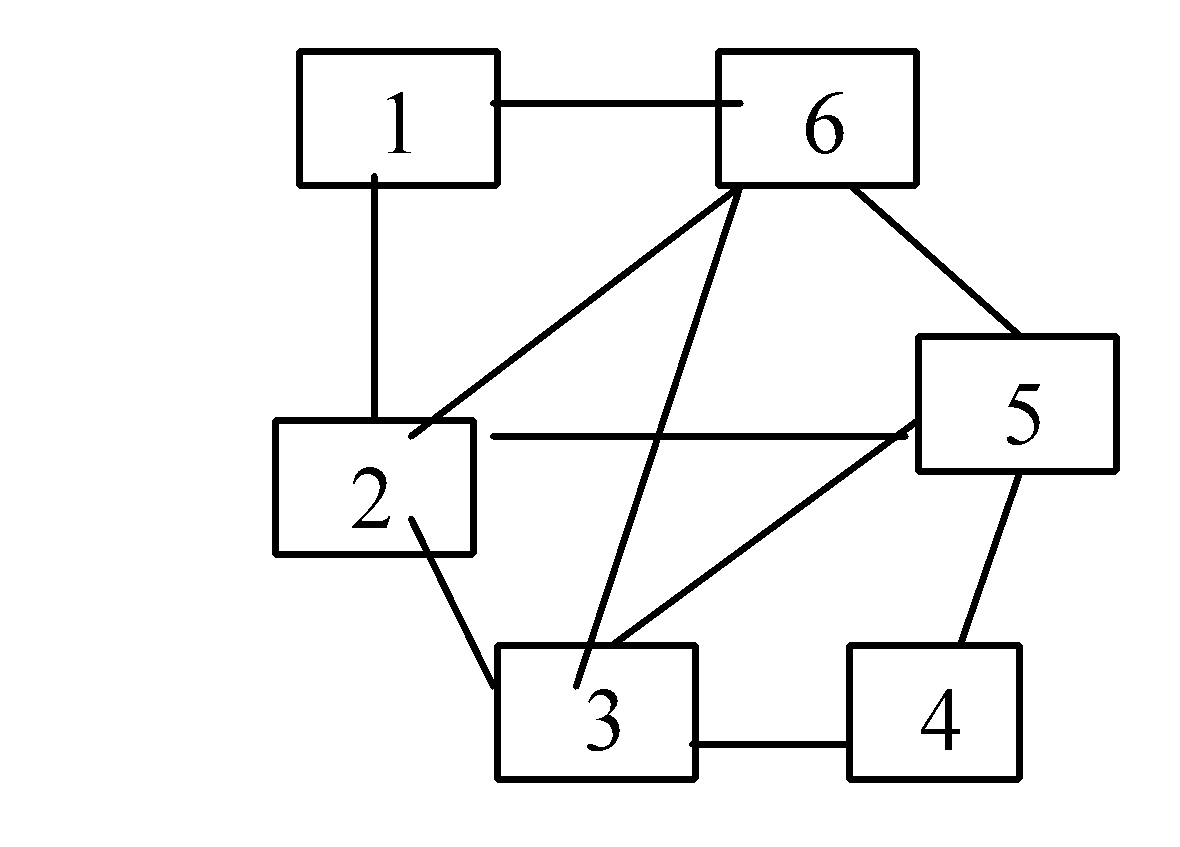
**C=[1,2,3,4,5,6,1] este ciclu hamiltonian**

**Graf hamiltonian** = un graf G care conţine un ciclu hamiltonian

Graful anterior este graf Hamiltonian.

Daca G este un graf cu n>=3 noduri astfel incat gradul fiecarui nod este mai mare sau egal decat n/2 atunci G este hamiltonian

**Lanţ eulerian** = un lanţ simplu care conţine 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 conţine toate muchiile grafului

Ciclul: C=[1.2.3.4.5.3.6.2.5.6.1] este ciclu eulerian

**Graf eulerian** = un graf care conţine un ciclu eulerian.

*Condiţie necesară şi suficientă*: Un graf este eulerian dacă şi numai dacă oricare vârf al său are gradul par.

**2 -** **Reprezentarea grafurilor neorientate**

**Matricea de adiacenta**

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 adiacentã matricea booleanã**

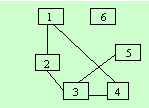
**Matricea de adiacentã** asociatã unui graf neorientat  cu n noduri se defineste astfel: A = (ai j) n x n cu

*Observatii:*

-         Matricea de adiacenta asociatã unui graf neorientat este o matrice simetricã

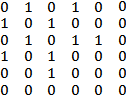
-         Suma elementelor de pe linia k reprezintã gradul nodului k

-         Suma elementelor de pe coloana k reprezintã gradul  nodului k



Fie graful din figura urmatoare:

Matricea de adiacenta:

nodul 1 are gradul 2

nodul 2 are gradul 2

nodul 3 are gradul 3

nodul 4 are gradul 2

nodul 5 are gradul 1

nodul 6 are gradul 0

Numarul de noduri este 6 si numarul de muchii este 5

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

5- reprezentand m, numarul de muchii

5 perechi x-y reprezentand extremitatile celor 5 muchii:

1-2 => a[1,2]=a[2,1]=1

1-4 => a[1,4]=a[4,1]=1

2-3 => a[2,3]=a[3,2]=1

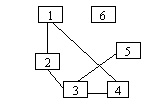
3-4=>  a[3,4]=a[4,3]=13-5 => a[3,5]=a[5,3]=1

**3-** **Reprezentarea grafurilor neorientate**

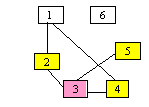
**Liste de adiacenta**

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 graful din figura urmatoare:

****

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

****

Nodul 6 nu are vecini (este izolat)

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 :

 Fie 𝐺 = (𝑉, 𝐸) un graf neorientat sau orientat cu 𝑛 vârfuri. Pentru a reprezenta graful prin liste de adiacență, vom reține pentru fiecare vârf 𝑥 al grafului toate vârfurile 𝑦 cu proprietatea că există muchia [𝑥, 𝑦] (pentru graf neorientat), respectiv există arcul (𝑥, 𝑦) (pentru graf orientat), formând 𝑛 liste de adiacență. Ordinea în care sunt memorate vârfurile într-o listă de adiacență nu contează.

**Implementare:**Cu vectori “clasici” – fiecare listă de adiacență este reprezentată ca un vector cu maxim 𝑛 componente în care vârfurile sunt memorate pe poziții consecutive.

Cu liste înlănțuite – fiecare listă de adiacență este reprezentată ca o listă înlănțuită; se reține pentru fiecare element al listei adresa spre următorul element precum și informația utilă.

Cu vectori din STL – fiecare listă de adiacență este reprezentată ca un vector din STL (𝑣𝑒𝑐𝑡𝑜𝑟).

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.

**4-** **Parcurgerea grafurilor**

1. **Parcurgerea in latime**

Rezolvarea multor probleme de grafuri, presupune parcurgerea lor de la un anumit nod. Pentru explorarea grafurilor, exista doua tipuri de algoritmi: de explorarea in latime si de explorare in adancime.

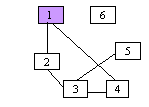
**Explorarea grafurilor in latime :**

La explorarea in latime, dupa vizitarea nodului initial, se exploreaza toate nodurile adiacente lui, se trece apoi la primul nod adiacent si se exploreaza toate nodurile adiacente acestuia si neparcurse inca, s.a.m.d.

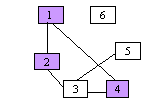
Fiecare nod se parcurge cel mult odata (daca graful nu este conex nu se vor putea parcurge toate nodurile)

De exemplul pentru garful din figura de mai jos, se va proceda in felul urmator:

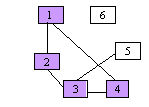
 se porneste din nodul 1, (se poate incepe de la oricare alt nod)

****

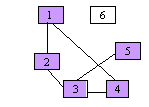
se exploreaza in continuare vecinii acestuia : nodul 2 si apoi 4,

** se obtine 1,2,4**

 dupa care din 2 se exploreaza nodul adiacent acestuia 3. Nodul 1 nu se mai viziteaza odata

**se obtine 1,2,4,3**

In  continuare ar trebui parcursi vecinii lui 4**(1,2,4,3**)  dar acesta nu mai are vecini nevizitati si se trece la vecinii lui 3**: 1,2,4,3**respectiv nodul 5 :

**se obtine 1, 2, 4, 3, 5**

Nodul 6 ramane neparcurs

**Algoritmul**

Se va folosi o coada  in care se inscriu nodurile in forma in care sunt parcurse: nodul initial varf (de la care se porneste), apoi nodurile a,b,..., adiacente lui varf, apoi cele adiacente lui a, cele adiacente lui b,... ,s.a.m.d.

Coada este folosita astfel:

- se pune primul nod in coada;

- se afla toate varfurile adiacente cu primul nod si se introduc dupa primul nod

- se ia urmatorul nod si i se afla nodurile adiacente

- procesul se repeta pana cand se ajunge la sfarsitul cozii

-Graful se va memora utilizand matricea de adiacenta a[10][10]

-pentru memorarea succesiunii nodurilor parcurse se va folosi un vector c[20] care va functiona ca o coada

-pentru a nu parcurge un nod de doua ori se va folosi un vector boolean viz[20] care va retine :

                                                - viz[k]=0 daca nodul k nu a fost vizitat inca

                                                - viz[k]=1 daca nodul k a fost vizitat

-doua variabile : prim si ultim vor retine doua pozitii din vectorul c si anume :

                                                - prim este indicele componentei pentru care se parcurg vecinii (indexul componentelor marcate cu rosu in sirurile parcurse anterior ). Prin urmare Varf=c[prim], este elementul pentru care se determina vecinii (nodurile adiacente)

                                                -ultim este pozitia in vector pe care se va face o noua inserare in vectorul c (evident, de fiecare data cand se realizeaza o noua inserare se mareste vectorul)

-vecinii nodului varf se « cauta » pe linia acestui varf : daca a[varf][k]=1 inseamna ca nodurile varf si k sunt adiacente. Pentru ca nodul k sa fie adaugat in coada trebuie ca nodul sa nu fi fost vizitat : viz[k]=0

**Observații:** Parcurgerea în lățime are o proprietate remarcabilă: fiecare vârf este vizitat pe cel mai scurt drum / lanț începând din vârful de start.

Complexitatea parcurgerii în lățime (BFS) în cazul reprezentării prin liste de adiacență este (𝑛 + 𝑚) (în cazul reprezentării prin matrice de adiacență complexitatea este 𝑂(𝑛 2 )).

1. **Parcurgerea in adancime**

Parcurgerea unui graf in adancime se face prin utilizarea stivei (alocate implicit prin subprograme recursive).

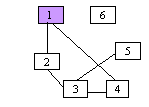
Pentru fiecare nod se parcurge primul dintre vecinii lui neparcursi inca

  Dupa vizitarea nodului initial x1, se exploreaza primul nod adiacent lui fie acesta x2 , se trece apoi la primul nod adiacent cu x2si care nu a fost parcurs inca , s.a.m.d.

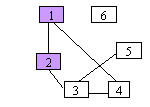
Fiecare nod se parcurge cel mult odata (daca graful nu este conex nu se vor putea parcurge toate nodurile)

De exemplul pentru garful din figura de mai jos, se va proceda in felul urmator:

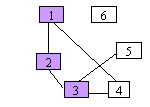
 se porneste din nodul 1, (se poate incepe de la oricare alt nod)

****

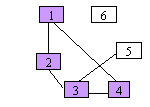
se exploreaza in vontinuare primul vecin al acestuia acestuia : nodul  2,

** se obtine 1,2**

 dupa care din 2 se exploreaza un nod adiacent cu acesta si care nu a fost vizitat : 3.( Nodul 1 nu se mai viziteaza odata)

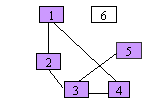
**se obtine 1,2,3**

In  continuare ar trebui sa se parcurga vecinul lui 3 nevizitat : 4

**se obtine 1, 2, 3, 4**

 Pentru nodul 4 ar trebui sa se parcurga primul sau vecin neparcurs (nodul 1 dar acesta a fost deja parcurs. Si nodul 3 a fost parcurs. Din 4 nu mai avem ce vizita si se trece la  nivelul anterior din stiva, la nodul 3 :

**1, 2, 3, 4**Se parcurge vecinul sau nevizitat, nodul 5 .

****SE obtine : **1, 2, 3, 4 , 5**.

Nodul 3 nu mai are vecini nevizitati si se trece pe nivelul anterior din stiva, nodul 2 : **1, 2, 3, 4 , 5.**Nici acesta nu mai are vecini nevizitati si se trece pe nivelul anterior la nodul 1**: 1, 2, 3, 4 , 5**. Cum nici acesta nu mai are vecini nevizitati se incheie algoritmul.

Nodul 6 ramane nevizitat.

**Algoritmul**

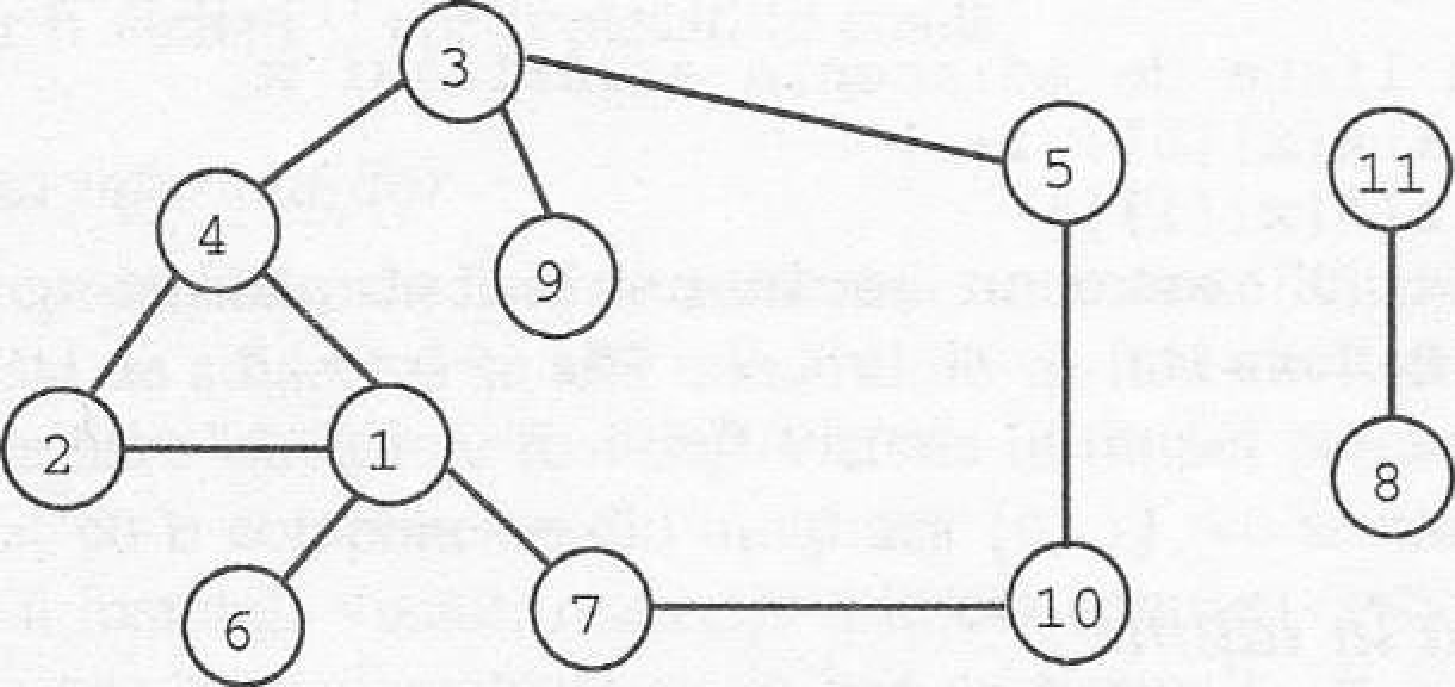
-Graful se va memora utilizand matricea de adiacenta a[10][10]

-pentru a nu parcurge un nod de doua ori se va folosi un vector boolean vizcare va retine :

                                                - viz[k]=0 daca nodul k nu a fost vizitat inca

                                                - viz[k]=1 daca nodul k a fost vizitat

-ca si la parcurgerea in latime vecinii unui nod se « cauta » pe linia acestui nod : daca a[nod][k]=1 inseamna ca nodurile nod si k sunt adiacente. Pentru ca nodul k sa fie fie parcurs trebuie ca nodul sa nu fi fost vizitat : viz[k]=0

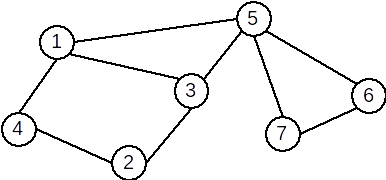
****

Parcurgând graful din figură în adâncime considerând drept vârf de start vârful 3 putem obține următoarea ordinea de vizitare a vârfurilor accesibile din nodul de start: 3,4,1,2,6,7,10,5,9.

(Pentru această succesiune, ordinea de vizitare a vecinilor unui vârf este ordinea crescătoare a numerelor lor).

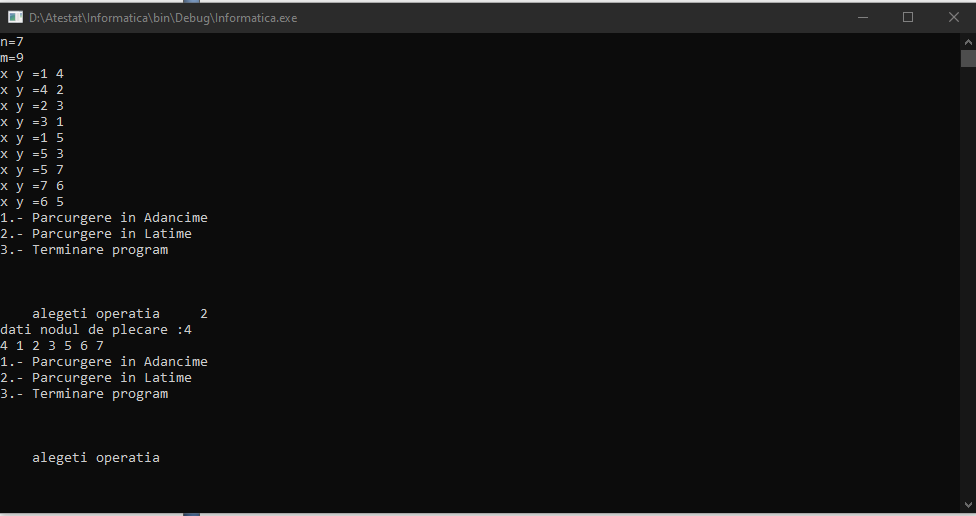
**Aplicație**

Se da un graf neorientat și un vârf oarecare. Să se afișeze succesiunea vârfurilor conform parcurgerii în lățime și în adâncime:

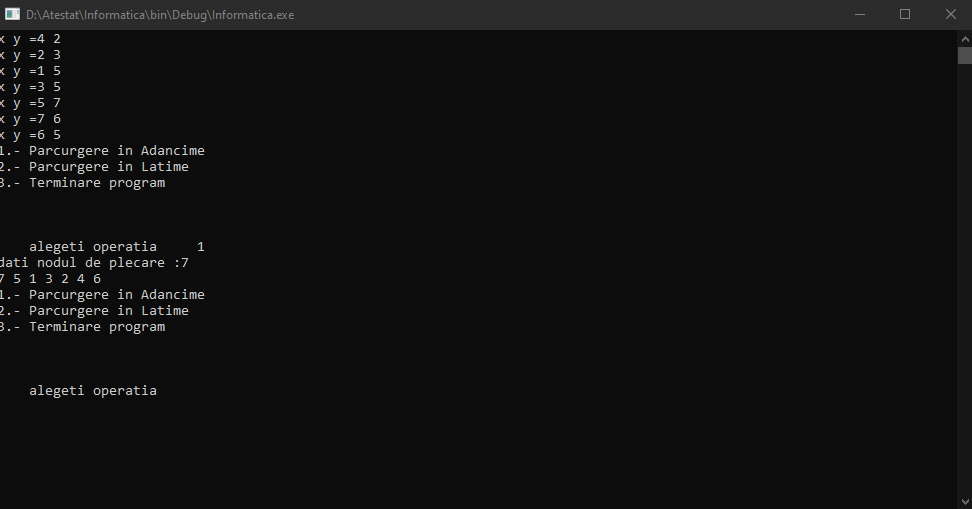


**Rezolvare:**

**In latime:**



**In adancime:**



**Listungul programului:**

#include <iostream>

using namespace std;

int a[20][20],coada[20], viz[20];

int i, n, el, j, p, u, pl, m, x, y,opt;

void citire(int&n,int&m)

{

cout<<"n="; cin>>n;

cout<<"m="; cin>>m;

for (i=1;i<=m;i++)

{cout<<"x y ="; cin>>x>>y;

a[x][y]=1; a[y][x]=1;}

}

void latime()

{

for (i=1;i<=n;i++)

viz[i]=0;

cout<<"dati nodul de plecare :"; cin>>pl;

viz[pl]=1;

coada[1]=pl;

p=1; u=1;

while (p<=u)

{ el=coada[p];

for (j=1;j<=n;j++)

if ((a[el][j]==1) && (viz[j]==0))

{u=u+ 1;

coada[u]=j;

viz[j]=1;}

p=p+1;}

for (i=1;i<=u;i++) cout<<coada[i]<<" ";

}

void adancime(int pl)

{int j;

cout<<pl<<" ";

viz[pl]=1;

for (j=1; j<=n;j++)

if ((a[pl][j]==1) && (viz[j]==0))

adancime(j);

}

int main()

{

citire(n,m);

while(1){ //clrscr();

cout<<"1.- Parcurgere in Adancime "<<"\n";

cout<<"2.- Parcurgere in Latime "<<"\n";

cout<<"3.- Terminare program "<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<" alegeti operatia ";

cin>>opt;

if(opt==1) {for (i=1;i<=n;i++)

viz[i]=0;

cout<<"dati nodul de plecare :"; cin>>pl;

adancime(pl);

cout<<endl;}

if(opt==2) {for (i=1;i<=n;i++)

viz[i]=0;

latime();

cout<<endl;}

if(opt==3)

break;

}

return 0;

}

**Bibliografie**

* ,,Initiere in Programarea Vizuala’’ ( Varianta Borland C++ Builder) ; Tudor Sorin ; Editura L&S ;
* ,,Programare in C++ Builder’’ ; Mihai Olteanu si Crina Grojan ; Editura Albastra
* “Totul despre C si C++”-Dr.Kris Jamsa & Lars Klander
* “Manual de informatica cls a XI-a,profil real,editura Didactica si Pedagogica”-Mariana Milosescu
* “Culegere de C/C++”-Stoilescu Dorian