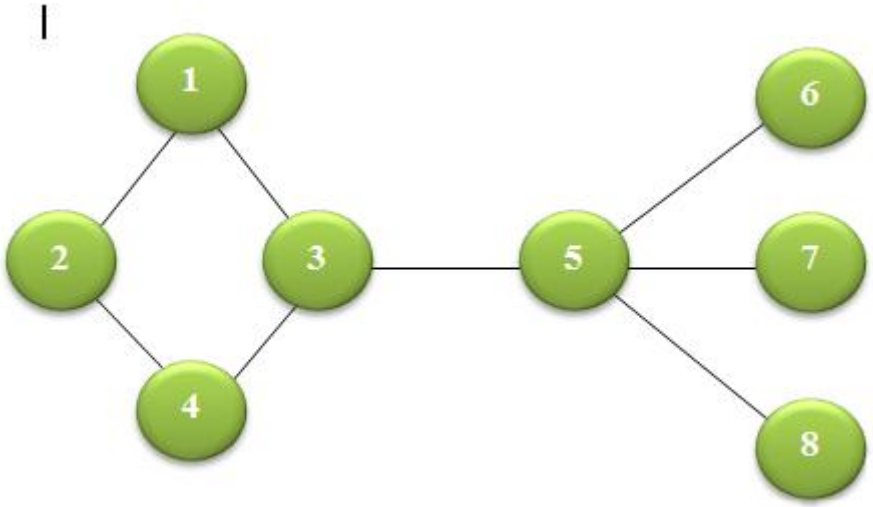
Conexitate.

Definiţie: Un graf se numeşte **conex** dacă pentru oricare două vârfuri **x** şi **y** diferite ale sale, există un lanţ care le leagă.
Definiţie: Se numeşte **componentă conexă** a grafului **G=(X,U)**, un subgraf **C=(X_1,U_1)** conex al lui **G** care are proprietatea că nu există nici un lanţ în **G** care să lege un vârf din mulţimea X_1 cu un vârf din mulţimea $X-X_1$.

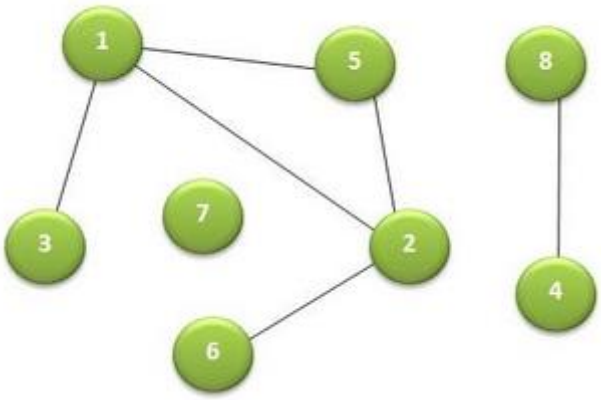
Exemplu 1: GRAF CONEX



GRAF CONEX - format dintr-o singură componentă conexă

(conţine toate nodurile grafului)

Exemplu 2: GRAF NECONEX



Graf NECONEX - alcătuit din trei componente conexe
Componenta conexa 1 conţine nodurile: 1,3,5,2,6
Componenta conexa 2 conţine nodurile: 4, 8
Componenta conexa 3 contine nodurile: 7

APLICATIE

În fişerul text **graf.in** este memorat un **graf neorientat,** astfel:

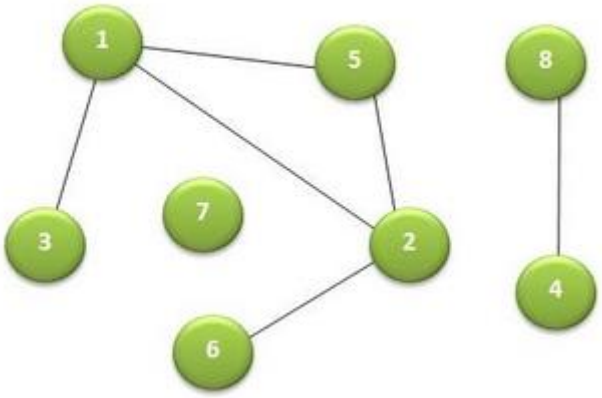
- pe prima linie un număr natural **n**, care reprezintă numărul de noduri ale unui graf neorientat.
- pe următoarele linii sunt memorate câte două numere naturale care reprezintă muchiile grafului.

Scrieţi un program care **să verifice dacă graful este conex.**

Exemplul 1:

Continutul fişierului text **graf.in**	Rezultate asteptate
8 **1 3** **1 2** **1 5** **2 5** **2 6** **4 8**	Dacă nodul de start este 1 atunci se vor afişa următoarele rezultate: **Graful NU este conex!**

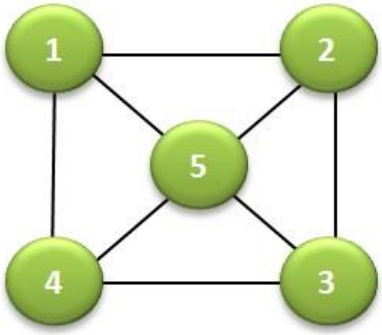
Graful memorat în fişerul text de mai sus are următorul aspect grafic:



Exemplul 2:

Continutul fişierului text **graf.in**	Rezultate asteptate
5 **1 2** **1 4** **1 5** **2 3** **2 5** **3 5** **3 4** **4 5**	Dacă nodul de start este 1 atunci se vor afişa următoarele rezultate: **Graful este conex!**

Graful memorat în fişerul text de mai sus are următorul aspect grafic:



Program C++:

```
#include<iostream>
#include<fstream>
using namespace std;
int a[20][20],c[20],v[20],ns,n,comp;
int prim;
int ultim;
```

```
// citirea grafului din fisier text si construirea matricei de adiacenta
```

```
void citire(int a[20][20], int &n)
{   ifstream f("graf.in");
    int x,y;
    f>>n;
    while(f>>x>>y)
       a[x][y]=a[y][x]=1;
     f.close();
}
```

```
// afisarea pe ecran a matricei de adiacenta
```

```
void afisare(int a[20][20],int n)
{    cout<<"Matricea de adiacenta este : "<<endl;
     for( int i=1;i<=n;i++)
       { for(int j=1;j<=n;j++)
            cout<<a[i][j]<<" ";
         cout<<endl;
       }
}
```

```
// returnează primului nod nevizitat
```

```
int exista_nod_nevizitat(int v[20], int n)
{   for(int i=1;i<=n;i++)
      if(v[i]==0)
         return i; // primul nod nevizitat
      return 0;   // nu mai exista noduri nevizitate
}
```

```
// parcurgerea în latime a unei componente conexe, plecând din nodul de start ns
```

```
void parcurgere_latime(int a[20][20], int n,int ns)
   { comp++;
     v[ns]=1;
     cout<<"Componenta conexa : "<<comp<<" este formata din nodurile :";
     cout<<ns<<" ";
     prim=ultim=1;
     c[ultim]=ns;
     while(prim<=ultim)
        {for(int i=1;i<=n;i++)
           if(a[c[prim]][i]==1)
             if(v[i]==0)
               { ultim++;
                 c[ultim]=i;
                 cout<<i<<" ";
                 v[i]=1;
               }
           prim++;
         }
      cout<<endl;
   }
```

```
// functia principala main()
```

```
int main()
{ citire(a,n);
```

```
  afisare(a,n);
  parcurgere_latime(a,n,1);
  if (exista_nod_nevizitat(v,n)!=0)
     cout<<"Graful NU este conex!";
  else
     cout<<"Graful este conex!";
return 0;
}
```