GRAFURI NEORIENTATE

1. Noţiunea de graf neorientat

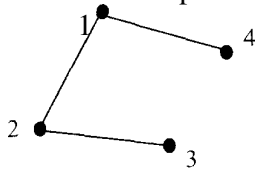
Definiţie. Se numeşte **graf neorientat** o pereche ordonată de mulţimi notată G=(V, M) unde:
V : este o mulţime finită şi nevidă, ale cărei elemente se numesc noduri sau vârfuri;
M : este o mulţime, de perechi neordonate de elemente distincte din V, ale cărei elemente se numesc muchii.
• **Exemplu** de graf neorientat:
G=(V, M) unde: V={1,2,3,4}
M={{1,2}, {2,3},{1,4}t}
Demonstraţie:
Perechea G este graf neorientat deoarece respectă definiţia prezentată mai sus, adică:
V : este finită şi nevidă;
M : este o mulţime de perechi neordonate (submulţimi cu două elemente) de elemente din V.

În continuare, vom nota submulţimea {x,y}, care reprezintă **o muchie**, cu [x,y] (într-un graf neorientat muchia [x,y] este aceeaşi cu muchia [y,x]). În baza celor spuse anterior, graful prezentat în exemplul de mai sus se reprezintă textual astfel:
G=(V, M) unde: V={1,2,3,4}
M={[1,2],[2,3],[1,4]}

În teoria grafurilor neorientate, se întâlnesc frecvent noţiunile:
- **extremităţile unei muchii**
  • fiind data muchia m=[x,y], se numesc extremităţi ale sale **nodurile** x şi y;
- **vârfuri adiacente**
  • dacă într-un graf există muchia m=[x,y], se spune despre **nodurile** x şi y ca sunt **adiacente;**
- **incidenţă**
  • dacă ml şi m2 sunt două muchii ale aceluiaşi graf, se numesc **incidente** dacă **au o extremitate comună**.
  Exemplu: m 1=[x,y] şi m2=[y,z] sunt incidente
  • dacă m=[x,y] este o muchie într-un graf, se spune despre ca şi **nodul x**, sau **nodul y**, ca sunt incidente.

Reprezentarea unui graf neorientat admite două forme, şi anume:
- reprezentare textuală: aşa cum s-a reprezentat graful din exemplul anterior;
- reprezentare grafică : **muchiile** sunt reprezentate prin **linii**,
  iar **nodurile** prin **puncte.**

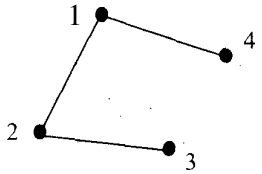
• **Exemplu** de graf neorientat reprezentat textual:
G=(V, M) unde: V={ 1,2,3,4]
M={[l,2], [2,3], [1,4]}
• **Exemplu** de graf neorientat reprezentat grafic (este graful de la exemplul anterior):



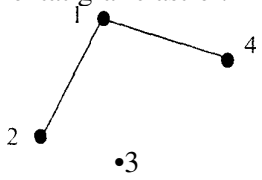
2. Noţiunea de graf parţial

Definiţie. Fie G=(V, M) un graf neorientat. Se numeşte **graf parţial**, al grafului G, graful neorientat $G_1=(V, M_1)$ unde $M_1 \subseteq M$.
Concluzie:
Un graf parţial al unui graf neorientat G=(V, M) are aceeaşi mulţime de vârfuri ca şi G iar mulţimea muchiilor este o submulţime a lui M sau chiar M.
• **Exemplu**: Fie graful neorientat:

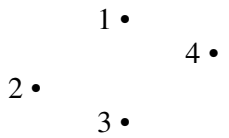
G=(V, M) unde: V={ 1,2,3,4}
M={[1,2], [1,4], [2,3]}
reprezentat grafic astfel:



1. Un exemplu de graf parțial al grafului G este graful neorientat:
G_1=(V, M,) unde: V={1,2,3,4}
M_1={[1,2],[1,4]} (s-a eliminat muchia [2,3])
reprezentat grafic astfel:



2. Un exemplu de graf parțial al grafului G este graful neorientat:
G_1=(V,M_1) unde: V={1,2,3,4}
M_1= ϕ (s-au eliminat toate muchiile)
reprezentat grafic astfel:

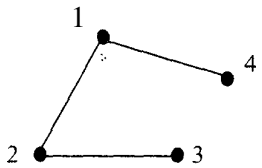


Observație. Fie G=(V, M) un graf neorientat. Un graf parțial, al grafului G, se obține păstrând vârfurile şi eliminând eventual nişte muchii (se pot elimina şi toate muchiile, sau chiar nici una).

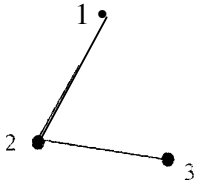
3. Noțiunea de subgraf

Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **subgraf** al grafului G, graful neorientat G_1=(V_1, M_1) unde $V_1 \subseteq V$ iar M, conține toate muchiile din M care au extremitățile în V_1.

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



1. Un exemplu de subgraf al grafului G este graful neorientat:
G_1=(V_1 , M_1) unde: V_1={1,2,3 } (s-a şters nodul 4) reprezentat grafic astfel:
M_1=([1,2],[2,3]} (s-a eliminat muchia [1,4J)



2. Un exemplu de subgraf al grafului G este graful neorientat:
G_1=(V_1 ,M_1) unde: V_1={1,2,3,4}
(s-a eliminat nodul 1)
M_1={[2,3]}(s-au eliminat muchiile [1,4], [1,2])

reprezentat grafic astfel:

Observaţie. Fie G=(V, VI) un graf neorientat. Un subgraf, al grafului G, se obţine ştergând anumite vârfuri şi odată cu acestea şi muchiile care le admit ca extremitate.

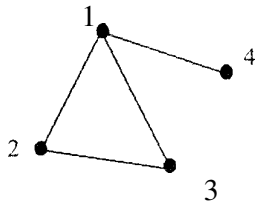
4. Gradul unui vârf

Definiţie. Fie G=(V, M) un graf neorientat şi x un nod al său. Se numeşte **grad al nodului x**, numărul muchiilor incidente cu x, notat **d(x).**

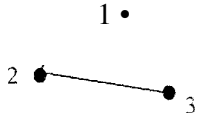
•**Exemplu**: Fie graful neorientat: reprezentat grafic astfel:

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

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



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)
Observaţii.
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.**
În graful care admite reprezentarea grafică:



deoarece **d(1)=0**, vârful **1** se numeşte **vârf izolat**, şi
deoarece **d(2)=d(3)=1**, vârfurile **2** şi **3** se numesc **vârfuri terminale.**

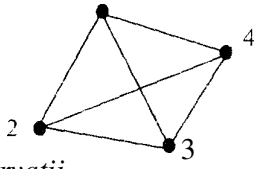
Propoziţie. În graful neorientat G=(V, M), in care V={x_1, x_2, ..., x_n} şi sunt m muchii, se verifică egalitatea : $\sum_{i=1}^{n} d(x_i) = 2m$

Demonstraţie:
Muchia [x,y] contribuie cu o unitate la gradul lui x şi cu o unitate la gradul lui y, deci, cu două unităţi la suma din enunţ. Cum în total sunt m muchii, rezultă că suma gradelor este 2m.

Având in vedere faptul ca suma gradelor vârfurilor dintr-un graf este un număr par (2m), a apărut corolarul prezentat mai jos.
Corolar. În orice graf neorientat, G=(V, M), există un număr par de vârfuri de grad impar.
Demonstratie:
Demonstraţia ţine cont de propoziţia de mai sus, adică de faptul că într-un graf neorientat suma S a tuturor gradelor nodurilor este un număr par (dublul numărului muchiilor).
Fie S1 suma gradelor vârfurilor de grad par, (este un număr par, ca sumă de numere pare) şi S2 suma gradelor vârfurilor de grad impar.
Cum S=S1+S2, rezultă că S2=S-S1, deci un număr par (ca diferenţă de numere pare).

5. Graf complet

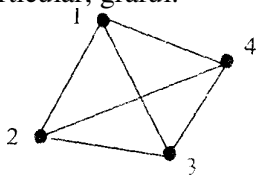
Definiţie. Fie G=(V, M) un graf neorientat. Graful G se numeşte **graf complet**, dacă **oricare două vârfuri distincte ale sale sunt adiacente.**
• **Exemplu** de graf neorientat complet:
G=(V, M) unde: V={ 1,2,3,4}
M={[1,2], [1,3], [l,4], [2,3], [2,4], [3,4]}
Reprezentarea sa grafică este:
1



Observaţii.

1. Într-un graf complet cu n vârfuri gradul fiecărui vârf este n-1, deoarece fiecare vârf este legat prin muchii de toate celelalte vârfuri.
2. Graful complet cu n vârfuri se notează cu K_n

În particular, graful:



se notează K_4 (este un graf complet cu 4 vârfuri).

Propoziţie. Într-un graf complet cu n vârfuri, notat K_n, există $\frac{n(n-1)}{2}$ muchii.

Demonstraţie:

Din fiecare vârf x, pleacă n-1 muchii, deci $d(x_i)=n-1$, pentru orice i= 1..n Cum folosind propoziţia prezentată în secţiunea gradul unui vârf.

$2m= d(x_1)+ d(x_2)+ ... +d(x_n) \rightarrow 2m=(n-1)+ (n-1)+... +(n-1)$

$\rightarrow 2m=n(n-1)/2$

6. Graf bipartit

Definiţie. Fie G =(V, M) un graf neorientat. Gratul G se numeşte **graf bipartit**, dacă există două mulţimi nevide Vl şi V2 cu proprietăţile:

$-V_1 \cup V_2 = V$

$-V_1 \cap V_2 = \phi$

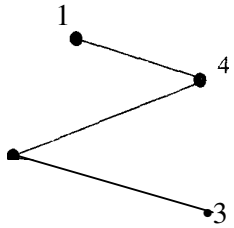
- orice muchie a lui C are o extremitate în V1 şi pe cealaltă în V2.

• Exemplu de graf neorientat bipartit:

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

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

Reprezentarea sa grafică este:



Demonstraţie:

Graful de mai sus este bipartit deoarece respectă întocmai definiţia grafului bipartit, adică există două mulţimi V1={1,2} şi V2={3,4} astfel încât: $-V_1 \cup V_2 = V$ $-V_1 \cap V_2 = \phi$

- orice muchie a lui C are o extremitate în V1 şi pe cealaltă în V2.

Cum, în plus, pentru orice x din V1 şi orice y din V2 există în G muchia [x,y], rezultă, conform definiţiei, ca graful G este bipartit complet.

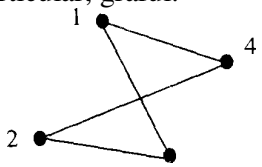
Observaţie. A demonstra că un graf au este bipartit complet înseamnă a demonstra :

- că este bipartit
- pentru orice x din Vl şi orice y din V2 există in G muchia [x,y].

Observaţie. Într-un graf bipartit complet în care V1 are p elemente şi V2 are q elemente există pq muchii.

Demonstraţie:

Fiecare vârf x_i din V1 este legat de toate vârfurile aflate în V2, altfel spus, există q muchii care-l admit ca extremitate pe x_i. Cum în Vl sunt p elemente, adică i= 1...p. înseamnă că elementele mulțimii V1 sunt legate prin pq muchii de elementele mulțimii V2.
Observație. Graful bipartit complet în care V1 are p elemente şi V2 are q elemente se notează cu $K_{p,q}$.
În particular, graful:



se notează $K_{2,2}$ (este un graf bipartit complet cu V1 = 2 şi V2=2 , V1 şi V2 din definiție).

8. Reprezentarea grafurilor neorientate

Fie G=(V, M) un graf neorientat, unde V={x_1, x_2,..., x_n} şi M={m_1 ,m_2,..., m_p }. Deoarece între mulțimea {x_1, x_2,..., x_n} şi mulțimea {1, 2,..., n} există o bijecție, $x_i \leftrightarrow i$, putem presupune, fără a restrânge generalitatea, mai ales pentru a uşura scrierea, ca V={ 1, 2,..., n}.
În baza celor spuse mai sus, mai departe,]n loc de x; vom scrie i şi]n loc de muchia [x_i,x_j] vom scrie [i,j].
Pentru a putea prelucra un graf neorientat cu ajutorul unui program, trebuie mai întâi să fie reprezentat in programul respectiv.
Pentru a reprezenta un graf, într-un program, există mai multe modalități folosind diverse structuri de date; dintre acesta, in continuare, vom prezenta:
- reprezentarea unui graf prin matricea de adiacență;
- reprezentarea unui graf prin listele de adiacență;
- reprezentarea unui graf prin şirul muchiilor.

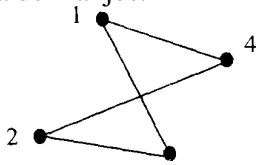
Matricea de adiacentă

Fie G=(V, M) un graf neorieritat cu n vârfuri (V={1,2, ..., n}) şi m muchii.
Matricea de adiacență, asociată grafului G, este o matrice pătratică de ordinul n, cu elementele definite astfel:

$$a_{i,j} = \begin{cases} 1, & daca\ [i,j] \in M \\ 0, & daca\ [i,j] \notin M \end{cases}$$

(altfel spus, $a_{i,j}$=1, dacă există muchie între i şi j şi $a_{i,j}$=0 dacă nu există muchie între i şi j)

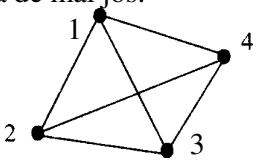
•**Exemplul 1**. Fie graful reprezentat grafic ca in figura de mai jos:



Matricea de adiacență, asociată grafului, este:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

• **Exemplul 2**. Fie graful reprezentat grafic ca in figura de mai jos:



Matricea de adiacentă, asociată grafului, este:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Comentarii:
1. **Matricea de adiacență** este o matrice **pătratică**, de ordin n, şi **simetrică** față de diagonala principală (adică **a[i][j]=a[j][i]).**
Secvențele de citire a matricei de adiacență, sunt:

```
int a[100][100];
.........
cout<<"n="; cin>>n;
for (i=1;i<=n-l;i++)    se citesc valorile elementelor
  for (j=i+l;j<=n;j++)  de deasupra diagonalei principale
    { cin>>a[i][j];     şi se transferă şi sub
      a[j][i]=a[i][j];} diagonala principală
```

```
                                        for (j=1;j<=n;j++)
                                            a[i][j]=0;
                                    for (i=1;i<=m;i++)
sau:                                    {cout<<"dati extremitatile muchiei "<<i;
.....                                   cin>>x >>y;
cin>>n; cin>>m;                         a[x][y]=1; a[y][x]= 1;}
for (i=1;i<=n;i++)
```

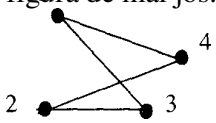
2. Matricea de adiacentă **are toate elementele de pe diagonala principală** egale cu **0**.
3. **Numărul elementelor egale cu 1 de pe linia i** (sau coloana i) este egal cu **gradul vârfului i**. Dacă vârful **i** este un **vârf izolat pe linia i** (şi coloana i) **nu sunt elemente egale cu 1**.

Liste de adiacență

Fie G=(V, M) un graf neorientat cu n vârfuri (V={1,2, ..., n}) şi m muchii.

Reprezentarea grafului G prin **liste de adiacență** constă în:
- precizarea **numărului de vârfuri, n**;
- pentru fiecare vârf i, se precizează **lista L_i a vecinilor săi**, adică lista nodurilor adiacente cu nodul i.

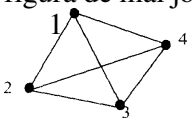
Exemplul 1. Fie graful reprezentat grafic ca in figura de mai jos:



Reprezentarea sa prin liste de adiacențe presupune:
- precizarea numărului de vârfuri n, n=4;
- precizarea listei vecinilor lui i, pentru i=1..n, astfel:

Vârful i	**Lista vecinilor lui i**
1	3,4
2	3,4
3	1,2
4	1,2

Exempul 2. Fie graful reprezentat grafic ca in figura de mai jos:



Reprezentarea sa prin liste de adiacențe presupune:
- precizarea numărului de vârfuri n, n=4;
- precizarea listei vecinilor lui i, pentru i= 1..n, astfel:

Vârful i	**Lista vecinilor lui i**
1	2,3,4
2	1,3,4
3	1,2,4
4	1,2,3

• *Comentarii:*
Acest mod de reprezentare se poate implementa astfel:
l. Se foloseşte un **tablou bidimensional T**, caracterizat astfel:
• are n +2m coloane;
• $T_{1,i}$=i pentru i=1..n;
• Pentru i=1..n $T_{2,}$i=k, dacă $T_{1,k}$ este **primul nod din lista vecinilor lui i**;
$T_{2,i}$=0, dacă **nodul i este izolat**;
• Dacă $T_{1,j}$=u, adică u este un nod din lista vecinilor lui i, atunci:
$T_{2,j}$=0, dacă **u este ultimul nod din lista vecinilor lui i**;
$T_{2,j}$=j+l, dacă **u nu este ultimul nod din lista vecinilor lui i.**

Exemplu de completare a tabloului pentru graful de la exemplul 1
Prima etapă. Se numerotează coloanele (1 ..n+2m) şi se trec vârfurile.

1	2	3	4	5	6	7	8	9	10	11	12
1	**2**	**3**	**4**								

A doua etapă. Se trec in tabel vecinii lui 1, începând de la coloana 5.

1	2	3	4	**5**	6	7	8	9	10	11	12
1	2	3	4	**3**	**4**						
5				**6**	**0**						

$T_{2,1}$=5, pentru ca primul vecin (3) al lui 1 s-a trecut la coloana 5 ($T_{1,5}$=3).
$T_{2,5}$=6, pentru că următorul vecin (4) al lui l s-a trecut la coloana 6 ($T_{1,6}$=4).

$T_{2,6}$=0, pentru ca vecinul $T_{1,6}$ (4) al lui 1 este ultimul din listă.

A treia etapă. Se trec in tabel vecinii lui 2, începând de la coloana 7.

1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	3	4	**3**	**4**				
5	**7**			6	0	**8**	**0**				

$T_{2,2}$=7, pentru că primul vecin (3) al lui 2 s-a trecut la coloana ($T_{1,7}$=3).

$T_{2,7}$=8, pentru ca următorul vecin (4) al lui 2 s-a trecut la coloana 8 ($T_{1,8}$=4).

$T_{2,8}$ =0, pentru că vecinul $T_{1,8}$ (4) al lui 2 este ultimul din listă.

A patra etapă. Se trec in tabel vecinii lui 3, începând de la coloana 9.

1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	3	4	3	4	**1**	**2**		
5	7	**9**		6	0	8	0	**10**	**0**		

$T_{2,3}$=9, pentru că primul vecin (1) al lui 3 s-a trecut la coloana 9 ($T_{1,9}$=1).

$T_{2,9}$=10, pentru că următorul vecin (2) al lui 3 s-a trecut la coloana 10 ($T_{1,10}$=2)

$T_{2,10}$= 0, pentru că vecinul $T_{1,10}$ (2) al lui 3 este ultimul din listă.

Ultima etapă. Se trec in tabel vecinii lui 4, începând de la coloana 11.

1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	3	4	3	4	1	2	**1**	**2**
5	7	9	**11**	6	0	8	0	10	0	**12**	**0**

$T_{2,4}$=11, pentru că primul vecin (1) al lui 4 s-a trecut la coloana 11 ($T_{1,11}$=1).

$T_{2,11}$=12, pentru că următorul vecin (2) al lui 4 s-a trecut la coloana 12 ($T_{1,12}$=2)

$T_{2,12}$=0, pentru că vecinul $T_{1,12}$ (2) al lui 4 este ultimul din listă.

2. Se foloseşte un **tablou unidimensional**, cu numele **cap**, şi un **tablou bidimensional**, cu numele **L** (care reține listele de vecini pentru fiecare nod), caracterizate astfel:

Tabloul **cap**:

- are **n componente**;
- **cap_i =c**, dacă **primul nod** din lista **vecinilor lui i** este trecut in tabloul **L** la coloana **c**, adică **$L_{1,c}$** este **primul vecin al lui i**,

şi **cap_i =0**, dacă **nodul i este izolat**

Tabloul **L**:

- are **2m componente**;
- dacă **k** este un **vecin al nodului i**, atunci:

$L_{1,k}$=k şi **$L_{2,p}$=0**, dacă **k este ultimul vecin din listă**, sau

$L_{1,p}$=k şi **$L_{2,p}$= p+l** dacă **k nu este ultimul vecin din listă** (**p** este **coloana la care s-a ajuns** în tabloul **L**).

• **Exemplu** de completare a tablourilor cap şi L, pentru graful de la exemplul 1.

Tabloul cap

1	2	3	4
1	**3**	**5**	**7**

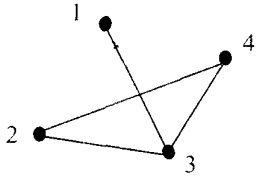
Tabloul L

1	2	**3**	4	**5**	6	**7**	8
3	4	3	4	1	2	1	2
2	0	4	0	6	0	8	0

3. Se foloseşte un **tablou bidimensional**, cu numele **L**, caracterizat astfel:

- are n **linii**;
- **pe linia i se trec vecini nodului i**.

• **Exemplu** de completare a tabloului L, pentru graful:



Tabloul L

3		
3	4	
1	2	4
2	3	

Implementarea în limbajul C++, a ideii prezentate mai sus, se realizează conform secvenței de program prezentată mai jos.

```
.....
int L[20][20];
int nr_vec[20];
cout<<"n="; cin>>n;
for (i=1;i<=n;i++)
    {cout<<"Dati numarul veciniior nodului "<<i; cin>>nr_vec[i];
    for (j=1;j<=nr_vec[i];j++)
        cin>>L[i][j];}
.....
```

Construirea matricei de adiacență când se cunoaşte L (listele vecinilor fiecărui nod).

```
.....
for (i=1;i<=n;i++)
    {for (j=1;j<=nr_vec[i];j++)
        a[i][L[i][j]]=1;}
.....
```

Construirea tabloului L (listele vecinilor nodurilor) **când se cunoaşte matricea de adiacentă**.

```
.....
for (i=1;i<=n;i++)
    {k=0;
    for (j=l;j<=n;j++)
        if (a[i][j]==1)
            {k=k+l ;
            L[i][k]=j;}
    }
```

4. Se foloseşte un **tablou unidimensional,** cu numele **L**, caracterizat astfel:
- componentele sale sunt de tip **referință**;
- are **n componente**;
- $\mathbf{L_i}$ pointează spre începutul listei vecinilor nodului i.

Şirul muchiilor

Fie G=(V, M) un graf neorientat cu n vârfuri (V={1,2,..., n}) şi m muchii. Reprezentarea grafului G constă in precizarea numărului n de noduri si numărului m de muchii, precum şi în precizarea extremităților pentru fiecare muchie în parte.

Comentarii:

Acest mod de reprezentare se implementează astfel:

1. Se dă numărul n de noduri, numărul m de muchii şi extremitățile fiecărei muchii, care sunt trecute în vectorii el şi e2 astfel:

extremitățile primei muchii sunt e1[i] si e2[1];
extremitățile celei de-a doua muchie sunt el[2] şi e2[2];
.....
deci M={ [e1[1],e2[1]] , [e1[2],e2[2]] ,..., [e1[m],e2[m]]}.

Secvența C++ corespunzătoare este:

```
int el[100],e2[l00];
int n, m, i;
.....
cout<<"n="; cin>>n;
```

```
cout<<"m="; cin>>m;
for (i=1;i<=m;i++)
    {cout<<"Dati extremitatile muchiei cu numarul "<<i;
    cin>>e1[i]>>e2[i];}
.....
```

Construirea matricei de adiacență, când se cunoaşte şirul muchiilor ca mai sus.

```
.....
cout<<"n="; cin>>n;
for (i=1;i<=m;i++)
    {a[el[i]][e2[i]]=1;
    a[e2[i]][el[i]]=1;}
```

Construirea şirului muchiilor, ca mai sus, când se dă matricea de adiacență.

```
.....
k=0;
for (i=l;i<=n-l;i++)
    for (j=i+1;j<=n;j++)
        if (a[i][j]=1)
                {k=k+1;
                e1[k]=i;
                e2[k]=j;}
m=k;
.....
```

2. Se foloseşte un **tablou unidimensional**, cu numele **e**, caracterizat astfel:
- componentele sale sunt de **tip structură**;
- are **m componente**;
- e_i reprezintă **muchia i**.

Pentru implementare este nevoie de:

```
typedef struct{
                    int x;
                    int y;
                    } muchie;
muchie e[20];
.....
```

Accesul la muchia **i** se face: **e[i].x ..... e[i].y**

Secvența C++ corespunzătoare este:

```
.....
cout<<"n="; cin>>n;
cout<<"m="; cin>>m;
for (i=1;i<;=m;i++)
        {cout<<"Dati extremitatile muchiei cu numarul "<<i;
        cin>>e[i].x>>e[i].y;}
```

Construirea matricei de adiacență, când se cunoaşte şirul muchiilor ca mai sus.

```
.....
cout<<"n=”; cin>>n;
for (i=l;i<=m;i++)
        {a[e[i].x][e[i].y]=1;
        a[e[i].y][e[i].x]=1;}
.....
```

Construirea şirului muchiilor, ca mai sus, când se dă matricea de adiacență.

```
.....
k=0;
for (i= 1;i<=n-1;i++)
    for (j=i+l;j<=n; j++)
```

```
        if (a[i][j]==1)
            { k =k+1;
            e[k].x=i;
            e[k].y=j;}
m=k;
.....
```

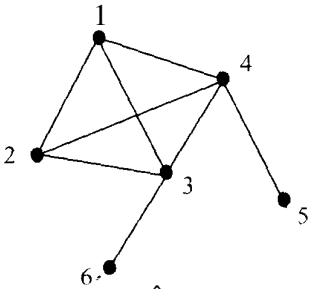
9. Parcurgerea grafurilor

Fie G=(V, M) graf neorientat, unde V={x_1, x_2,..., x_n } şi M={m_1, m_2,..., m_p}.
Prin **parcurgerea grafului** G se înţelege vizitarea, într-un mod sistematic, a tuturor nodurilor, plecând de la un nod pl (nod de plecare) mergând pe muchii incidente două câte două.
Un graf poate fi parcurs în următoarele două moduri:
- **în lătime** (**BF** = Breadth First)
- **în adâncime** (**DF** = Depth First)

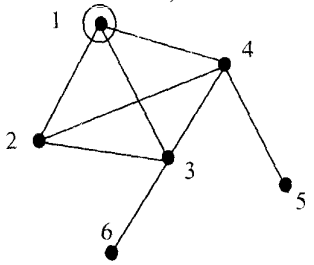
Parcurgerea în lăţime (BF- breadth first)

Fie G=(V, M) un graf neorientat cu n vârfuri (V={1,2, ..., n}) şi m muchii.
Algoritmul-de parcurgere a grafului în lăţime, folosind o coadă, este:
- iniţial **toate nodurile se consideră nevizitate**;
- se citeşte nodul de plecare **pl**, care **se consideră acum vizitat,** şi **se trece in coadă pe ,prima poziţie**;
- **se trec in coadă toate nodurile nevizitate până in prezent şi sunt adiacente cu nodul de plecare** (odată cu trecerea lor in coadă se marchează ca fiind vizitate);
- **se trece la următorul element din coadă**, care ia rolul nodului de plecare, şi **se reia pasul anterior**;
.....
- algoritmul se termină **după ce sunt parcurse toate elementele din coadă**.

• Exemplul 1. Fie graful reprezentat grafic ca în figura de mai jos:



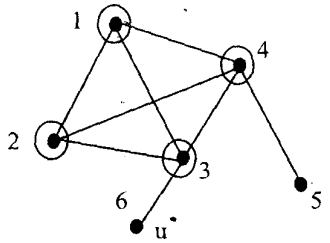
Observaţie. În continuare, un nod se consideră vizitat când este încercuit:
şi cu p şi u notăm indicele primului, respectiv ultimului element din coadă.
Parcurgerea in lăţime, a grafului de mai sus, presupune parcurgerea etapelor:
* La început, nici un nod nu este vizitat (graful arată ca in figura iniţială, adică nici un nod nu este încercuit).
* Se pleacă de la nodul 1, care se trece in coadă pe prima poziţie şi se marchează ca fiind vizitat.



p=u

1							

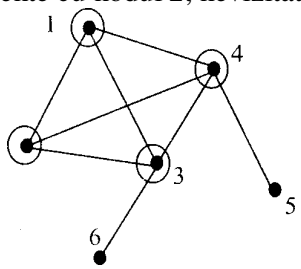
*Se vizitează şi se trec în coadă toate nodurile adiacente cu nodul 1, nevizitate încă (2,3,4).



p u

1	2	3	4				

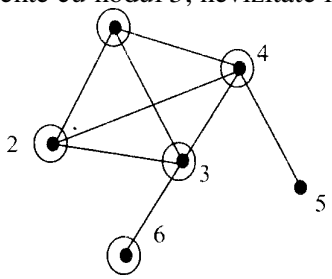
* Se trece la următorul element din coadă; acesta este 2. Se vizitează şi se trec în coadă toate nodurile adiacente cu nodul 2, nevizitate încă (nu este nici un nod care să verifice condiţiile).



p u

1	2	3	4				

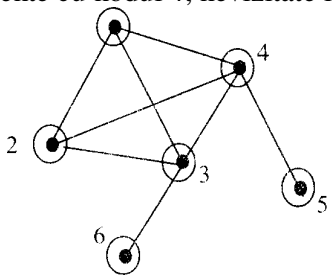
*Se trece la următorul element din coadă; acesta este 3. Se vizitează şi se trec în coadă toate nodurile adiacente cu nodul 3, nevizitate încă (b).



p u

1	2	3	4	6			

* Se trece la următorul element din coadă; acesta este 4. Se vizitează şi se trec în coadă toate nodurile adiacente cu nodul 4, nevizitate încă (5).



p u

1	2	3	4	6	5		

*Se trece la următorul element din coadă: acesta este 6. Se vizitează şi se trec în coadă toate nodurile adiacente cu nodul 6, nevizitale încă (nu este nici un nod care să verifice condiţiile).

p u

1	2	3	4	6	5		

*Se trece la următorul element din coada; acesta este 5. Se vizitează şi se trec în coada toate nodurile adiacente cu nodul 5, nevizitate încă (nu este nici un nod care să verifice condiţiile).

p = u

1	2	3	4	6	5		

* Algoritmul se încheie aici (nu mai sunt noduri).

Deci, **parcurgerea in lăţime a grafului** este: 1 2 3 4 6 5

În continuare, vom prezenta programele C++ care implementează algoritmul prezentat mai sus.

Programul 1 (abordare nerecursivă)

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int a[20][20],coada[20], viz[20];
int       i, n , el, j, p, u, pl, m, x, y;
void main()
{clrscr();
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;}
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]<< " ";
getch();}
```

Programul 2 (abordare recursivă)

Comentarii la funcția parc_latime:

- are un parametru formal, i, care reprezintă poziția curentă la care s-a ajuns în coadă;
- procedează astfel:
    - se parcurg nodurile grafului, cu j:
        dacă j este adiacent cu nodul curent din coadă şi j este nevizitat
            - se adaugă la coadă;
            - şi se marchează ca fiind vizitat;
    - dacă mai sunt elemente în coadă se trece la următorul şi se reapelează funcția

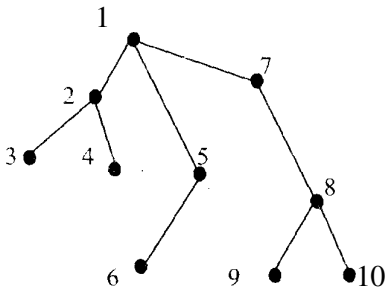
```
# include <conio.h>
#include <iostream.h>
#include <stdio.h>
int a[20][20];
int coada[20], vizitat[20];
int       i, n , j, u, pl, m,x, y;
void parc_latime(int i)
{int j;
for (j=1;j<=n;j++)
          if ((a[coada[i]][j]==1) && (vizitat[j]==0))
                                    { u=u+1;
                                    coada[u]=j;
                                    vizitat[j]=1 ;}
if (i<=u) parc_latime(i+1);}
```

```
void main()
{ clrscr();
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;}
for (i=1;i<=n;i++)
        viz[i]=0;
cout<<"dati nodul de plecare :"; cin>>pl;
vizitat[pl]=1;
coada[1]=pl;
u=1 ;
parc_latime(1);
for (i=1;i<=u;i++)
        cout<<coada[i]<<" ";
getche(); }
```

Parcurgerea in adâncime (DF-depth first)

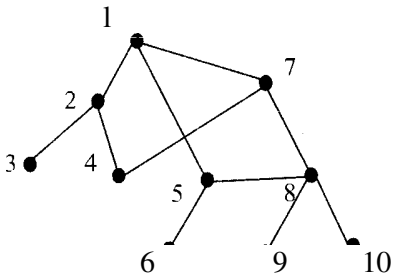
Fie G =(V, M) un graf neorientat cu n vârfuri (V={1 ,2, ... n}) şi m muchii.
Algoritmul recursiv **de parcurgere a grafului in adâncime** este implementat în funcţia parc_adancime, caracterizată astfel:
- are un parametru formal (**nodul curent**, asupra căruia se aplică):
- procedează astfel:
    - afişează **nodul asupra căruia se aplică şi-l marchează ca fiind vizitat**;
    - pentru **fiecare vecin nevizitat de-al nodului curent**
        - **se autoapelează** asupra sa.

• **Exemplul 1**. Fie graful reprezentat grafic ca în figura de mai jos:



Aplicarea algoritmului de parcurgere în adâncime, asupra grafului de mai sus, plecând de la primul nod, conduce la afişarea următoarei secvenţe:
1 2 3 **4 5 6 7 8 9 10**

• **Exemplul 2**. Fie graful reprezentat grafic ca în figura de mai jos:



Aplicarea algoritmului de parcurgere în adâncime, asupra grafului de mai sus, plecând de la primul nod, conduce la afişarea următoarei secvenţe:
1 2 3 4 7 8 5 6 9 10

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int a[20][20];
int vizitat[20];
int i,n,j,pl,m,x,y;
void parc_adancime(int pl)
{int j;
cout<<pl<<" ";
viz[pl]=1;
for (j=1; j<=n;j++)
        if ((a[pl][j]==1) && (vizitat[j]==0))
parc_adancime(j);}
void main()
{cout<<"n m "; cin>>n>>m;
for (i=1;i<=m;i++)
        {cout<<"x y "; cin>>x>>y;
        a[x][y]=1; a[y][x]=1;}
```

```
for (i=1;i<=n;i++)
        vizitat[i]=0;
cout<<"dati nodul de plecare :"; cin>>pl;
parc_adancime(pl);
getch();}
```

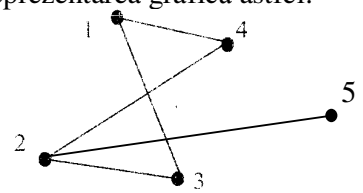
10. Conexitate

Vor fi prezentate noțiunile:
- lanț
- ciclu
- graf conex
- componentă conexă

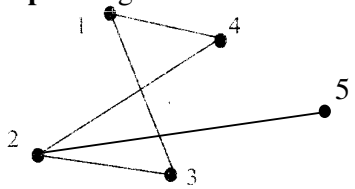
Lanț

Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **lanț**, în graful G, o succesiune de noduri, notată L = $[x_{i1}, x_{i2}, ..., x_{ik}]$, cu proprietatea că oricare două noduri consecutive sunt adiacente, altfel spus $[x_{i1},x_{i2}],..., [x_{ik-1},x_{ik}] \in M$
Se întâlnesc noțiunile:
- **extremitățile lanțului**
  - fiind dat lanțul L = $[x_{i1}, x_{i2}, ..., x_{ik}]$, se numesc **extremități ale sale nodurile** x_{i1} şi x_{ik} (x_{i1} - **extremitate inițială**; x_{ik} - **extremitate finală**);
- **lungimea lanțului**
  - fiind dat lanțul L = $[x_{i1}, x_{i2}, ..., x_{ik}]$ prin **lungimea sa** se înțelege **numărul de muchii care apar în cadrul lui**

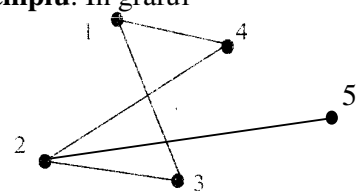
• **Exemplu** de lanț:
Fie graful G=(V, M) unde:
V={ 1,2,3,4,5}
M={[1,3], [1,4], [2,3], [2,4], [2,5]}
cu reprezentarea grafică astfel:



Lanțul L1=[1, 3, 2, 4] este în graful G lanț cu lungimea 3 şi extremitățile 1 şi 4.
L2=[5, 2, 4, 1, 3, 2] este în graful G lanț cu lungimea 5 şi extremitățile 5 si 2.
Observație Dacă L=$[x_{i1}, x_{i2}, ..., x_{ik}]$, este lanț în graful G, atunci şi L1= $[x_{ik},...,x_{i2},x_{i1}]$, este lanț în graful G.
Definiție. Fie G-(V, M) un graf neorientat. Se numeşte **lanț elementar**, în graful G, lanțul L = $[x_{i1}, x_{i2}, ..., x_{ik}]$, cu proprietatea că oricare două noduri ale sale sunt distincte (altfel spus: printr-un nod nu se trece decât o singură dată).
• **Exemplu**: În graful



lanțul L1=[l, 3, 2, 4] este lanț elementar.
Definiție. Fie G=(V,M) un graf neorientat. Se numeşte **lanț neelementar** în graful G lanțul L=$[x_{i1}, x_{i2}, ..., x_{ik}]$, cu proprietatea că nodurile sale nu sunt distincte două câte două (altfel spus: prin anumite noduri se trece de mai multe ori).
•**Exemplu**: În graful



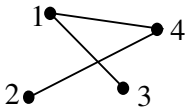
lanțul L2=[5, 2, 4, 1, 3, 2] este lanț neelementar (prin 2 s-a trecut de două ori).

Ciclu

Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **ciclu**, în graful G, lanțul C = [x_{i1} , x_{i2}, ,..., x_{ik}], cu proprietatea că $x_{i1}=x_{ik}$ şi **are muchiile diferite două câte două**.

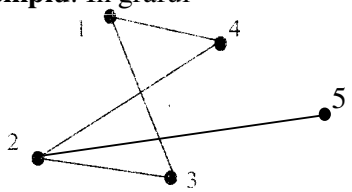
• **Exemplu**: În graful

lanțul C=[1, 3, 2, 4, 1] este ciclu.



Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **ciclu elementar**, în graful G, un ciclu cu proprietatea că **oricare două noduri ale sale, cu excepția primului şi a ultimului, sunt distincte**.

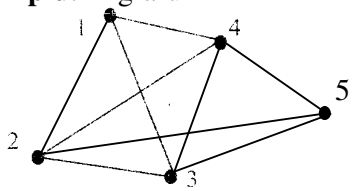
• **Exemplu**: În graful



ciclul C=[1, 3, 2, 4, 1] este ciclu elementar.

Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **ciclu neelementar**, în graful G, un ciclu cu proprietatea că **nodurile sale, cu excepția primului şi a ultimului, nu sunt distincte.**

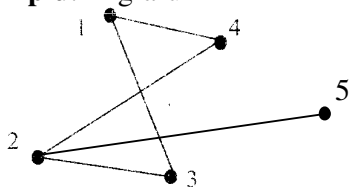
• **Exemplu**: În graful



ciclul C=[5, 3, 4, 1, 2, 4, 5] este ciclu neelementar (prin 4 s-a trecut de două ori).

Definiție. Fie G=(V, M) un graf neorientat. Două **cicluri** C1 şi C2 sunt **egale**, dacă muchiile lor induc acelaşi graf parțial al subgrafului generat de vârfurile ce aparțin lui C1, respectiv lui C2.

• **Exemplu**: În graful



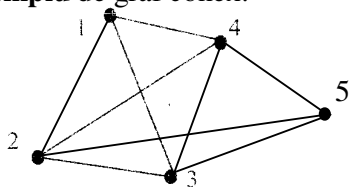
ciclul Cl=[1, 3, 2, 4, 1] este egal cu ciclul C2=[3, 2, 4, 1, 3].

Observație. Un **ciclu** se numeşte **par**, dacă **lungimea sa** este **un număr par** şi se numeşte **impar,** dacă **lungimea sa** este un număr **impar**.

Graf conex

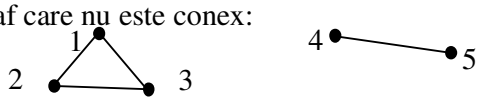
Definiție. Fie G=(V,M) un graf neorientat. Graful G se numeşte **conex** dacă pentru oricare două vârfuri x şi y, x≠y, **există un lanț în C de la x la y**.

• **Exemplu** de graf conex:



Graful este conex, deoarece oricare ar fi vârfurile x şi y, x≠y, există un lanț în G care să le lege.

•**Exemplu** de graf care nu este conex:



Graful nu este conex, deoarece există două vârfuri, cum ar fi 1 si 4, pentru care nu există nici un lanț în graf care să le lege.

Componentă conexă

Definiție. Fie G=(V, M) un graf neorientat. Se numeşte **componentă conexă**, un graf neorientat $G_1=(V_1,M_1)$ care verifică următoarele condiții:

- este **subgraf al grafului G**;
- este **conex**;
- **nu există nici un lanț în G care să lege un nod din V_1 cu un nod din $V-V_1$**.

•**Exemplu**: Fie graful G=(V, M) : V={ 1,2,3,4,5,6},M={[1,2], [1,3], [2,3],[4,5], [4,6]}

Pentru graful de mai sus, graful G1=(V1,M1) unde: V1={4,5,6} şi M1={ [4,5], [4,6]} este componentă conexă, deoarece:

- este subgraf al grafului G;
- este conex:
- nu există nici un lanț în G care şi lege un nod din V_1, cu un nod din $V-V_1$ ={1,2 3},

La fel se poate spune şi despre graful G2=(V2,M2) unde: V2={1,2,3} şi M2={[1,2], [1,3], [2,3]}

În concluzie, graful, din figura de mai sus, este format din două componente conexe.

Observație. Fie G=(V, M) un graf neorientat. **Graful G este conex** dacă şi numai dacă **este format dintr-o singură componentă conexă.**

• **Exemplu** de graf conex (este format dintr-o singură componentă conexă):

Observație. Fie G=(V, M) un graf neorientat. Pentru **a verifica dacă graful este format din una sau mai multe componente conexe** se procedează astfel:

- se parcurge graful, prin una din metodele de parcurgere;
- dacă după parcurgere mai există în graf noduri nevizitate, atunci graful este format din mai multe componente conexe, altfel este format dintr-o singură componentă conexă, adică graful este conex.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int a[20][20];
int viz[20];
int i, n ,j, pl, m, x, y, ok;
void parc_adancime(int pl)
{int j;
viz[pl]=1;
for (i=1;j<=n;j++)
if ((a[pl][j]==1) && (viz[j]==0))
parc_adancime(j);}
void main(){
cout<<"n m "; cin>>n>>m;
for (i=1;i<=m;i++)
    {cout<<"x y"; cin>>x>>y;
    a[x][y]=1; a[y][x]=1;}
for (i=1;i<= n;i++) viz[i]=0;
cout<<"dati nodul de plecare :"; cin>>pl;
parc_adancime(pl);
```

```
ok=0;                                   //se verifica daca mai sunt
for (i=1;i<=n;i++)                      //noduri nevizitate
        if (viz[i]==0)  ok=1;
if (ok) cout<<"graful este format din mai multe componente conexe";
        else cout<<"graful este conex";
getche( ); }
```

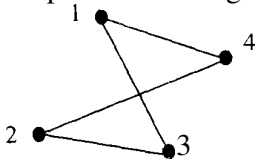
11. Grafuri Hamiltoniene

Definiţie. Fie G=(V, M) un graf neorientat. Se numeşte **lanţ hamiltonian**, în graful G, un **lanţ elementar care conţine toate vârfurile grafului G.**

• **Exemplu** de lanţ hamiltonian:

Fie graful G=(V, M) unde: V={1,2,3,4}, M={[l,3], [1,4],[2,3],[2,4],}

Reprezentarea sa grafică este:



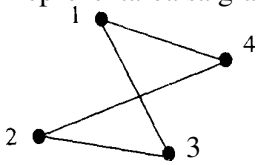
Lanţul L=1, 3, 2, 4 este, în graful G, lanţ hamiltonian.

Definiţie. Fie G=(V, M) un graf neorientat. Se numeşte **ciclu hamiltonian**, în graful G, un **ciclu elementar care conţine toate vârfurile grafului G**.

• **Exemplu** de ciclu hamiltonian:

Fie graful G=(V, M) unde: V={ 1,2,3,4} M={ [1,3], [1,4], [2,3],[2,4]}

Reprezentarea sa grafică este:



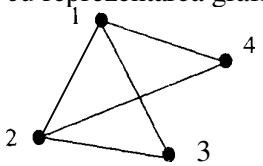
Ciclul C=1, 3, 2, 4, 1este, în graful G, ciclu hamiltonian.

Definiţie. Fie G=(V, M) un graf neorientat. Graful G este **hamiltonian** dacă conţine **cel puţin un ciclu hamiltonian.**

•**Exemplu** de graf hamiltonian:

Graful G=(V, M) unde:V={ 1,2,3,4} M={[1,2], [1,3], [1,4], [2,31, [2,4]}

cu reprezentarea grafică:



este hamiltonian, deoarece conţine cel puţin un ciclu hamiltonian; ciclul C= l, 3, 2, 4, 1 este, în graful G, ciclu hamiltonian.

Observaţie. Fie G=(V, M) un graf neorientat. Ca în graful G să existe un **ciclu hamiltonian**, trebuie ca el să **aibă cel puţin trei vârfuri**.

Teoremă. Graful complet K_n este **graf hamiltonian**.

Demonstraţie:

Orice succesiune x_{i1} , x_{i2}, ,..., x_{in}; x_{i1} de n+ 1 noduri distincte (excepţie fac primul şi ultimul) am alege, poate fi privită ca un ciclu hamiltonian, deci graful K_n este hamiltonian.

Teoremă. Fie G= (V, M) un graf neorientat. Dacă **are n≥3 noduri** şi gradul fiecărui vârf x verifică relaţia **d(x) ≥n/2**, atunci G este **hamiltonian**.

Problema determinări tuturor ciclurilor hamiltoniene dintr-un graf neorientat.

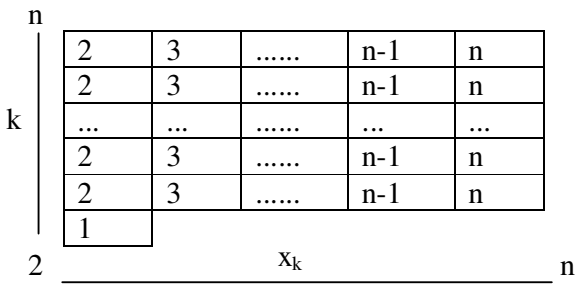
Fie G=(V, M) un graf neorientat, cu n vârfuri. Să se determine toate ciclurile hamiltoniene din graful G.

Rezolvare: Problema se rezolvă folosind metoda Backtracking. Pentru rezolvare se vor folosi:

k : variabilă întreagă care reprezintă la al câtelea nod s-a ajuns(al doilea, al treilea...)

x : vector cu componente întregi cu proprietatea: x_k :reprezintă al k-lea nod din ciclu.

Observaţie. Pentru a evita parcurgerea unui drum de 2 ori se va recurge la strategia de a atribui lui x, valoarea l, adică toate ciclurile să plece de la primul nod: din acest motiv $x_k \in \{2.... n\}$ pentru $k \in \{2,... n\}$
În concluzie, a rezolva problema înseamnă a determina vectorii:
$x=(x_1,x_2,...,x_n)$
unde $x_1=1$ şi $x_k \in \{2.... n\}$ pentru $k \in \{2,... n$
Tabla va arăta astfel:



2	3	.....	n-1	n
2	3	.....	n-1	n
...	...	.....	...	...
2	3	.....	n-1	n
2	3	.....	n-1	n
1				

Pentru reprezentarea grafului în program se va folosi matricea de adiacenţă, definită astfel:
$a_{i,j}=1$, dacă există muchie între nodurile i şi j;
$a_{i,j}=0$, dacă nu există muchie între nodurile i şi j.
Exemplu: Pentru graful de mai jos

matricea de adiacenţă se defineşte astfel:

$$A=\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Concluzii:
1. Între nodurile k si i, există muchie dacă $a_{k,i}=1$ (şi $a_{i,k}=1$), deci între nodurile x_k şi x_j există muchie dacă $a[x_k][x_i]=1$ (si $a[x_i][x_k]=1$).
2. Nodul x_k trebuie să fie diferit de nodul x_i, pentru i=1...k-1.
3. Nodul x_n trebuie să fie legat prin muchie de nodul x_1 adică $a[x_n][x_1]=1$
* Comentarii la procedura Valid:
Trebuie verificat că:
1. există muchie între nodurile x_{k-1} şi x_k, adică trebuie verificat că $a[x_{k-1}][x_k]=1$;
2. nodul x_k este diferit de toate nodurile prin care s-a trecut, adică:
$x_k \neq x_i$ pentru i=1...k-1.
3. dacă s-a ajuns la al n-lea nod din ciclu, trebuie să existe muchie între acesta primul nod, adică: dacă k=n atunci $a[x_k][x_1]=1$

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir[20];
sir x;
int i, k, n;
int as, ev;
int a[20][20];
void succ(sir x, int k, int &as)
{if (x[k]<n)
        {as=1;
        x[k]=x[k]+1;}
```

```
    else as=0;}
void valid( sir x, int k, int &ev)
{ev=l ;
if (a[x[k-1]][x[k]]==0)          dacă nu există drum între x_{k-1} şi x_k
                ev=0;
    else
        {for (i=1;i=k-l;i++)
                if (x[i]==x[k]) ev=0;
        if ((k==n) && (a[x[n]][x[l]]==0)) ev=0;}
}
void afis(sir x, int k)
{int i;
for (i=l;i<=k;i++)
        cout<<x[i]<<" ";
 cout<<x[1]<<" "; cout<<endl;}
void main() {
cout<<"n= "; cin>>n;
for (i=1;i<=n-l;i++)
    for (j=i+l;j<=n;j++)
          {cin>>a[i][j];
          a[j][i]=a[i][j]; }
x[1]=1;
k=2;
x[k]=1;          priviţi tabla
while (k>1){
                do{succ(x,k,as);
                    if (as)        valid(x,k,ev);
                }while (as && !ev);
                if (as)
                        if (k==n) afis(x,k);
                            else
                                {k=k+ 1;
                                x[k]=1;}
                    else k=k-l;
}
getch();}
```

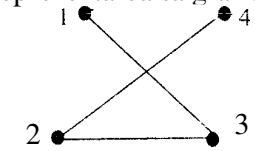
12. Grafuri Euleriene

Definiţie. Fie G=(V, M) un graf neorientat. Se numeşte **lanţ eulerian,** in graful G, un lanţ care conţine toate muchiile grafului G, fiecare muchie apărând în lanţ o singură dată.

• **Exemplu** de lanţ eulerian:

Fie graful G=(V, M) unde: V={ 1,2,3,4} M={[1,3],[2,3], [2,4]}

Reprezentarea sa grafică este:



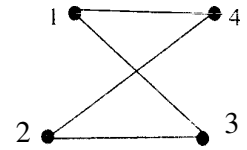
Lanţul L=[1, 3, 2, 4] este, in graful G, lanţ eulerian.

Definiţie. Fie G= (V,M) un graf neorientat. Se numeşte **ciclu eulerian,** în graful G, un ciclu care conţine toate muchiile grafului G, fiecare muchie apărând în ciclu o singură dată.

• **Exemplu** de ciclu eulerian:

Fie graful G=(V, M) unde: V={1,2,3,4} M={ [1,3], [1,4], [2,3], [2,4]}

Reprezentarea sa grafică este:



Ciclul C=[l, 3, 2, 4, l] este, în graful G, ciclu eulerian.

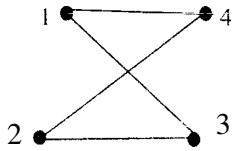
Teoremă. Fie G=(V, M) un graf neorientat. Graful G, fără vârfuri izolate, este eulerian dacă şi numai daca este conex şi gradele tuturor vârfurilor sale sunt numere pare.

•**Exemplul** l.

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

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

Reprezentarea sa grafică este:



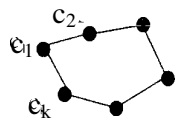
este graf eulerian, deoarece verifică condiţiile teoremei anterioare, adică:

- nu are vârfuri izolate;
- este conex;
- gradele tuturor vârfurilor sunt numere pare.

Definiţie. Un graf care conţine cel puţin un **ciclu eulerian** se numeşte **graf eulerian**.

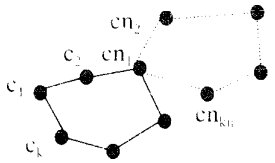
Algoritmul de determinare a unui ciclu eulerian într-un graf neorientat.

Pas 1. Se detemină ciclul C=(c_1 ,c_2 ,..., c_k, c_1), unde c_1=1.

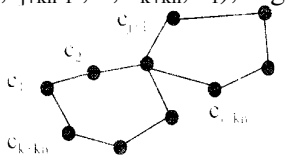


Pas 2. Se caută, printre nodurile ciclului determinat la pasul anterior, un nod pentru care mai există muchii incidente cu el neluate încă. Fie acesta c_j construim ciclul C_n= (cn_1 ,cn_2 ,..., cn_k, cn_1), unde cn_i=c_j

cn- ciclu nou

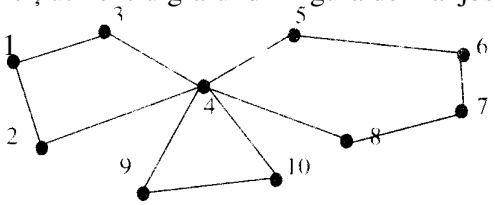


Pas 3.Ciclul determinat la pasnl 2 se "concatenează" la ciclul C, obţinându-se astfel ciclul C=(c_1 ,c_2 ,..., c_j, c_{j+1}...,c_{j+kn-1} ,..., c_{k+kn}, c_1), figurat mai jos:



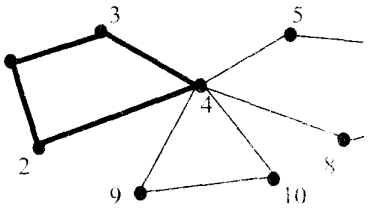
Pas4. Dacă nu sau ales toate muchiile, se reia pasul 2.

• Exerciţiu: Pentru graful din figura de mai jos, să se determine un ciclu eulerian.

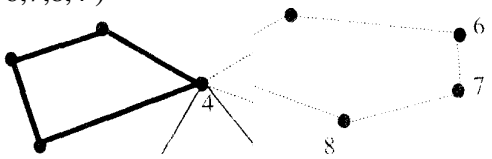


Rezolvare:

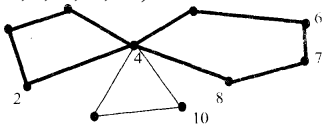
Pas1. Se determină un ciclu plecând de la nodul 1; fe acesta: C-('l, 3, 4, 2, 1) (priviţi figura).



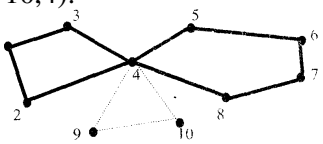
Pas2. Se caută, printre nodurile ciclului determinat la pasul anterior, un nod pentru care mai există muchii incidente cu el neluate încă; fie acesta nodul 4 -1. Construim ciclul Cn, plecând de la acest nod: Cn= (4, 5, 6,7,8,4)



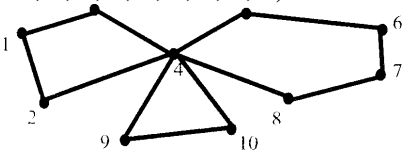
Pas3. Ciclul determinat la pasul 2 se "concatenează la ciclul C, obținându-se astfel ciclul: C=(1, 3, 4, 5, 6, 7,8, 4, 2, 1)



Pas4. Deoarece mai sunt muchii neluate încă, se reia pasul 2.
Pas2. Se caută, printre nodurile ciclului determinat până în prezent, un nod pentru care mai există muchii incidente cu el neluate încă; acesta este nodul 4. Construim ciclul Cn, plecând de la acest nod: Cn=(4, 9, 10,4).



Pas 3. Ciclul determinat la pasul 2 se "concatenează la ciclul C, obținîndu-se astfel ciclul: C=(1, 3, 4, 9, 10, 4, 5, 6, 7, 8, 4, 2, 1).



programul de determinare a unui ciclu eulerian într-un graf neorientat.

```
#include <iostream.h>
#include<conio.h>
#include<stdio.h>
typedef int mut[20][20];
typedef int sir[20];
int a[20][20];
sir viz, c, cn, gr;
int i, k, j, n, kn, poz, m, eulerian, x, y;
int grad(int i)
{ int s, j;
s=0;
for (j=1;j<=n;j++) s=s+a[i][j];
return s;}
int varf_izolate()
{ int i, ok;
ok=0;
for (i=1; i<=n; i++)
        if (grad(i)==0) ok=1;
return ok;}
int grade_pare()
{ int i, ok;
```

```
ok=1;
for (i=1;i<=n;i++)
        if (grad(i)%2!=0) ok=0;
return ok;}
void adancime(int i )
{ int j;
viz[i]=1;
for (j=1;j<=n;j++)
        if ((viz[j]==0) && (a[i][j]==1)) adancime(j);}
int conex()
{int i, ok;
        for (i=1;i<=n;i++) viz[i]=0;
adancime(1);
ok=1;
for (i=1;i<=n;i++)
        if (viz[i]==0)  ok= 0;
return ok;}
void main( )
{ clrscr( );
cout<<"m="; cin>>m;
cout<<"n="; cin>>n;
for (i=1;i<=m;i++)
{ cout<<"x y"; cin>>x>>y;
a[x][y]=a[y][x]=1;}
for (i=1;i<=n;i++)  gr[i]=grad(i);
eulerian=(!varf_izolate()) && grade_pare()&&conex();
if (!eulerian) cout<<"graful nu este eulerian";
        else
            {c[1]=1;
            k=1;
            do{
                for (j=1;j<=n;j++)
                    if (a[c[k]][j]==1)
                        {k=k+1;
                        c[k]=j;
                        gr[c[k] ]=gr[c[k]]-1;
                        gr[c[k-1]]=gr[c[k-1]]-1;
                        a[c[k-1]][c[k]]=0;
                        a[c[k]][c[k-1]]=0;
                        break;}
                }while (c[k]!=1);
            while (k-1<m){
                        for (j=1;j<=k-1;j++)
                        if (gr[c[j]]>0) { cn[1]=c[j];
                                        poz= j;
                                        break;}
                        kn=1;
                        do{
                        for (j=1;j<=n;j++)
                            if (a[cn[kn]][j]==1)
                                        { kn=kn+1;
                                        cn[kn] =j;
                                        gr[cn[kn]]=gr[cn[kn]]-1 ;
                                        gr[cn[kn-1]]=gr[cn[kn-1]]-1;
                                        a[cn[kn-1]][cn[kn]]=0;
```

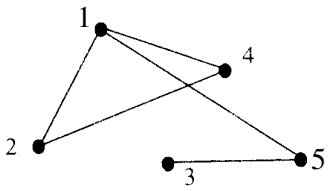
```
m=12
n=10
x y1 2
x y2 4
x y1 3
x y3 4
x y4 5
x y5 6
x y6 7
x y7 8
x y8 4
x y4 9
x y4 10
x y9 10
1 2 4 9 10 4 5 6 7 8 4 3 1
```

```
                                    a[cn[kn]][cn[kn-1]]=0;
                                    break;}
                            }while (cn[kn]!=cn[1]);
                            for (j=k;j>=poz;j--) c[j+kn-1]=c[j] ;
                            for (j=1 ;j<=kn-1;j++) c[poz+j]=cn[j+1];
                            k=k+kn-1;
                            }
                }
for (i=1;i<=k;i++)
cout<<c[i]<<" ";
getche(); }
```

Probleme

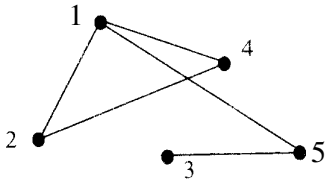
1. **Noţiunea de graf neorientat**

1. Să se precizeze dacă reţelei de circulaţie din oraşul dumneavoastră i se poate asocia un graf neorientat; în caz afirmativ, să se definească graful corespunzător.
2. Având la dispoziţie un grup de n persoane, n∈ N* , să se precizeze dacă i se poate asocia un graf neorientat; în caz afirmativ, să se definească graful corespunzător.
3. Având la dispoziţie o hartă cu n ţării, n∈ N*. să se precizeze dacă i se poate asocia un graf neorientat; în caz afirmativ, să se definească graful corespunzător.
4. Având la dispoziţie toate stelele, să se precizeze dacă li se poate asocia un graf neorientat; justificaţi răspunsul.
5. Pentru graful reprezentat în figura de mai jos
a) precizaţi mulţimea nodurilor;
b) precizaţi mulţimea muchiilor;
c) daţi exemple de noduri adiacente;
d) pentru fiecare muchie precizaţi extremităţile sale;
e) daţi exemple de muchii incidente.

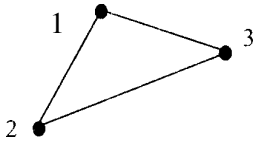


2. Noţiunea de graf parţial

1. Să se determine două grafuri parţiale ale grafului de mai jos:



2. Să se determine toate grafurile parţiale ale grafului de mai jos:

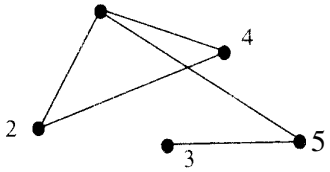


3. Fie G un graf neorientat, cu n vârfuri şi m muchii. Să se determine numărul grafurilor parţiale ale grafului G.
4. Se citesc din 2 fişiere text informaţii despre 2 grafuri neorinetate: pe prima linie numărul de noduri şi de pe următoarele rânduri, matricea de adiacenţă. Să se verifice daca graful reţinut în cel de-al doilea fişier este graf parţial al grafului reţinut în primul fişier
5. Se citesc dintr-un fişier informaţiile despre graful neorintat: de pe prima linie numărul de noduri n şi eticheta unui nod x, şi apoi, de pe următoarele rânduri, matrice de adiacenţă a grafului. Să se afişeze într-un alt fişier text informaţiile despre graful paţial obţinut prin eliminarea tuturor muchiilor care au extremităţi un nod cu gradul par şi nodul x.

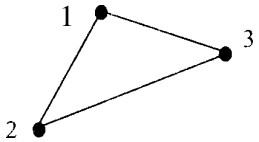
3. Noţiunea de subgraf

1. Să se determine două subgrafuri ale grafului de mai jos:

1



2. Să se determine toate subgrafurile grafului de mai jos:

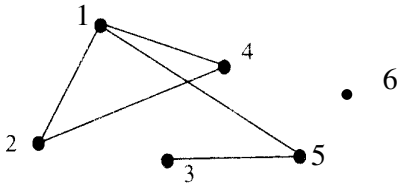


3. Fie G un graf neorientat, cu n vârfuri şi m muchii. Să se determine numărul subgrafurilor grafului G.

4. Se citesc din 2 fişiere informaţiile despre 2 grafuri neorintate: de pe prima linie numărul de noduri n şi apoi, de pe următoarele rânduri, matricea de adiacenţă a grafului. În fisierul al doilea pe ultimul rând după matricea de adiacenţă, este memorat un şir ce reprezintă etichetele nodurilor. Să se verifice dacă graful 2 este subgraf al grafului 1.

4. Gradul unui vârf

l. Fiind dat graful de mai jos, să se determine pentru fiecare vârf în parte gradul său; să se precizeze vârfurile terminale şi vârfurile izolate.



2. Să se demonstreze că orice graf G, cu n ≥2 noduri, conţine cel puţin două vârfuri care au acelaşi grad.

3. Să se verifice dacă există grafuri cu 5 noduri pentru care:

a) şirul gradelor vârfurilor sale este: 1,2,3,0,5

b)şirul gradelor vârfurilor sale est3: 1,2,3,4,1

4.Fie graful G, cu n vârfuri şi m muchii, astfel încât să fie îndeplinită relaţia:

$$m > \frac{(n-1)(n-2)}{2}$$

Să se demonstreze că G nu are vârfuri izolate.

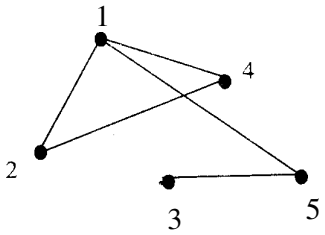
5. Fie G un graf neorientat, cu n vârfuri şi m muchii, reprezentat prin matricea de adiacenţă. Să se realizeze programe, în C/C++, care:

a) afişează gradele tuturor vârfurilor;
b) afişează vârfurile de grad par;
c) afişează vârfurile izolate;
d) afişează vârfurile terminale;
e) verifică dacă graful are vârfuri izolate;
t) verifică dacă graful are vârfuri terminale;
g) verifică dacă graful are vârfuri interioare (nu sunt nici izolate nici terminale);
h) verifică dacă graful are toate vârfurile izolate;
i) verifică dacă graful are toate vârfurile interioare (nu sunt nici izolate nici terminale):
j) afişează gradul unui vârf dat:
k) afişează vecinii unui nod dat, vf;
l) verifică dacă un vârf dat este terminal, izolat sau interior;
m) afişează gradul cel mai mare şi toate vârfurile care au acest grad
n) afişează frecventa vârfurilor:
   izolate : n 1
   terminale : n2
   interioare : n3
o) fiind dat şirul g_1, ...,g_n, să se verifice dacă poate reprezenta şirul gradelor vârfurilor în această ordine;
p) fiind dat şirul g_1 ...,g_n, să se verifice dacă poate reprezenta şirul gradelor vârfurilor (nu neapărat în această ordine).

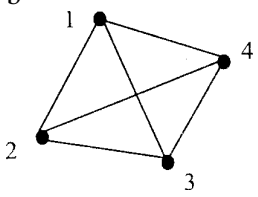
5. Graf complet

1. Fiind date grafurile de mai jos, să se precizeze care dintre ele este complet şi să se justifice răspunsul.

a



b



2. Pentru grafurile K_3 şi K_5 :

a) să se precizeze gradul fiecărui vârf;

b) să se precizeze numărul de muchii;

c) să se realizeze o reprezentare grafică.

3. Fie graful G, cu n vârfuri, dat prin matricea de adiacență. Să se realizeze un subprogram care precizează dacă graful este complet, astfel:

a) făcând o analiză asupra nodurilor;

b) făcând o analiză asupra muchiilor.

4. Fie graful G, cu n vârfuri, dat prin matricea de adiacență. Să se realizeze subprograme care precizează:

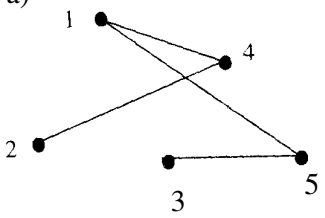
a) câte muchii mai trebuie adăugate pentru a deveni complet;

b) între ce noduri mai trebuie adăugate muchii astfel încât graful să devină complet.

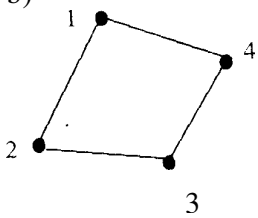
6. Graf bipartit

l. Fiind date grafurile de mai jos, să se precizeze care dintre ele este bipartit şi să se justifice răspunsul.

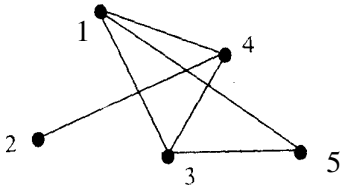
a)



b)



2. Ce muchie trebuie eliminată din graful prezentat mai jos astfel încât să devină bipartit?



3. Fie graful bipartit G, fără vârfuri izolate, dat prin matricea de adiacență. Să se realizeze un program care determină mulțimile V 1 şi V2 despre care se vorbeşte în definiție.

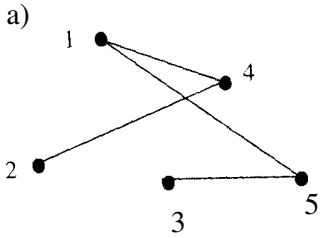
4. Fie graful G cu n vârfuri, dat prin matricea de adiacență. Să se realizeze un program care precizează dacă graful este bipartit.

5. Sa se genereze toate grafurile neorientate bipartite complete cu n noduri.

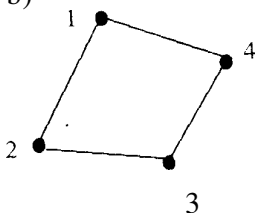
7. Graf bipartit complet

1. Fiind date grafurile de mai jos să se precizeze care dintre ele este bipartit complet şi să se justifice răspunsul.

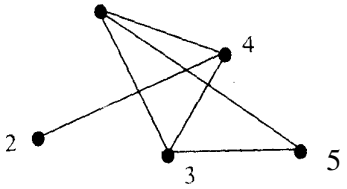
a)



b)



2. Ce muchie trebuie adăugată în graful prezentat mai jos astfel încât să devină bipartit complet:



3. Fie graful G, cu n vârfuri reprezentate {1... n}. Presupunând că graful este bipartit complet, astfel încât $|V_1| = p,\ cu\ p < n\ si\ V_1 = \{1,...,p\}$ să se construiască matricea de adiacență.

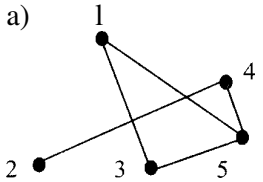
4. Fie graful G, cu n vârfuri reprezentate {1... n}. Presupunând că graful este bipartit complet şi că V 1 este formată din nodurile reprezentate prin numere pare, să se construiască matricea de adiacență.

5. Să se determine numărul total de grafuri bipartit complete cu n vârfuri date.

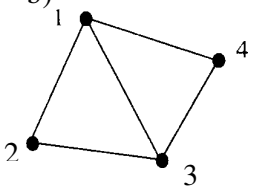
8. Reprezentarea grafurilor

l. Fiind date grafurile de mai jos

a)



b)



să se precizeze pentru fiecare in parte:

l. matricea de adiacență

2. listele de adiacență

3. şirul muchiilor

2. Fiind dată o matrice pătratică de ordin n, să se precizeze dacă poate fi considerată matricea de adiacentă a unui graf neorientat cu n vârfuri.

3. Să se determine numărul total de grafuri neorientate care au vârfurile { 1,...,n}.

4. Să se realizeze un program care generează matricele de adiacență ale tuturor grafurilor neorientate cu vârfurile { 1,...,n}.

5. Să se realizeze un program în C++ care, fiind date matricele de adiacență A1 şi A2 de dimensiune n, verifică dacă matricea A2 poate reprezenta un graf partial al grafului reprezentat de matricea A1.

6. Să se realizeze un program în C++ care să verifice dacă un graf este pentru alt graf subgraf.

7. Să se realizeze un program care generează toate grafurile bipartite complet cu n vârfuri.

8. Fie G un graf neorientat, cu n vârfuri şi cu muchiile m_1, m_2 ..., m_p. Să se realizeze un program, în C++, care determină toate grafurile parțiale ale lui G.

9. Fie G un graf neorientat, cu vârfurile 1, 2, ..., n, dat prin matricea de adiacentă. Să se realizeze un program în C++ care determină toate subgrafurile lui G.

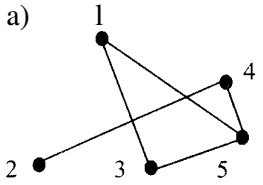
10. Fiind dat un grup de persoane, reprezentate prin numere de la 1 la n, si precizându-se relațiile de simpatie astfel: pentru fiecare persoană i, se citesc numerele de ordine ale persoanelor pe care aceasta le simpatizează (numerele se dau pe aceeaşi linie separate prin spații), să se precizeze dacă în grupul de persoane amintit, toate simpatiile sunt reciproce.

11. Să se realizeze un program care permite desenarea unui graf neorientat cu n vârfuri şi m muchii, cunoscându-se lista muchiilor.

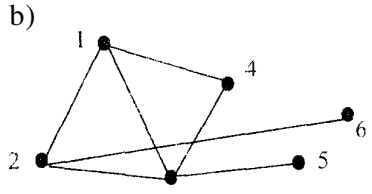
9. Parcurgerea grafurilor

1.Fiind date grafurile de mai jos:

a)



b)

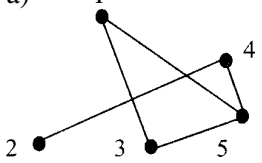


să se precizeze, pentru fiecare în parte, lista nodurilor obținută în urma parcurgerii:
- în lățime;
- în adâncime.
2. Fie G un graf, cu n noduri şi m muchii. Precizându-se un nod, de exemplu nodul l, sa se realizeze un program care afişează toate nodurile accesibile din acest nod.

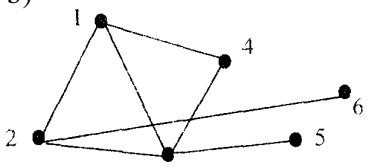
10. Conexitate

1.Fiind date grafurile de mai jos:

a)



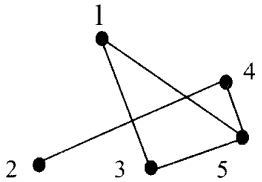
b)



sa se precizeze pentru fiecare în parte un lanț, un lanț de lungime 4, un lanț elementar, un ciclu, un ciclu elementar, două cicluri egale.
2. Fie G un graf neorientat, cu n noduri şi m muchii. Precizându-se doua noduri np(nodul de plecare) şi ns (nodul de sosire), să se determine toate lanțurile elementare care le admit ca extremități.
3. Fiind dat un graf neorientat, cu n noduri şi m muchii să se determine toate lanțurile elementare care au cea mai mare lungime.
4. Să se realizeze un program care, fiind dat un graf neorientat, verifică dacă conține un ciclu de lungime
5. Să se realizeze un program care, fiind dat un graf neorientat, afişează toate ciclurile elementare de lungime p, plecând de la nodul 1 .
6. Să se realizeze un program care, fiind dat un graf neorientat , determină câte componente conexe are.
7. Să se realizeze un program care, fiind dat un graf neorientat , determină toate componentele conexe ale sale.
8. Să se realizeze un program care, fiind dat un graf neorientat, determină toate perechile de vârfuri între care există cel puțin un lanț.
9. Speologii au cercetat n culoare subterane, pentru a stabili dacă aparțin aceleiaşi peşteri. Prin tehnici specifice de curenți de aer şi de colorare a cursurilor de apă, a fost demonstrată existenta unor canale de legătură între mai multe culoare. Precizându-se perechile de culoare între care au fost stabilite legături, să se afle dacă sistemul de culoare aparține unei singure peşteri.
10. Într-un grup de n persoane, se precizează perechi de persoane care se consideră prietene. Folosind principiul că "prietenul prietenului meu mi-este prieten", să se determine grupurile cu un număr maxim de persoane între care se pot stabili relații de prietenie, directe sau indirecte.

11. Cicluri Hamiltoniene

l. Pentru graful de mai jos; să se dea exemplu de un lanț si de un ciclu hamiltonian.



2. Să se realizeze un program care pentru un graf dat verifică dacă satisface condițiile teoremei prezentate in secțiunea 11.

3. Să se arate că numărul ciclurilor hamiltoniene ale grafului K_n cu n≥3, este $\frac{(n-1)!}{2}$

4. Să se arate că numărul ciclurilor elementare ale grafului K_n cu n≥3, este $\frac{1}{2}\sum_{k=3}^{n}\frac{n(n-1)...(n-k+1)}{k}$

5. Să se realizeze un program care pentru un graf dat verifică dacă este hamiltonian.

6. La curtea regelui Artur s-au adunat 2n cavaleri si fiecare dintre ei are printre cei prezenți cel mult n-1 dușmani. Arătați că Merlin, consilierul lui Artur, poate să-i așeze pe cavaleri, la o masă rotundă, în așa fel încât nici unul dintre ei să nu stea alături de vreun dușman de-al său.
7. Se consideră n persoane. Fiecare are printre cei prezenți cel mult n/2 dușmani. Să se determine toate posibilitățile de așezare a acestora la o masă rotundă astfel încât nici unul dintre ei să nu stea lângă un dușman al său.
8. La un cenaclu literar sunt invitați un număr de n elevi, identificații cu l...n, de la L licee, identificate 1...L. Să se determine toate modalitățile de așezare a acestora la o masă rotundă astfel încât să nu fie doi elevi de la același liceu vecini.

1. Noțiunea de graf neorientat

Problema 1

Răspunsul este afirmativ (Da). Definim graful neorientat asociat rețelei astfel:
- mulțimea nodurilor este mulțimea intersecțiilor dintre străzi și a capetelor de străzi (la ieșirea din oraș); este mulțime finită și nevidă
- mulțimea muchiilor este mulțimea bucăților de stradă dintre două intersecții sau dintre o intersecție și un capăt de stradă (ia ieșirea din oraș).

Problema 2

Răspunsul este afirmativ (Da). Definim graful neorientat asociat astfel:
- mulțimea nodurilor este mulțimea persoanelor (este mulțime finită și nevidă);
- muchia dintre nodurile x și y se definește ca fiind reprezentarea ideii "persoanele x și y se cunosc" (pot exista nenumărate definiții; dați și altele).

Problema 3

Răspunsul este afirmativ (Da). Definim graful neorientat asociat astfel:
- mulțimea nodurilor este mulțimea țărilor (este mulțime finită și nevidă):
- muchia dintre nodurile x și y se definește ca fiind reprezentarea ideii "din țara x se poate ajunge în țara y, cu avionul" (pot exista nenumărate definiții).

Problema 4

Dacă admitem că există o infinitate de stele: Răspunsul este negativ (Nu), deoarece mulțimea nodurilor trebuie să fie, conform definiției, finită (și nevidă).
Dacă admitem ca stelele sunt într-un număr finit: Răspunsul este pozitiv (Da) și putem defini graful neorientat asociat lor astfel:
- mulțimea nodurilor este mulțimea stelelor (este mulțime finită și nevidă);
- muchia între nodurile x și y se definește ca fiind reprezentarea ideii "de pe steaua x se poate vedea steaua y" (pot exista nenumărate definiții).

Problema 5

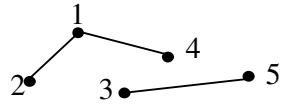
a) V={ l, 2, 3,4 ,5};
b) M={[1,2], [1,4], [1,5], [2,4], [3,5]};
c) Nodul 1 este adiacent cu nodul4; nodul 3 este adiacent cu nodul 5; ...
d) [1,2] : 1 și 2; [1,4] : 1 și 4; [1,5] : 1 și 5; [2,4] : 2 și 4; [3,5] : 3 si 5; e) [1,2] și [1,4]; [2,4] și [l,4]; ...

2. Noțiunea de graf parțial

Problema 1

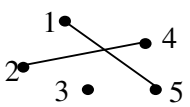
Cele două grafuri parțiale vor fi reprezentate prin desen în figurile de mai jos: Primul graf parțial (se elimină muchiile [1,5], [2,4])

V_1={ l, 2, 3, 4. 5}
M_1 ={ [1,5], [1,4] ,[3,5]}



Al doilea graf parțial (se elimină muchiile [1,2], [1,4], [3,5];)

V_1={ l, 2, 3, 4. 5}
M_1 ={ [1,5], [2,4] }



Problema 2

Grafurile parțiale, care se obțin plecând de la graful din enunț, sunt în număr de:

1 dacă se elimină 0 muchii C_3^0

3 dacă se elimină o muchiile C_3^1

3 dacă se elimină două muchii C_3^2

1 dacă se elimină toate muchiile C_3^3

deci, în total, 1+3+3+1=8 grafuri parțiale.

Problema 3

Având in vedere că: "un graf parțial al grafului număr oarecare de muchii", putem scrie:
Numărul total al grafurilor parțiale ale grafului G, este suma dintre numărul grafurilor parțiale care se obțin:

prin eliminarea unui număr de 0 muchii: C_m^0

prin eliminarea unui număr de 1 muchii: C_m^1

prin eliminarea unui număr de 2 muchii: C_m^2

.....

prin eliminarea unui număr de m-1 muchii: C_m^{m-1}

prin eliminarea unui număr de m muchii: C_m^m

prin adunare se obține: $C_m^0 + C_m^1 + C_m^2 + ... + C_m^{m-1} + C_m^m = 2^m$

Observație. Suma de mai sus se calculează astfel:

În relația $C_m^0 + C_m^1 x^1 + C_m^2 x^2 + ... + C_m^{m-1} x^{m-1} + C_m^m x^m = (1+x)^m$ se înlocuieşte x cu valoarea 1.

Problema 4

Se citesc din 2 fişiere text informații despre 2 grafuri neorinetate: pe prima linie numărul de noduri şi de pe următoarele rânduri, matricea de adiacență. Să se verifice daca graful reținut în cel de-al doilea fişier este graf parțial al grafului reținut în primul fişier.

Rezolvare:

Se verifică dacă cele 2 grafuri au acelaşi număr de noduri şi dacă graful G2 nu conține muchii care nu există în graful G1. Matricile de adicență ale celor 2 grafuri au dimensiunea n, respectiv m. Funcția grafp() verifică dacă G2 este graf parțial al grafului G1.

```
#include <conio.h>
#include <iostream.h>
#include <fstream.h>
int a1[20][20],a2[20][20],i,j,n,m;
ifstream f1("gp1.txt");
ifstream f2("gp2.txt");
int grafp()
{if(m!=n) return 0;
        else for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                if(a2[i][j]==1 &&a1[i][j]==0)
        return 0;
return 1;}
void main()
{
clrscr();
f1>>n;
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)
f1>>a1[i][j]; f1.close();
f2>>m;
for (i=1;i<=m;i++)
for (j=1;j<=m;j++)
f2>>a2[i][j]; f2.close();
if(grafp()) cout<<"este graf partial";
else cout<<"nu este graf partial";
getche();}
```

gp1.txt

```
4
0 1 0 1
1 0 1 0
0 1 0 0
```

1 0 0 0

gp2.txt
0 1 0 1

1 0 0 0
0 0 0 0
1 0 0 0

Problema 5.
Se citesc dintr-un fişier informaţiile despre graful neorintat: de pe prima linie numărul de noduri n şi eticheta unui nod x, şi apoi, de pe următoarele rânduri, matrice de adiacenţă a grafului. Să se afişeze într-un alt fişier text informaţiile despre graful paţial obţinut prin eliminarea tuturor muchiilor care au extremităţi un nod cu gradul par şi nodul x.

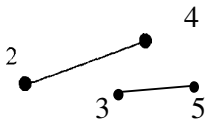
Rezolvare:
În vectorul v se reţin nodurile de grad par.

```
#include <conio.h>
#include <iostream.h>
#include <fstream.h>
int a[20][20],v[20],i,j,n,m,x;
ifstream f1("gp3.txt");
ofstream f2("gp4.txt");
int grad(int i)
{int j,g=0;
for(j=1;j<=n;j++) g=g+a[i][j];
return g;}
void graf_partial()
{int i,k=0;
for(i=1;i<=n;i++)
if(grad(i)%2==0) {k++; v[k]=i;}
for(i=1;i<=k;i++)
if(a[v[i]][x]==1) {a[v[i]][x]=0; a[x][v[i]]=0;}}
void scrie()
{int i,j;
f2<<n<<" "<<endl;
for (i=1;i<=n;i++)
for (j=1;j<i;j++)
if(a[i][j]==1) f2<<i<<" "<<j<<endl;
f2.close();}
void main()
{
f1>>n>>x;
while(f1>>i>>j)
a[i][j]=a[j][i]=1;
f1.close();
graf_partial();
scrie();}
```

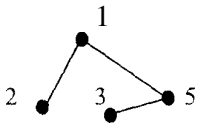
gp3.txt
7 6
1 2
1 3
1 4
2 3
2 5
3 5
3 6
5 6
6 7

gp4.txt
7
2 1
3 1
3 2
4 1
5 2
5 3
6 5
7 6

3. Noţiunea de subgraf
Problema 1
Cele două subgrafuri vor fi reprezentate prin desen în figurile de mai jos:
Primul subgraf (se elimină nodul 1 (odată cu el şi muchiile incidente [1,2], [1,4], [1,5]))



Al doilea subgraf (se elimină nodul 4 (odată cu el şi muchiile incidente [l,4], [2,4]))



Problema 2

Subgrafurile care se obțin plecând de la graful din enunț sunt în număr de:

1 dacă se elimină 0 noduri C_3^0

3 dacă se elimină 1 nod C_3^1

3 dacă se elimină două noduri C_3^2

deci, în total 1+3+3=7 subgrafuri.

Atenție! Toate nodurile nu se pot elimina pentru că s-ar obține un graf cu mulțimea vârfurilor vidă (acest lucru nu este permis de definiție).

Problema 3

Având în vedere că: un subgraf al grafului G se obține prin eliminarea unui număr oarecare de noduri (diferit de numărul total de noduri ale grafului), putem scrie:

Numărul total al subgrafurilor grafului G este suma dintre numărul subgrafurilor care se obțin:

prin eliminarea unui număr de 0 noduri: C_n^0

prin eliminarea unui număr de 1 noduri: C_n^1

prin eliminarea unui număr de 2 noduri: C_n^2

.....

prin eliminarea unui număr de n-1 noduri: C_n^{n-1}

prin adunare se obține: $C_n^0 + C_n^1 + C_n^2 + ... + C_n^{n-1} += 2^n - 1$

Problema 4

Se citesc din 2 fişiere informaţiile despre 2 grafuri neorintate: de pe prima linie numărul de noduri n şi apoi, de pe următoarele rânduri, matrice de adiacenţă a grafului. In fisierul al doilea pe ultimul rând după matricea de adiacenţă, este memorat un sir ce reprezintă etichetele acestor noduri.Să se verifice dacă graful 2 este subgraf al grafului 1.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
int n,m,i,j, a1[10][10],a2[10][10],v[10];
ifstream f1("sg1.txt");
ifstream f2("sg2.txt");
int subgraf()
{if(m>n) return 0;
else
{for (i=1;i<=m;i++) if(v[i]>n) return 0;
for (i=1;i<=m;i++)
for (j=1;j<=m;j++)
if(a2[i][j]!=a1[v[i]][v[j]]) return 0;}
return 1;}
void main()
{f1>>n;
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)  f1>>a1[i][j];
f1.close();
f2>>m;
for (i=1;i<=m;i++)
for (j=1;j<=m;j++) f2>>a2[i][j];
for (i=1;i<=m;i++) f2>>v[i];
f2.close();
if(subgraf()) cout<<"este subgraf";
else cout<<"nu este subgraf";
getch();}
```

sg1.txt

```
4
0 1 1 1
1 0 1 0
```

1 1 0 1 **0 1 1**
1 0 1 0 **1 0 1**
1 1 0
sg2.txt **1 2 3**
3

4. Gradul unui vârf

Problema 1

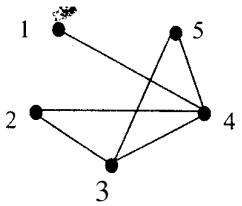
d(1)=3; d(3)=1; d(5)=2; 3 este vârf terminal;
d(2)=2; d(4)=2; d(6)=0; 6 este vârf izolat;

Problema 2

Fie V={x_1, x_2, ... x_n}. Presupunem că graful nu conține două vârfuri care să aibă acelaşi grad, deci $d(x_i)\#d(x_j)$ pentru i#j şi $d(x_i)\in\{0,1,...,n-1\}$ pentru i=1...n. În concluzie, şirul gradelor vârfurilor coincide cu {0,1,...,n-1} făcând eventual abstracție de ordine (presupunem de exemplu : $d(x_1)=0$, $d(x_2)=1$, ..., $d(x_n)=n-1$). Deci, aşa cum se vede şi în exemplul prezentat între paranteze, există un vârf care are gradul 0, adică nu este legat de nici un vârf, şi un vârf care are gradul n- 1, adică este legat de toate celelalte, deci şi de cel de gradul 0. Contradicție, deoarece cel de gradul 0 nu este legat de nici un vârf.
În concluzie, presupunerea făcută la începutul rezolvări este falsă.

Problema 3

a) Categoric nu, deoarece dacă ar exista un astfel de graf ar conține un vârf care ar avea gradul 5 (adică ar fi legat de încă 5 vârfuri). Acest lucru nu se poate întâmpla deoarece fără el în graf mai sunt decât 4 vârfuri.
b) Dacă un astfel de graf ar exista, l-am putea reprezenta astfel:



Nodul 3 a fost legat de nodul 5 (dar putea fi legat şi de nodul 1); acest lucru nu este posibil deoarece astfel acesta ar căpăta gradul 2 şi el practic îl are l.

Problema 4

Presupunem că graful are un vârf izolat şi toate celelalte noduri generează un subgraf complet (oricare două dintre ele sunt legate printr-o muchie), adică graful ar avea $\frac{(n-1)(n-2)}{2}$ muchii, oricum nu mai multe decât $\frac{(n-1)(n-2)}{2}$ aşa cum se spune în enunț. Cum situația aleasă este cea mai convenabilă în acest sens, înseamnă că un astfel de graf nu poate exista.

Problema 5

Observație. A determina gradul vârfului i, înseamnă a determina numărul elementelor care au valoarea 1 şi se găsesc pe linia i în matricea de adiacență.
Mai jos, este prezentată muchia care returnează gradul vârfului i (Funcția constă în calcularea sumei elementelor de pe linia i din matricea de adiacență).

Funcția care returnează gradul nodului i

```
int grad( int i)
{int s, j;
s=0;
for (j=1;j<=n;j++)
   s=s+aIi][j];
return s;}
```

problema a

```
#include<iostream.h>
#include<conio.h>
```

```
#include<stdio.h>
 typedef int mat[20][20];
mat a;
int       i, j, n;
int grad( int i)
{int s, j;
s=0;
for (j=1;j<=n;j++)
    s=s+aIi][j];
return s;}
void main()
{ clrscr();
cout<<"n="; cin>>n;
for (i=1;i<=n-1;i++)
    for (j=i+1;j<=n;j++)
          {cout<<"a["<<i<<",”<<j<<"]=”;                se citeşte matricea de adiacență
          cin>>a[i][j];
          a[j][i]=a[i][j];}
for (i=1;i<=n;i++)
          cout<<"varful "<<i<<" are gradul"<< grad(i)<<endl;
getche(); }
```

Problema b

```
for (i=1;i<=n;i++)
          if (grad(i) % 2==0) cout<<i<<” ”;
```

Problema c

```
for (i=1;i<=n;i++)
          if (grad(i) ==0) cout<<i<<” ”;
```

Problema d

```
for (i=1;i<=n;i++)
          if (grad(i) ==1,) cout<<i<<” ”;
```

Problema e

```
ok=0;
for (i=1;i<=n;i++)
          if (grad(i) ==0) ok=1;
if (ok) cout<<"Da";
          else cout<<"Nu";
```

Problema f

```
ok=0;
for (i=1;i<=n;i++)
          if (grad(i) =1) ok=1;
if (ok) cout<<"Da";
          else cout<<"Nu";
```

Problema g

```
ok=0;
for (i=1;i<=n;i++)
          if (grad(i) >1) ok=1;
if (ok) cout<<"Da";
          else cout<<"Nu";
```

Problema h

```
ok=1;
for (i=1;i<=n;i++)
          if (grad(i) !=0)
                    ok=0;
if (ok) cout<<"Da":
          else cout<<"Nu";
```

Problema i

```
ok=1;
for (i=1;i<=n;i++)
        if (grad(i) <=1)
                ok=0;
if (ok) cout<<"Da";
        else cout<<"Nu";
```

Problema j

```
cout<<"Dati varful”; cin>>vf;
cout<<" varful "<<vf<<" are gradul "<< grad(vf);
```

Problema k

```
cout<<"Dati varful”; cin>>vf;
for(j=1;j<=n;j++)
        if (a[vf][j]==i ) cout<<j<<” ”;
```

Problema l

```
cout<<"Dati varful : ": cin>>vf;
if (grad(vf)==0) cout<<"varful "<<vf<<" este izolat"<<endl;
        else if (grad(vf)==1) cout<<"varful "<<vf<<" este terminal";
                else cout<<"varful "<<vf<<" este interior: ";
```

Problema m

- se determină gradul cel mai mare, n gr_max (este o problemă simplă de determinare a maximului);
- se parcurg toate nodurile, cu i
    dacă gradul vârfului i este egal cu gr_max
        se afişează nodul i

```
gr_max=grad(1);
for (i=2;i<=n;i++)
        if (grad(i)>gr_max) gr_max=grad(i);
cout<<"cel mai mare grad este "<< gr_max<<endl;
cout<<"si nodurile care au acest grad sunt"<<endl;
for ( i=l ; i<=n; i++)
        if (grad(i)=gr_max) cout<<i<<” ”;
```

Problema n

```
n1=0; n2=0; n3=0;
 for (i=1;i<=n;i++)
if (grad(i)=0)  n1=n1+1;
        else if (grad(i)==1) n2=n2+1;
                else n3=n3+1;
cout<<"In graful dat sunt : "<<endl;
cout<<"         "<<nl<<" varfuri izolate"<<endl;
cout<<"         "<<n2<<" varfuri terminale"<<endl;
cout<<"         <<n3<<" varfuri interioare"<<endl;
```

Problema o

```
for (i=1;i<=n;i++) cin>>g[i];
ok=1;
for (i=1;i<=n;i++)
        if (grad(i)!=g[i]) ok=0;
cout<<ok;
```

Problema p

- se citeşte şirul g;
- se construieşte şirul gradelor vârfurilor, gr;
- se sortează crescător cele două şiruri, folosind funcţia:

```
void sort_crescator(sir x, int n)
{int i, ok, aux;
  do{
```

```
      ok=1;
      for (i=1;i<=n-l;i++)
         if (x[i]>x[i+l])
                {aux= x[i];
                x[i]=x[i+1];
                x[i+1] =aux;
                ok=0; }
  }while (ok==0);
}
```

- se verifică dacă sunt identice:

```
for (i=1;i<=n;i++) cin>>g[i];
for (i=1;i<:=n;i++) gr[i]=grad(i);
sort_crescator(g,n);
sort_crescator(gr,n);
ok=1;
for (i =1;i<=n;i++)
if (gr[i]!=g[i]) ok=0;
cout<<ok;
```

5. Graf complet

Problema 1

a. Graful nu este complet, deoarece există două noduri între care nu există muchie (1 si 3).

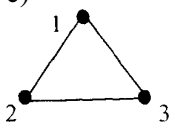
b. Graful este complet, deoarece oricare ar fi două vârfuri distincte exista o muchie care le unește.

Problema 2

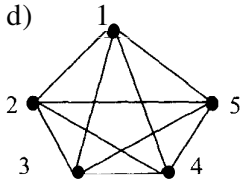
a)K_3 : d(x)=3-1=2 $\forall x \in V$; K_5 : d(x)=5-1=4 $\forall x \in V$;

b) K_3: m=3(3-1)/2=3; K_5: m=5(5-1)/2=10;

c)



d)



Problema 3

Problema a

Observație. A verifica că un graf este complet, făcând o analiză asupra vârfurilor, înseamnă a verifica dacă toate vârfurile au gradul n-1.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int mat[20][20];
mat a;
int      i,j,n,c;
int grad( int i)
{int s,j;
s=0;
for (j=1;j<=n;j++) s=s+a[i][j];
return s;}
void complet(int &c)
{int i;
c=1;
for (i=1;i<=n;i++)
        if (grad(i)!=n-1) c=0;
}
void main()
{ clrscr();
cout<<"n="; cin>>n;
for (i=1;i<=n-1;i++)
for (j=i+1 ;j<=n;j++)
```

```
{cout<<"a["<<i<<","<<j<<"]=";
cin>>a[i][j];
a[j][i]=a[i][j];}
complet(c);
if(c==1) cout<<"graful este complet";
else cout<<"graful nu este complet";
getche(); }
```

Problema b

Observație. A verifica că un graf este complet, făcând o analiză asupra muchiilor, înseamnă a verifica dacă are n(n-1)/2 muchii, altfel spus: trebuie verificat dacă toate elementele de deasupra diagonalei principale din matricea de adiacență sunt egale cu 1.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int mat[20][20];
mat a;
int       i, j, n;
void complet()
{int i,j,nr=0;
for (i=1;i<=n-1;i++)
   for (j=i+1 ;j<=n;j++)
        if (a[i][j]==1)  nr++;
if(nr==(n*(n-1)/2)) cout<<"graf complet";
                  else cout<<"graful nu este
        complet";}
void main()
{ clrscr();
cout<<"n="; cin>>n;
for (i=1;i<=n-1;i++)
   for (j=i+1;j<=n;j++)
   {cout<<"a["<<i<<","<<j<<"]=";
   cin>>a[i][j];
   a[j][i]=a[i][j];}
complet();
getche();
}
```

6. Graf bipartit

Problema 1

a) Graful este bipartit, deoarece respectă definiția; V1={ 1, 2, 3}, V2={4, 5}.
b) Graful este bipartit, deoarece respectă definiția; Vl={ 1, 3}, V2= { 2, 4}.

Problema 2

Pentru a obține un graf bipartit, trebuie eliminată muchia [1,3]; V 1={ 1, 2, 3 } , V2={4, 5}.

Problema 3

Fie graful bipartit G, fără vârfuri izolate, dat prin matricea de adiacență. Să se realizeze un program care determină mulțimile V 1 şi V2 despre care se vorbeşte în definiție.

Rezolvare:

- la început V1=[ ] şi V2=[ ]
- se parcurge matricea de adiacentă, linie cu linie, deasupra diagonalei principale
dacă pe linia i se găseşte elementul a[i,j]=1,
atunci
dacă i aparține deja lui V2 atunci
j se adaugă la mulțimea V1: V1:=Vl+1j]
altfel
i se adaugă la mulțimea V1: V 1:=V 1+[i]
j se adaugă la mulțimea V2: V2:=V2+[j]

Soluția este de a genera într-un vector nodurile care aparțin mulțimilor V1, V2, astfel: dacă un element are valoare a 1, nodul care eticheta corspunzătoare indicelui elementului mulțimii V1; altfel, aparține mulțimii V2.

Funcția gererare() generează grafurile bipartite complete. În vectorul a se generează nodurile mulțimilor V1 şi V2. Inițial elementele vectorului auvaloarea 0. Variabila posibil se foloseşte pentru a verifica dacă mai există posibilitățile de generare de submulțimi (are valoare 1 atunci când mai este posibilsă se genereze submulțimile)

```
#include <conio.h>
#include <iostream.h>
#include<math.h>
void generare (int n)
{int a[20]={0},i,j,k=0,posibil=1;
while (posibil)
{j=n;
while(j>0&&a[j]==1) {a[j]=0;j--;}
if(j==0) posibil=0;
else
{a[j]=1; k++;
if(k<=pow(2,n)-2)
```

```
{cout<<"graful "<<k<<endl<< "Multimea V1:";
for(i=1;i<=n;i++) if(a[i]) cout<<i<<" ";
cout<<"Multimea V2:";
for(i=1;i<=n;i++) if(!a[i]) cout<<i<<" ";
cout<<endl;
cout<<"Muchiile sunt:";
for(i=1;i<=n;i++)
if(a[i]==1)
for(j=1;j<=n;j++)
if(a[j]==0&&i!=j)  cout<<"["<<i<<","<<j<<"] ";
cout<<endl;}}}}

void main()
{ int n;
cout<<"numar de noduri="; cin>>n;
generare(n);
getch();}
```

Problema 4

4. Fie graful G cu n vârfuri, dat prin matricea de adiacență, într-un fişier text. Să se realizeze un program care precizează dacă graful este bipartit.

Rezolvare: *Pt. a verifica dacă graful este bipartit, se generează mulțimile de noduri V1 şi V2 până când aceste mulțimi îndeplinesc condiția unui graf bipartit, sau până când s-au generat toate mulțimile şi nici na dintre variante nu a îndeplinit condiția pt. graful bipartit (între orice pereche de 2 elemente din cele 2 mulțimi există o muchie care să le lege în graf)*

Se generează într-un vector toate submulțimile care se pot obține din cele n etichete de noduri (exceptând mulțimea inițială şi cea vidă) şi se verifică dacă nodurile aparținând celor 2 mulțimi generate pot fi mulțimile unui graf bipartit.

Funcția bipartit() verifică dacă graful este bipartit furnizând un rezultat logic. Elementele vectorului x în care se generează mulțimile V1 şi V2 sunt inițial 0. Variabila gasit se foloseşte pentru a verifica dacă s-au găsit cele două mulțimi de noduri corespunzătoare unui graf bipartit (are valoarea 1 atunci când s-au găsit)

```
#include <conio.h>
#include <iostream.h>
#include <fstream.h>
#include<math.h>
int a[20][20],v[20],i,j,n,m,x;
ifstream f("gb.txt");
int bipartit()
{int x[10]={0},i,j,m,k=0,posibil=1,gasit=0;
while(posibil&&!gasit)
{m=n;
while(m>0&&x[m]==1) {x[m]=0;m++;}
if(m==0) posibil=0;
else
{x[m]=1; k++;
if(k<=pow(2,n)-2)
for(i=1,gasit=1;i<=n&&gasit;i++)
for(j=1;j<=n&&gasit;j++)
if(a[i][j]==1)
if((x[i]==1&&x[j]==1)||(x[i]==0&&x[j]==0))
gasit=0;}}
return gasit;}
void main()
{
f>>n;
while(f>>i>>j)
a[i][j]=a[j][i]=1;
f.close();
if (bipartit()) cout<<"este bipartit";
else cout<<"nu este bipartit";
getch();}
```

gb.txt

```
4
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
```

Problema 5.

Să se gereze toate grafurile neorientate bipartite complete cu n noduri.

Rezolvare: *problema se reduce la a genera toate submulțimile care se pot obține din cele n elemente. Numărul total de submulțini obținut este 2^2-2 (mai puțin mutimea inițială şi mulțimea vidă). Soluția este de a genera într-un vector nodurile care aparțin mutimilor V1 şi V2, astfel: dacă un elemnt are valaorea 1, nodul care are eticheta corespunzătoare indicelui elementului aparține mulțimii V1; altfel apaține mulțimii V2.*

Funcția generare() generează nodurile mulțimilor V1 şi V2. Inițial elementele vectorului au valaorea 0. Variabila posibil se foloseşte pentru a verifica dacă mai există posibilități de generare de submulțimi(are valorea 1 când e posibil să se mai genereze submulțimi)

```
include<iostream.h>
#include<conio.h>
#include<math.h>
```

```
void generare (int n)
{int a[10]={0},i,j,k=0,posibil=1;
while(posibil)
        {j=n;
        while(j>0&&a[j]==1) {a[j]=0;j--;}
        if(j==0) posibil=0;
              else{a[j]=1; k++;
        if(k<=pow(2,n)-2)
                {cout<<"graful"<<k<<endl<<"multimea V1: ";
                for(i=1;i<=n;i++) if(a[i]) cout<<i<<" ";
                cout<<endl;
                cout<<"multimea V2:";
                for(i=1;i<=n;i++) if(!a[i]) cout<<i<<" ";
                cout<<endl;
                cout<<"muchiile sunt: ";
                for(i=1;i<=n;i++)
                    if(a[i]==1)
                    for(j=1;j<=n;j++)
                            if(a[j]==0&&i!=j) cout<<"["<<i<<","<<j<<"]";
                cout<<endl;}}}}
void main()
{int n;
cout<<"numarul de noduri"; cin>>n;
generare(n);getch();}
```

7. Graf bipartit complet

Problema 1

a) Graful **este bipartit**, deoarece respectă definiția, V1={ 1, 2, 3 }, V2={4, 5}, dar **nu este complet**, deoarece există noduri in V1 (ex. 2) nelegate de toate nodurile din V2 (ex. 5).

b) Graful **este bipartit** deoarece respectă definiția, V1={ 1, 3}, V2={2, 4}, dar cum toate nodurile din V 1 sunt legate de toate nodurile din V2 înseamnă că **este bipartit complet**.

Problema 2

Pentru a obține un graf bipartit complet trebuie adăugată muchia [2,5]; V 1={ 1, 2, 3 } , V2={4, 5}.

Problema 3

Se procedează astfel:

Pe liniile l..p din matricea de adiacență

se pune valoarea 1 pe coloanele p+l..n, deoarece toate elementele din V 1 sunt legate de toate elementele din V2

(nu trebuie uitat ca matricea de adiacență este simetrică față de diagonala principală)

```
.....
for (i=1;i<=p;i++)
for (j=p+l;j<=n:j++)
                { a[i][j]=1;
                a[j][i]=1;}
.....
```

Problema 4

Se procedează astfel:

Este suficient să gândim construirea matricei deasupra diagonalei principale, pentru că ea este simetrică față de diagonala principală. Matricea se construieşte astfel:

- **pe liniile pare**, de deasupra diagonalei principale,

se pune valoarea 1 pe coloanele impare, deoarece toate elementele din V 1 sunt legate de toate elementele din V2

- **pe liniile impare**, de deasupra diagonalei principale, se pune valoarea 1 pe coloanele pare, deoarece toate elementele din V2 sunt legate de toate elementele din V 1

(nu trebuie uitat ca matricea de adiacență este simetrică față de diagonala principală)

.....

```
for (i=1;i<=n-l;i++)
    for (j=i+l;j<=n;j++)
        if ((i % 2==0) && (j % 2!=0})
                { a[i][j]=1;
                a[j][i]=1;}
for (i=1;i<=n-l;i++)
    for (j=i+1;j<n;j++)
        if ((i % 2!=0) && (j % 2==0))
                { a[i][j]=1;
                a[j][i]=1;}
```

.....

Problema 5

Un graf bipartit complet este unic determinat de o partiție a lui V n doua submulțimi V l şi V2, disjuncte şi nevide. A determina toate grafurile bipartit complete, înseamnă a determina în câte moduri se pot construi mulțimile V1 şi V2. Pentru aceasta procedăm astfel:

În mulțimea V1 se pune **nodul 1**, pentru a nu repeta soluțiile
(mai sunt n-1 noduri nepuse).

Partiția_1 la V1 se adaugă 0 noduri (sunt C_{n-1}^{0} situații) iar la V2 restul

Partiția_2 la V1 se adaugă 1 noduri (sunt C_{n-1}^{1} situații) iar la V2 restul

Partiția_3 la V1 se adaugă 2 noduri (sunt C_{n-1}^{2} situații) iar la V2 restul

.....

Partiția_n-1 la V1 se adaugă n-2 noduri (sunt C_{n-1}^{n-2} situații) iar la V2 restul

(alte partiție nu mai există, pentru că la Vl nu se pot adăuga încă n-1 noduri deoarece V2 ar fi vidă, în acest caz , şi ar fi în contradicție cu definiția grafului bipartit).

În total sunt $C_{n-1}^{0}+C_{n-1}^{1}+C_{n-1}^{2}+...+C_{n-1}^{n-2}$ posibilități de construire. Această sumă se calculează astfel:

$C_{n-1}^{0}+C_{n-1}^{1}+C_{n-1}^{2}+...+C_{n-1}^{n-2}+C_{n-1}^{n-1}=2^{n-1}$

de unde rezultă ca:

$C_{n-1}^{0}+C_{n-1}^{1}+C_{n-1}^{2}+...+C_{n-1}^{n-2}=2^{n-1}-C_{n-1}^{n-1}=2^{n-1}-1$

8. Reprezentarea grafurilor

Problema 1

a)

$$A=\begin{pmatrix}0&0&1&0&1\\0&0&0&1&0\\1&0&0&0&1\\0&1&0&0&0\\1&0&1&0&0\end{pmatrix}$$

Vârful i	Lista vecinilor lui
1	3,5
2	4
3	l, 5
4	2
5	1, 3

M={[e1[1],e2[1]]=[1,3],[e1[2],[2]]=
[1,5],[e1[3],e2[3]]=[2,4],[e1[4],e2[4]]=[3,5]}

b)

$$A=\begin{pmatrix}0&1&1&1\\1&0&1&0\\1&1&0&1\\1&0&1&0\end{pmatrix}$$

Vârful i	Lista vecinilor lui
1	2,3,4
2	1,3
3	l, 2,4
4	1,3

M={[el[1],e2[1]]=[1,2], [el[2],e2[2]]=[1,3],
[el[3],e2[3]]=[1,4], [el[4],e2[4]]
=[2,3],[el[5],e2[5]]=[3,4]}

Problema 2

A verifica dacă matricea poate reprezenta matricea de adiacență a unui graf neorientat, trebuie verificate următoarele:

- matricea are pe diagonala principală numai elemente egale cu 0;
- matricea are deasupra diagonalei principale numai elemente de 0 şi 1;

- matricea este simetrică față de diagonala principală.

```
.....
ok1=1;
for (i=l;i<=n;i++)
        if (a[i][i]!=0) ok1=0;
ok2=1;
for (i=1;i<=n-l;i++)
    for (j=i+l;j<=n;j++)
        if (!((a[i][j]==a) || (a[i][j]=1)) ok2=0;
ok3=1;
for (i=1;i<=n-l;i++)
    for (j=i+l ;j<=n;j++)
        if (a[i][j] !=a[j][i]) ok3=0;
ok=okl && ok2 && ok3;
cout<<ok;
.....
```

Problema 3

Un graf neorientat, cu n vârfuri, este unic determinat de o matrice de adiacență pătratică de ordinul n. Deci, a determina grafurile neorientate cu n vârfuri înseamnă a determina toate matricele pătratice de ordinul n, caracterizate astfel:
- au numai elemente de 0 şi/sau 1;
- pe diagonala principală an numai elemente de 0;
- sunt simetrice față de diagonala principală.

Citind cu atenție caracteristicile matricei de adiacență, constatăm cu uşurință că pentru a determina o astfel de matrice este suficient să-i determinăm elementele de deasupra diagonalei principale, deoarece matricea are deasupra diagonalei principale $(n^2-n)/2$ elemente, problema se reduce la a determina toți vectori de lungime $(n^2-n)/2$ cu elemente de 0 şi 1; aceşti vectori sunt în număr de $2^{(n^2-n)/2}$ (este vorba de determinarea submulțimilor unei mulțimi cu $(n^2-n)/2$ elemente).

Problema 4

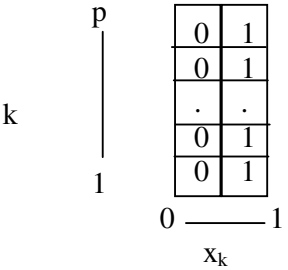
Problema se reduce la a determina toți vectori de lungime $(n^2-n)/2$ cu elemente de 0 şi/sau 1; (vezi explicațiile de la problema anterioară). Acest lucru se face folosind metoda Backtracking, astfel:
- se generează pe rând toți vectorii de lungime $p=(n^2-n)/2$ cu elemente de 0 şi/sau 1;
    - pentru fiecare vector generat, se construieşte matricea de adiacență corespunzătoare, astfel:
        - pe diagonala principală se pune 0;
        - deasupra diagonalei principale se pun elementele vectorului, astfel:

```
ks=1;
for (i=1;i<=n-l;i++)
    for (j=i+l ;j<=n;j++)
        {a[i][j]=x[ks];
        ks=ks+1;
        a[j][i]=a[i][j];}
```

        - elementele de sub diagonala principală trebuie puse astfel încât matricea să fie simetrică față de diagonala principală.

În concluzie, trebuie generate elementele mulțimii $\{(x_1,x_2,...,x_p) \mid x_k \in \{0,1\}, k=1,...,p\}$.
$x=(x_1,x_2,...,x_p)$ unde $x_k \in \{0,1\}$; $k \in \{1,...,p\}$; $p=(n^2-n)/2$

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir[100];
sir x;
int i, k, n, p;
int as, ev;
int a[100][1001;
void succ(sir x, int k, int &as)
{if (x[k]<I)
```



```
        {as=1;
        x[k]=x[k]+l;}
    else as=0;}
void valid( int &ev)
{ev=1;}
void afis(sir x)
{int i, j, ks;                          generarea matricei de adiacență,
ks=1;                                   plecând de la vectorul generat
for (i=1;i<=n-1;i++)
    for (j=i+l;j<=n;j++)
                {a[i][j]=x[ks];
                ks=ks+1;
                a[j][i]=a [i][j];}
for (i = 1 ;i<=n;i++)                   afişarea matricei de adiacență
    {for (j=1;j<=n;j++)
         cout<<a[i]]j]<<" ";
    cout<<endl;}
cout<<endl<<endl;}
void main()
{cout<<"n=";cin>>n;
k=1;
x[k]=1;                                 priviți tabla
while (k>0) {
                do{ succ(x,k,as);
                    if (as) valid(ev);
                    } while (as && !ev);
                if (as)
                    if (k=p) afis(x);
                        else { k=k+ 1;
                               x[k]=1;}
                    else k=k-1;
                }
}
```

Problema 5

Matricea de adiacență A2 reprezintă un graf parțial al grafului reprezentat de matricea de adiacență A1 dacă acolo unde elementele matricei A1 sunt 0 şi elementele corespunzătoare din matricea A2 sunt 0; în rest nu contează, adică, dacă elementele lui A1 sunt 1 elementele corespunzătoare din A2 pot fi 0 sau 1 (acest lucru se exprimă astfel: A2[i,j]<=A1[i,j]). Este suficient să facem verificarea pentru elementul de deasupra diagonalei principale.

```
.................
ok=1;
for (i=l;i<=n-1;i++}
    for (j=i+l;j<=n;j++)
       if (!(A2[i][j]<=A1[i][j]) ok=0;
if (ok) cout<<"A2 reprezinta un graf partial al grafului reprezentat de A1";
     else cout<<"A2 nu reprezinta un graf partial al grafului reprezentat de A1 ";
.....................
```

Problema 7

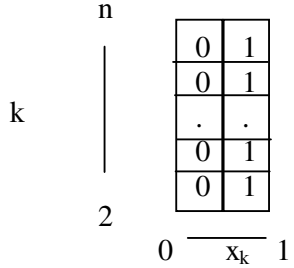
Un graf bipartit complet este unic determinat de o partiție a lui V în două submulțimi V1 şi V2, distincte şi nevide. A determina toate grafurile bipartit complete, înseamnă a determina toate modurile în care se pot construi mulțimile V1 si V2, din elementele {1 ...n}.

Acest lucru se realizează astfel:

În mulțimea V1 se pune nodul 1, pentru a nu repeta soluțiile (mai sunt n-1 noduri nepuse). Problema revine la a determina toate posibilitățile de a plasa vârfurile 2...n în prima sau în a doua mulțime; orice astfel de plasare convine cu excepția plasării tuturor vârfurilor în prima mulțime.

Pentru a rezolva această problemă vom folosi metoda Backtracking, astfel:
Se generează toţi vectorii:
$x=(x_1x_2,...,x_n)$ cu $x_1=1$ şi $x_k \in \{0,1\}$ pentru $k \in \{2,...,n\}$, fără ca toate elementele să fie egale cu 1, cu următoarea semnificaţie:

dacă $x_i=1$, atunci i face parte din V1
altfel i face parte din V2




```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir[ 100];
sir x;
int i, k, n; int   as, ev;
void succ(sir x, int k, int&as)
{if (x[k]<1)
            {as=1;
            x[k]=x[k]+1;}
   else as=0;}
void valid( int &ev )
{ev=1;}
void afis(sir x, int k)
{int i, s;
s=0;
for (i=l;i<=k;i++)
        s=s+x[i];
if (s!=n)
        {cout<<"V 1:";
        for (i=1;i<=n;i++)
                if (x[i]==1) cout<<i<<" ";
        cout<<endl;
        cout<<"V2 :";
        for (i=1;i<=n;i++)
                if (x[i]==0) cout<<i<<" ";
        cout<<endl<<endl;}
}
void main()
{ clrscr();
cout<<"n="; cin>>n;
x[1]=1;
k=2;
x[k]=-1;        priviţi tabla
while (k> 1){
                do{ succ(x,k,as);
                    if (as)
                    valid(ev);
                    }while (as && !ev);
                if (as)
                      if (k==n) afis(x,k);
                          else
                              {k=k+ 1;
                              x[k]=-1;}
                  else k=k-1;
                }
getche();}
```

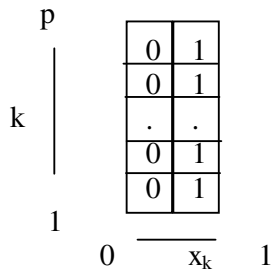
dacă toate componentele se verifică au valoarea 1

din V1 fac parte numai elementele i pentru care x_i =l

din V2 fac parte numai elementele i pentru care x_i =0

Problema 8

După cum se vede în enunțul problemei, graful se dă prin precizarea numărului de noduri şi a şirului muchiilor sale m_1 ... m_p. Plecând de la faptul că un graf parțial al său se obține prin eliminarea unui număr oarecare de muchii, păstrând aceeaşi muchie de vârfuri, problema dată se reduce la generarea submulțimilor mulțimii $\{m_1 ... m_p\}$. Acest lucru îl vom realiza folosind metoda Backtracking, astfel:
- se generează submulțimile mulțimii $\{m_1 ... m_p\}$
- pentru fiecare astfel de submulțime generată, construim matricea de adiacență si o afişăm.
Se generează toți vectorii:
$x=(x_1,x_2,...,x_p)$ cu $x_k \in \{0,1\}$ pentru $k \in \{1,...,p\}$ cu următoarea semnificație: dacă $x_i=0$, atunci m_i se elimină din mulțimea $\{m_1 ... m_p\}$ altfel m_i nu se elimină din mulțimea $\{m_1 ... m_p\}$

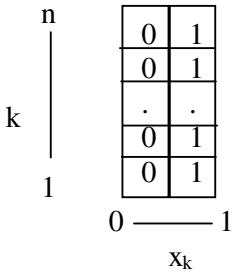



```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir[100];
typedef struct{
        int x, y; } muchie;
sir x;
int a[100][100];
int i, k, n, p, nr, j;
int as, ev;
muchie m[100];
void succ(sir x, int k, int &as)
{if (x[k]<l)
        {as= 1;
        x[k]=x[k]+1;}
void valid(int &ev)
{ ev=1;}
void afis(sir x, int k)
{ int i;
for (i=1;i<=k;i++)
        if (x[i]==1)
        {a[m[i].x][m[i].y]=1;
        a[m[i].y][m[i].x]=1;}          construirea matricei de adiacentă
nr=nr+ 1 ;
cout<<"matricea de adiacenta al celui de-al "<<nr<<"-lea graf partial"<<endl;
for (i=1;i<=n;i++)                    afişarea matricei de adiacență
    {for (j=1;j<=n;j++)
        cout<<a[i][j]<<" ";
    cout<<endl;}
}
 void main()
{ clrscr();
nr=0;
cout<<"n=", cin>>n;
cout<<"p="; cin>>p;
for (i=1;i<=p;i ++) cin>>m[i].x>>m[i].y;
k=1;
x[k]=- 1;
while (k>0){                          priviti tabla
                do {succ(x,k,as);
                if (as) valid(ev);
                }while (as &&!ev);
                if (as)
                        if (k==p) afis(x,k);
                           else
                           {k=k+ 1;
```

```
                    x[k]=-1;}
                else k=k-1;
            }
getch();}
```

Problema 9

Ținând cont de faptul ca un subgraf al unui graf se obține prin eliminarea unui număr de noduri, şi a muchiilor incidente cu aceste noduri, tragem următoarea concluzie:
Problema se reduce la a genera toate submulțimile mulțimii {l, ..., n} (fără mulțimea vidă), iar pentru fiecare submulțime generată se afişează decât acele muchii ale grafului care au ambele extremități printre elementele submulțimii.
Rezolvarea problemei se face folosind metoda Backtracking, şi constă în a genera elementele mulțimii:
$\{(x_1,x_2...,x_n)\ I\ x_k \in \{0,1\}, k=1,...,n$, nu toate nule$\}$.
(dacă x_k =1, înseamnă că nodul k face parte din submulțime, altfel nu)
$x=(x_1,x_2,...,x_n)$ unde $x\in \{0,1\}$;
$k\in \{1,...,n)$;




```
#include <conio.h>
#include <iostream.h>
 #include <stdio.h>
 typedef int     sir [ 100];
typedef int mat[20[20];
int       e l, e2;
int i,j,n,m,ns,k; mat a;
sir x;
int as, ev;
void succ(sir x, int k, int &as)
{if (x[k]< 1)
            {as= 1;
            x[k]=x[k]+1;}
   else as=0;}
void afis(sir x, int k)
{ int i;
ns=ns+1;                                          ns este numărul soluției
cout<<"subgraful cu numarul "<< ns<<"are nodurile ";      curente
cout<<endl;
for (i=1;i<=k;i++)
        if (x[i]==1)                              se afişează nodurile
                cout<<i<<" ",                     corespunzătoare unei soluții
cout<<endl;
cout<<"si muchiile ";
for (i=1;i<=k-l;i++)    se afişează muchiile
   for (j=i+l;j<=k;j++)         corespunzătoare unei soluții
      if ((x[i]==1) && (x[j]==1) && (a[i][j]==1))
            cout<<" (,<<i<<","<<j<<")"<<endl;
cout<<endl<<endl;}
void valid(sir x, int k, int &ev)
{ int i, s;
ev=l;
if (k=n)
        { s=0;                    aici se elimină soluția
        for (i=1;i<=k;i++)        cu toate componentele
                s=s+x[i];         egale cu 0
        if (s==0) ev=0;}
}
void main()
```

```
{ clrscr();
cout<<"n="; cin>>n;
cout<<"m=”; cin>>m;
for (i=1;i<=m;i++)
        {cout<<"el e2 "; cin>>el>>e2;
        a[el][e2]=1;
        a[e2][e1]=1;}
ns=0;
x[1]=-1;
k=1;
while (k>0){
            do{
               succ(x,k,as);
               if (as) valid(x,k,ev);
               }while (as && !ev);
            if (as)
               if (k==n) afis(x,k);
                        else {
                                k=k+1 ;
                                x[k]=- 1;}
              else k=k-1,
            }
getch();}
```

Problema 10

Problema se reduce la:
- a ne imagina un graf neorientat, nodurile grafului sunt persoanele iar muchiile sunt reprezentarea relațiilor de simpatie între persoane, care este reprezentat prin listele de adiacență (citite de la tastatură)
- a construi matricea de adiacență corespunzătoare (plecând de la listele de adiacență);
- a verifica proprietatea de simetrie fată de diagonala principală, adică: a[i][j]=a[j][i] , pentru $1 \leq i, j \leq n$.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int mat[20][20];
int i, j, n, vec;
mat a;
int ok;
void main()
{ clrscr(); c
out<<"n="; cin>>n;
for (i=1;i<=n;i++)                          se construieşte matricea de adiacență plecând de la listele
        {cout<<"Pentru "<<i<<endl;                  de adiacenta
        cout<<"dati numarul vecinilor";
        cin>>vec:
        for (k=l;k<=vec;k++)
                {cin>>j;
                a[i][j]=1;}
        }
ok=1;  se verifică proprietatea de simetrie
for (i=1;i<=n-l ;i++)
   for (j=i+l;j<=n;j++)
        if (a[i][j]!=a[j][i])
                {ok =0;
```

```
                cout<<"nepotrivire intre "<<i<<" si"<<j;}
if (ok) cout<<"toate simpatiile sunt reciproce",
getche(); }
```

9. Parcurgerea grafurilor

Problema 1

Listele nodurilor obținute în urma parcurgerii in lățime:

a) 1, 3, 5, 4, 2;

b) l, 2, 3, 4, 6, 5;

Listele nodurilor obținute în urma parcurgerii în adâncime:

a) 1, 3, 5, 4, 2;

b) 1, 2, 3, 4, 5, 6;

Problema 2

Această problemă este, până la urmă, o problemă de parcurgere a unui graf. Pentru a determina nodurile care sunt accesibile din nodul 1, se procedează astfel:

- la început, nici un nod nu se consideră vizitat;
- se vizitează nodul 1;
- se aplică una dintre procedurile de parcurgere, de exemplu în adâncime, plecând de la nodul 1;
- se afişează toate nodurile care au fost vizitate, în urma apelului procedurii de care am vorbit (aceste noduri sunt, de fapt, nodurile la care se poate ajunge plecând de la nodul 1).

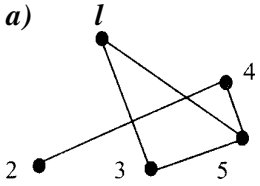
```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir [100î;
typedef int mat[20][20];
int i, n, pl, m, x, y;
mat a;
sir       viz;
void parc_adancime(int pl)
{ int j;
viz[pl]=l;
for (j=1;j<=n;j++)
        if ((a[pl][j]==1) && (viz[j]==0))
                parc_adancime(j);}
void main()
{ clrscr();
cout<<"n        m"; cin>>n>>m;
for (i= l;i<=m;i++)
        {cout<<"x y "; cin>>x>>y;
        a[x][y]=1;
        a[y][x]=1;}
for (i=1;i<=n;i++)
        viz[i]=0;
p1=1;
parc adancime(pl);
cout<<" nodurile care sunt accesibile din nodul 1 sunt ";
for (i=1;i<=n;i++)
        if (viz[i]==1) cout<<i<<" ";
getche(); }
```

10. Conexitate

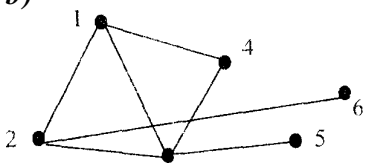
Problema1

Fiind date grafurile de mai jos:

a) ***l***



b)



sa se precizeze pentru fiecare în parte un ***lanț,*** un ***lanț de lungime 4,*** un ***lanț elementar,*** un ***ciclu,*** un ***ciclu elementar, două cicluri egale.***

a) L1=[1, 3, 5] lanț;
L2=[1, 3, 5, 4, 2] lanț de lungime 4;
Oricare dintre lanțurile de mai sus sunt elementare.
C1=[1, 3, 5, 1] ciclu;
Ciclul de mai sus este elementar.
C1=[l, 3, 5, 1] este egal cu C2=[3, 5, 1, 3];

b)L1=[l, 2, 3, l, 4] lanț;
L2=[4, 1, 2, 3, S] lanț de lungime 4;
Lanțul de mai sus este elementar.
Cl=[1, 2, 3, 1] ciclu;
Ciclul de mai sus este elementar.
C1=[l, 2, 3, 4, 1] este egal cu C2=[4, 3, 2, l, 4];

Problema 2

Fie G un graf neorientat, cu n noduri şi m muchii. Precizându-se doua noduri np (nodul de plecare) şi ns (nodul de sosire), să se determine toate lanțurile elementare care le admit ca extremități.

Problema se rezolvă folosind metoda Backtracking, şi se reduce la a determina şirurile de forma x_1..... x_p unde x_l=np şi x_p=ns (deci, construcția unei soluți se opreşte atunci când o componentă a sa primeşte valoarea ns), cu componente distincte, astfel încât între x_i şi x_{i+1} să existe muchie în graful dat.
În concluzie, soluția este x=(x_1x_2,...,x_p), unde x_1=np, x_p=ns, $x_k \in \{1,...,n\}$ $k \in \{2....,p-1\}$; $x_i \neq x_j$ pentru i≠j, şi a[x_{i-1}][x_i]=1
Comentarii la functia Valid:
Trebuie verificat ca:
- există muchie între nodurile x_{k-1} şi x_k, adică trebuie verificat că a[x_{k-1}][x_k]=1;
- nodul x_k este diferit de toate nodurile prin care s-a trecut, adică:
xk≠x_i pentru i=1...k-1.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir [ 100];
typedef int mat[20][20];
int       e l, e2;
int i,j,n,m,ns,np,k;
mat a;
sir x;
int as, ev;
void succ(sir x, int k, int &as)
{if(x[k]<n)
        { as=1;
        x[k]= x[k]+1;}
   else as=0;}
void afis(sir x, int k)
{int i;
for (i=1;i<=k;i++)
        cout<<x[i]<c" ";
cout<<endl;}
void valid(sir x, int k, int &ev)
```

```
{ int i;
ev=1;
if (a[x[k-1]][x[k]]==0) ev=0;                    trebuie să existe muchie între
   else                                          x_{k-1} si x_k
        for (i=1;i<=k-l;i++)
              if (x[k]==x[i]) ev=0;}
void main()
{ clrscr();
cout<<"n=": ciu>>n;
cout<<"m=”; cin>>m;
for ( i=1;i<=m;i++)
       {cout<<"el       e2 "; cin>>el>>e2;
       a[e1][e2]=1;
       a[e2] [e1]=1;}
cout<<"np="; cin>>np;
cout<<"ns="; cin>>ns;
x[1]=np;
k=2;
x[k]=0;
while (k>1){
              do{
                   succ(x,k,as);
                   if (as) valid(x,k,ev);
                   }while (as && !ev);
              if (as)
                   { if (k<=n)
                      if (x[k]=ns)
                            afis(x,k);
                        else
                            {k=k+ 1;
                            x[k]=0;}
                   }
                 else k=k-1;
              }
getche(); }
```

Problema 3

Problema se rezolvă folosind metoda Backtracking, astfel:

- se generează aranjamentele muiţimii { 1,..., n} in vectorul x=(x_1,x_2,...,x_{n1}) unde nl=n...2 (deci, se începe cu generarea soluţiilor cu lungimea cea mai mare);
- dintre vectorii generaţi, se aleg cei care verifica proprietate: între x_i si x_{i+1} există muchie şi se afişează cei cu lungimea cea mai mare (acest lucru estc ilustrat în functia Afis).

Comentarii la functia Afis:

- la primul apel al funcţiei (ns=l) se păstrează lungimea vectorului soluţie (lung=k) pentru ca aceasta este cea mai mare;
- la urmatoarele apeluri nu se ia in calcul decât vectorii cu lungimca egală cu lung

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir [ 100];
typedef int mat[20][20];
int el, e2; int i, j , n,n l, m , k, ns, lung;
mat a;
sir x;  int as, ev;
void succ(sir x, int k, int &as)
{ if (x[k]<n)
                                        {as=1;
                                        x[k]=x[k]+1; }
                                   else as=0;}
void afis(sir x, int k)
{ int i;
ns=ns+ 1;
if (ns==1) lung=k;
if (k==lung)
                                        {for (i=1;i<=k,i++) cout<<x[i]<<” ”;
                                        cout<<endl;}
```

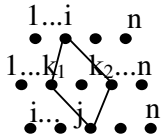
```
}
void valid(sir x, int k, int &ev)
{int i; ev=1;
if (a[x[k-1]][x[k]]==0) ev=0;
   else
        for (i=1;i<=k-l;i++)
              if (x[k]== x[i]) ev=0;
}
void main()
{ clrscr();
cout<<"n=”; cin>>n; cout<<"m=’; cin>>m;
for (i=1;i<=m;i++)
       {cout<<"el e2 ";cin>>el>>e2;
       a[e1][e2]=1;
       a[e2][el]=1;}
ns=0;
for (n1=n;n1>=2;n1--)
                                        {x[1]=0;
                                        k=1;
                                        while (k>0){
                                                do{
                                                    succ(x,k,as);
                                                    if (as) valid(x,k,ev);
                                                    }while (as && !ev);
                                                if (as)
                                                   if (k= nl) afis(x,k);
                                                          else
                                                              {k=k+ 1;
                                                              x[k]=0;}
                                                  else k=k-1;
                                                }
                                        }
                                    getche();}
```

Problema 4

Dacă ar exista un ciclu de lungime 4 ar fi de forma: i, kl, j, k2, i. Pentru a demonstra existența unui astfel de ciclu, procedăm astfel:

- încercăm să arătăm că pentru o pereche de noduri (i, j) există alte două noduri kl şi k2 diferite de acestea şi legate de ele prin muchii (a[i,kl]=1, a[j,kl]=1 şi a[i,k2]=1, a[j,k2]=1)

```
.....................
ok=0;
        for (i=1;i<=n-l;i++)
                for (j=i+l ;j<=n;j++)
nr =0;
for (k=1;k<=n;k++)
if ((i!=k) && (j!=k) && (a[i][k]==1) && (a[j][k]==1)
                        nr=nr+ 1;
if (nr>= 2)
ok=1;}
.....................
```



Problema 5

Problema se rezolvă folosind metoda Backtracking, şi se reduce la a determina şirurile de forma $x_1...,x_p$ unde:

-x1=1

-componentele sunt distincte;

- între x_i şi x_{i+1} i=1...p-1, să existe muchie în graful dat;

- x_1 şi x_p sunt unite printr-o muchie.

În concluzie, soluția este $x=(x_1,x_2,....x_p)$, unde $x_1=1$, $x_k \in \{2,...,n\}$ $k \in \{2,...,p\}$;
$x_i \neq x_j$ pentru $i \neq j$, a[$x_{i-1}x_i$]=1 pentru i=2..p, şi a[x_1,x_p]=1

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
typedef int sir[100];
typedef int mat[20][20];
int e 1., e2;
int i, j, n, m, p, k;
mat a;
sir x;
int as, ev;
void succ(sir x, int k, int &as)
{if (x[k]<n)
       {as=1;
       x[k]=x[k]+1;}
   else as=0;}
void afis(sir s, int k)
{int i;
for (i=1;i< =k;i++)     cout<<x[i]<<” ”;
```

```
cout<<x[1]<< "          ;
cout<<endl;}
void valid(sir x, int k, int &ev)
{int i;
ev=1;
if (a[x[k-1]][x[k]]==0) ev=0;
else
        {for (i=1;i<=k-l;i++)
                if (x{k{==x[il) ev=0;
        if ((k==p) && (a[ x[ k]][ x[1]]==0))
ev=0;}
}
void main()
{ clrscr();
cout<< " n="; cin>>n;
cout<<m="; cin>>m:

for(i=1;i<=m;i++)
          {cout<<"el   e2 "; cin>>el>>e2;
          a[el][e2]=1;
                a[e2][e1]==1;}
cout< <"p="; cin>>p;
x[1]=1;
k=2;
x[k]=1;
while (k>1) {
                do{
                   succ(x,k,as );
                   if (as) valid(x,k,ev);
                   }while (as && !ev);
                if (as)
                  if (k==p) afis(x,k);
                        else
                            {k=k+ l ;
                             x[k]=l;}
                else k=k-1;
                }
getche();}
```

Problema 6

Până la urmă, este o problemă de parcurgere a unui graf şi se rezolzâ astfel:
- se citeşte matricea de adiacenţă;
- toate nodurile se consideră iniţial nevizitate;
- la început, sunt 0 componente conexe;
- se parcurg nodurile grafului
    - dacă se găseşte un nod nevizitat încă
        - înseamnă ca s-a mai găsit o componentă conexă
        - şi se vizitează toate nodurile la care se poate ajunge plecând fie la acest nod (pentru aceasta este nevoie de una dintre procedurile de parcurgere a unui graf (în adâncime sau în lăţime))

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int viz[ 100],a[20][20], i,n,m,x,y,nr,j;
void parc_adancime(int pl)
{int j;
viz[pl]=1;
for (j=1;j<=n;j++)
if ((a[pl][j]==1) && (viz[j]==0))
parc_adancime(j);}
 void main()
{ clrscr();
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;}
for (i=1;i<=n;i++) viz[i]=0;
nr=0;
for (i=1;i<=n;i++)
if (viz[i]==0)
{nr=nr+1;
parc_adancime(i);}
cout<<"are "<<nr<<" componente conexe";
getche();}
```

Problema 7

Se procedează astfel:
- se citeşte matricea de adiacentă;
- toate nodurile se consideră iniţial nevizitate;
- se parcurg nodurile grafului
    - dacă se găseşte un nod nevizitat încă,
        - se vizitează si se afisează:
        -se vizitează şi se afişează toate nodurile la care se poate ajunge plecând de la acest nod

(acest lucru se realizează printr-un apel ai procedurii parc_adancime).

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int viz [ 100],a[20][20],i,n,m,x,y,nr;
void parc_adancime(int pl )
{int j;
viz[pl]=1; cout<<pl<<" ";
for (j =1;j<=n;j++)
if ((a[pl][j]==1) && (viz[j]==0))
        parc_adancime(j);}
void main()
{ clrscr(); 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;}
for (i=1;i<=n;i++) viz[i]=0;
nr=0;
for (i=1;i<=n;i++)
    if (viz[i]==0)
          {nr=nr+ 1 ;
        cout<< " componenta conexa cu numarul" <<nr <<" are
nodurile"<<endl;
        parc_adancime(i);
        cout<<endl;}
getche( );
```

Problema 8

Problema se reduce, până la urmă, la a afişa toate perechile de forma (il, i2), unde il si i2 sunt noduri din aceeaşi compunentă conexă pentru aceatsta:

- se generează toate componentele conexe:
- pentru fiecare componetă conexă generată, se afişează toate perechile ordonate de noduri (il,i2), care se pot forma cu elementele sale.

Observaţie. Pentru a nu încurca nodurile care fac parte din componente conexe diferite, astfel încât să afişăm de două ori aceleaşi noduri, procedăm astfel:

- după ce se afişează perechile formate din nodurile unei componente conexe, se măresc cu 1 toate elementele din vectorul viz, care corespund nodurilor componentei afişate, pentru a deveni 2, astfel încât să nu fie confundate cu modurile componentei conexe care se va determina, pentru care vectorul viz va avea valoarea 1.

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>
int viz[100],a[20][20], i,n,m,x,y,i1,i2;
void parc_adancime(int pl)
{int j; viz[pl]=1;
for (j=1 ;j<= n;j++)
        if ((a[pl][j]==1) && (viz[j]==0))
                parc_adancime(j);}
void main()
{ clrscr();
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;}
for (i=1;i<=n;i++) viz[i]=0;
for (i=1;i<=n;i++)if (viz[i]==0)
        { parc_adancime(i);
        for (i1=1;i1<=n-1;i1++)
        for (i2=i1+1 ; i2<=n; i2++)
                if ((viz[i1]==1) && (viz[i2]==1))
                        cout<<"("<<i1<<","<<i2<
                <")";
        for (i1=1;i1<=n;i1++)
                if (viz[i1]==1) viz[i1]=viz[i1]+1;
        cout<<endl;}
getche();}
```

Problema 9

Sistemului de culoare i se pune în corespondenţă un graf neorientat, astfel:

- nodurile grafului sunt culoarele;
- muchiile grafului sunt legăturile între culoare.

A verifica dacă toate culoarele fac parte din aceeaşi peşteră, înseamnă a verifica dacă graful asociat este conex (adică este format dintr-o singură componentă conexă):

- la început, nodurile sunt nevizitate;
- se aplică funcţia de parcurgere a grafului, plecând de la un nod oarecare;
- dacă, după parcurgere, în graf mai sunt noduri nevizitate, nu este conex;

(Vezi problema rezolvată din secţiunea 10)

```
cout<<"dati nodul de plecare pare adancime(pl);
ok=0;
for (i=1;i<=n;i++)
        if (viz[i]==0) ok=1;
if (ok) cout<<"sunt mai multe pesteri";
        else cout<<"este o singura pestera";
```

Problema 10

Grupului de persoane, despre care se vorbeşte i se pune în corespondență un graf neorientat, astfel:
- nodurile grafului sunt persoanele;
- muchiile grafului sunt precizările ideii că două persoane sunt prietene (în mod direct).

A determina grupurile de persoane, cu un număr maxim de membri, între care se pot stabili prietenii, directe sau indirecte, se reduce la a detennina componentele conexe cu cele mai multe noduri (este o problemă de maxim). Acest lucru se realizează astfel:
- pe măsură ce se determină componenta conexă, cu numărul nr,
  - se determină numărul de noduri ale sale în comp[nr] şi i se păstrează nodurile în mulțimea varfuri[nr]
  - dacă comp[nr]> max (max - cel mai mare număr înregistrat până în prezent) devine max
- se afişează nodurile tuturor componentelor conexe care au max noduri.

Atenție! Funcția parc_adancime are un parametru în plus (nr), pe care îl foloseşte în scopul de a diferenția elementele diferitelor componente conexe, astfel:
- la fiecare apel, elementele care au fost vizitate sunt marcate cu nr (viz[i]=nr).

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h> t
typedef int sir [ 100];
typedef int mat[20][20];
mat a;
sir viz, comp;
int i, n , m , x, y, max,j, nr;
mat varfuri;
void parc_adancime(int pl, int nr)
{ int j;
viz[pl]=nr;
for (j=1;j<=n;j++)
if ((a[pl][j]==1) && (viz[j]==0))
parc_adancime(j,nr);}
void main()
{ clrscr();
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;}
for (i=1;i<=n;i++) viz[i]=0;
max=0; nr=0;
for (i=1;i<=n;i++)
if (viz[i]==0) { nr=nr+ 1;
parc_adancime(i,nr);
comp[nr]=0;
for(j=1;j <=n;j++)
if (viz[j]==nr)
{ comp[nr]=comp[nr]+1;
varfuri[nr][comp[nr]]=j;}
if (comp[nr]>max)
max=comp[nr];}
for (i=1;i<=nr;i++)
if (comp[i]==max)
{ cout<<i<<":::::";
for (j=1;j<=comp[i];j++)
cout<<varfuri[i][j]<<" ";
cout<<" ";}
getch();
}
```

11. Cicluri Hamiltoniene

Problema 1

L=[1, 3, 5, 4, 2]: este lanț hamiltonian;
C=[5, 4, 2, 3, 1, 5] : este ciclu hamiltonian;

Problema 2

Dacă $n \geq 3$ şi $d(x) \geq n/2$ pentru orice nod x, atunci graful este hamiltonian.

```
.....................
int grad(int i)
{int     s, j;
s=0;
for (j=1;j<=n;j++)
```

```
        s=s+a [i][j];
return s;}
.....
ok 1=( n>=3 );
ok2=1;
for (i=1;i<=m;i++)
if (! (grad(i)>=n/2)) ok2=0;
ok==ok 1 && ok2;
if (ok) cout<<"se verifiica conditiile teoremei ";
        else cout<<"nu se verifica conditiile teoremei";
.....
```

Problema 3

Demonstrăm acest lucru folosind metoda inducției matematice. Pentru a uşura scrierea, vom nota cu H(n) numărul ciclurilor hamiltoniene ale grafului K_n în baza acestei notații, enunțul problemei care să demonstrăm că: $H(n)=\frac{(n-1)!}{2}\forall n\geq 3$

I. Verificare:

Pentru n= 3 $H(n)=H(3)=\frac{(3-1)!}{2}=\frac{2!}{2}=1$ (adevărat, pentru că într-un graf complet cu trei vârfuri este un singur ciclu hamiltonian)

II. Demonstratia P(n) →P(n+ l)

(n_1)!

P(n) : $H(n)=\frac{(n-1)!}{2}$

P(n+l) : $H(n+1)=\frac{(n+1-1)!}{2}$

Altfel spus, dacă se ştie că numărul ciclurilor hamiltoniene în graful K_n este (n-1)!/2, să se demonstreze că numărul ciclurilor harniltoniene în K_{n+1} este n!/l.

Având la bază faptul că din fiecare ciclu hamiltonian din K_n se pot obține n cicluri hamiltoniene distincte în K_{n+1} prin intercalarea nodului n+l în toate cele n locuri posibile (între două noduri succesive; ex: x_1 x_2 ... x_n x_1(între x_1 x_2, x_2 x_3, ..)), putem spune că în K_{n+1} sunt n ori numărul ciclurilor din K_n cicluri hamiltoniene, ceea ce ne permite să scriem:

$$H(n+1)=n\cdot H(n)=n\cdot\frac{(n-1)!}{2}=\frac{(n)!}{2}=\frac{(n+1-1)!}{2}$$

Problema 4

Având la bază:

1) fiecare ciclu elementar este un ciclu hamiltonian in subgraful complet indus de vârfurile sale şi invers, fiecare ciclu hamiltonian dintr-un subgraf cu k vârfuri este ciclu elementar în G.

2) într-un graf complet cu k vârfuri sunt $\frac{(k-1)!}{2}$ cicluri hamiltoniene;

3) într-un graf cu n vârfuri pot exista C_n^k subgrafuri cu k noduri:

tragem concluzia:

Într-un graf cu n vârfuri, numărul ciclurilor elementare este egal cu:

$$C_n^3\frac{(3-1)!}{2}+C_n^4\frac{(4-1)!}{2}+...+C_n^n\frac{(n-1)!}{2}=\sum_{k=3}^{n}C_n^k\frac{(k-1)!}{2}=\sum_{k=3}^{n}\frac{n!}{k!(n-k)!}\frac{(k-1)!}{2}=\sum_{k=3}^{n}\frac{n!}{k(k-1)!(n-k)!}\frac{(k-1)!}{2}$$

$$=\frac{1}{2}\sum_{k=3}^{n}\frac{n(n-1)...(n-k+1)}{k}$$

Problema 5

Problema se face la fel ca şi problema determinării ciclurilor hamiltoniene, pe care am prezentat-o în secțiunea 11., numai că în situatia de față, va trebui ca în loc să se afişeze ciclurile să se dea răspunsul da sau nu. Acest lucru se poate face astfel:

Se foloseşte o variabilă booleană ok, care la începutul programului are valoarea false (se presupune ca nu există cicluri), iar în procedura de afişare să primească valoarea true (adică s-a găsit un ciclu hamiltonian; eventual, la primul apel al procedurii Afis să se renunțe la generarea ciclurilor care ar mai putea fi găsite) şi să se afişeze înainte de terminarea programului.
Altfel:
Se procedează la fel ca la problema 2 (adică, se verifică dacă sunt satisfăcute condițiile teoremei prezentată în secțiunea 11)

Problema 6

Asociem grupului de persoane, despre care se vorbeşte în enunțul problemei, graf neorientat definit astfel: nodurile grafului reprezintă persoanele iar muchia între nodurile i şi j reprezintă precizarea ideii că persoanele i şi j sunt prietene, adică în relație de neduşmănie. Din ipoteza că fiecare cavaler are cel mult (n-1) duşmani, rezultă că pentru fiecare cavaler există cel puțin 2n-(n-1)=(n+1)≥2n/2 cavaleri cu care acesta se află în relație de neduşmănie, adică de prietenie. Ținând cont de cele spuse mai sus, putem trage concluzia că în graful asociat grupului de persoane sunt veriticate condițiile teoremei prezentată în secțiunea 11. conform căreia în graful respectiv există cel puțin un ciclu hamiltonian. Ciclului hamiltonian îi asociem o aşezare a cavalerilor la masa rotundă în condițiile cerute.