**Verificare ciclu cu BF**

**int bf(int s)**

**{**

**int st,dr,i,x[50],j;**

**st=dr=1; viz[s]=1; x[1]=s;**

**while(st<=dr)**

**{**

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

**if(a[x[st]][i]!=0)**

**{**

**if(!viz[i])**

**{**

**dr++;**

**x[dr]=i;**

**viz[i]=1;**

**a[x[st]][i]=a[i][x[st]]=0;// sterg muchia**

**}**

**else return 1;// are ciclu plecand din nodul s**

**}**

**st++;**

**}**

**return 0;**

**}**

**……………….**

**for(int i=1;i<n;i++) if(bf(i)) ok=1;**

**//Afisare componente conexe**

**int bf(int s)**

**{**

**int st,dr,i,x[50],j;**

**st=dr=1; viz[s]=1; x[1]=s;**

**while(st<=dr)**

**{**

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

**if(a[x[st]][i]!=0 && !viz[i])**

**{**

**dr++;**

**x[dr]=i;**

**viz[i]=1;**

**}**

**st++;**

**}**

**return 0;**

**}**

**void df(int k)**

**{**

**viz[k]=1;**

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

**if(a[x[k-1][i]==1 && viz[i]==0)**

**{ x[k]=i; df(k+1);}**

**}**

**………………………….**

**for(k=1;k<=n;k++)**

**if(!viz[k])**

**{**

**df(k); // bf(k);**

**afis();**

**}**

**Graful este hamiltonian. Determinare ciclu hamiltonian  
n noduri, m muchii**

**void hamilton(int nod, int k)**

**{**

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

**if(!viz[i] && a[x[k-1]][i])**

**{**

**x[k]=i;**

**viz[i]=1;**

**if(x[k]==nod && k==n+1) afis(k);**

**else back(nod,k+1);**

**viz[i]=0;**

**}}**

**…………………………..**

**x[1]=1 ;**

**back(2)**

**void cicluelementar(int nod, int k) // toate ciclurile elementare plecand din varful nod**

**{**

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

**if(!viz[i] && a[x[k-1]][i])**

**{**

**x[k]=i;**

**viz[i]=1;**

**if(x[k]==nod && k>3) afis(k);**

**else cicluelementar(nod,k+1);**

**viz[i]=0;**

**}}**

**//toate ciclurile de lungime k**

**int valid(int j) // verifica conditii de continuare**

**{if(j>1)**

**if(a[x[j-1]][x[j]]==0) return 0; /// nu exista muchie**

**if(j==k) if(a[x[j]][x[1]]==0) return 0;// nu se formeaza ciclu**

**return 1;**

**}**

**void back(int j)**

**{for(int i=1;i<=n;i++)**

**if(!viz[i])**

**{x[j]=i;**

**viz[i]=1;**

**if(valid(j)) // sunt indeplinite conditiile**

**if(j==k) afis();**

**else back(j+1);**

**viz[i]=0;**

**}**

**}**

**// toate ciclurile euleriene cu Backtracking**

**int valid(int k)**

**{if(k>1)**

**if(a[x[k-1]][x[k]]==0) return 0;**

**return 1;**

**}**

**void back(int k)**

**{for(int i=1;i<=n;i++)**

**{x[k]=i;**

**if(valid(k)) {**

**a[x[k-1]][x[k]]=0;// se sterg muchiile**

**a[x[k]][x[k-1]]=0;**

**if(k==m+1) afis();**

**else back(k+1);**

**a[x[k-1]][x[k]]=1;// se repun muchiile**

**a[x[k]][x[k-1]]=1;**

**}**

**}**

**}**

**distantele minime de la varful k la celelalte varfuri ale grafului.**

**Date.in**

**12 13   
1 2   
1 3   
1 4   
2 8   
3 6   
3 7   
4 6   
4 5   
8 9   
7 9   
7 10   
6 10   
12 11   
8   
Fisierul de iesire:   
1 2   
2 1   
3 3   
4 3   
5 4   
6 4   
7 2   
8 0   
9 1   
10 3   
11 -   
12 -**

**void bf(int k)**

**{**

**int st,dr,j;**

**st=dr=1; x[1]=k; viz[k]=1;**

**d[k]=0;// vector distante minime**

**while(st<=dr)**

**{**

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

**if(a[x[st]][j]==1 && viz[j]==0)**

**{**

**dr++;**

**x[dr]=j;**

**viz[j]=1;**

**d[j]=d[x[st]]+1;// vectorul cu distantele**

**}**

**st++;**

**}**

**}**

**Graf eulerian ? graf conex + toate nodurile au grade pare**

**date.in:   
6 10   
1 2   
1 6   
2 3   
2 5   
2 6   
3 4   
3 5   
3 6   
4 5   
5 6   
date.out: este eulerian  1 2 3 5 6 2 5 4 3 6 1**

**int n,m,a[100][100],viz[100],k,g[100];**

**// graf eulerian = graf conex cu grade pare**

**int grad(int k)//calculeaza gradul varfului k**

**{**

**int s=0;**

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

**if(a[k][i]==1) s++;**

**return s;**

**}**

**void DF(int s)//parcurge graful din varful s si marcheaza varfurile accesibile**

**{**

**viz[s]=1;**

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

**if(a[s][i]==1 && viz[i]==0)**

**DF(i);**

**}**

**int conex() //verific conexitatea grafului**

**{**

**DF(1);**

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

**if(viz[i]==0) return 0;**

**return 1;**

**}**

**int euler()// verific daca este eulerian**

**{**

**if(!conex()) return 0; //conex**

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

**if(g[i]%2==1) return 0;//si toate gradele pare**

**return 1;**

**}**

**g[i]=gradul nodului i**

**DETERMINARE CICLU EULERIAN**

**void ciclu\_eulerian(int k)//construieste un ciclu eulerian**

**{**

**int maxx=0,nmax=0;**

**g<<k<<" "; //afiseaza varful curent**

**for(int i=1;i<=n;i++)//cauta varful urmator cu grad maxim**

**{**

**if(a[k][i]==1)**

**if(g[i]>maxx)**

**{ maxx=grad(i); nmax=i; }**

**}**

**if(nmax!=0)**

**{ a[k][nmax]=a[nmax][k]=0;//sterge mughia**

**g[k]--;//scade gradele celor 2 noduri incidente cu muchia stearsa**

**g[nmax]--;**

**ciclu\_eulerian(nmax);//merge in varful urmator- recursiv**

**}**

**}**

**Se da un graf neorientat cu n noduri si m muchii precizat prin lista muchiilor. Se citesc apoi 2 varfuri x si y.**

**Afisati toate lanturile elementare care au ca extremitate initiala varful x si extremitate finala varful y.**

**date.in:   
8 10   
1 2   
1 3   
2 3   
2 8   
3 4   
3 7   
4 5   
5 6   
6 7   
7 8   
8 4**

**date.out:   
8 2 1 3 4   
8 2 1 3 7 6 5 4   
8 2 3 4   
8 2 3 7 6 5 4   
8 7 3 4   
8 7 6 5 4**

|  |
| --- |
| **void back(int k)**  **{**  **for(int i=1;i<=n;i++)**  **if(!viz[i] && a[x[k-1]][i]==1)**  **{**  **x[k]=i;**  **viz[i]=1;**  **if(i==y) afis(k);**  **else back(k+1);**  **viz[i]=0;**  **}}**  **int main()**  **{ citire();**  **x[1]=x; viz[x]=1;**  **back(2);**  **}** |
|  |
|  |

**Se da un graf neorientat conex cu n varfuri si m muchii.**

**Eliminati un numar minim de muchii din graf astfel incat acesta sa nu mai contina nici un ciclu.**

**Se vor afisa muchiile eliminate si matricea de adiacenta a grafului partial obtinut astfel.   
Exemplu:**

**date.in   
4 6   
1 2   
1 3   
1 4   
2 3   
2 4   
3 4   
date.out   
[2,3]   
[2,4]   
[3,4]   
0111   
1000   
1000   
1000**

**void sterg(int s)**

**{**

**int st,dr,i,x[50];**

**st=dr=1; viz[s]=1; x[1]=s;**

**while(st<=dr)**

**{**

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

**if(a[x[st]][i]==1)**

**if(!viz[i])**

**{ dr++;**

**x[dr]=i;**

**viz[i]=1;**

**a[i][x[st]]=0;// sterg muchia pe car am parcurs-o**

**}**

**else**

**{**

**fout<<”[“<<x[st]<<”,”<<i<<”]”<<endl;**

**a[x[st]][i]=a[i][x[st]]=0; // sterg definitiv muchia**

**}**

**st++; }}**

**int main()**

**{ int x,y,i,j;**

**sterg(1); for(i=1;i<=n;i++) for(j=1;j<i;j++) a[i][j]=a[j][i]; //repun muchiile sterse la parcurgere**

**Se citeste un graf neorientat cu n noduri si m muchii dat prin vectorul muchiilor.   
- Construiti MATRICEA LANTURILOR folosind algoritmul Roy-Warshall.   
- Folosind matricea lanturilor afisati componentele conexe.**

**Exemplu:   
date.in   
7 4   
1 2   
1 3   
2 5   
6 7   
date.out   
1 2 3 5   
4   
6 7**

**int k,m,n,x[100],a[100][100],viz[100];**

**void rw()**

**{int i, j, k;**

**for(k=1;k<=n;k++)**

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

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

**a[i][j]=a[i][j] || ( a[i][k]&&a[k][j])**

**}**

**void afis() // afisare lanturi intre oricare 2 noduri**

**{for(int i=1;i<=n;i++)// afisare lanturi pornind din nod i**

**if(!viz[i])**

**{ g<<i<<" "; viz[i]=1;**

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

**if(a[i][j])**

**{ g<<j<<" "; viz[j]=1;}**

**g<<endl;**

**}**

**}**

**Se da un graf neorientat cu n varfuri si m muchii prin lista muchilor. Construiti matricea lanturilor folosind parcurgerea in adancime.   
Exemplu:**

**date.in   
8 9   
1 2   
1 4   
2 3   
3 4   
3 5   
5 6   
7 8   
4 6   
date.out   
0 1 1 1 1 1 0 0   
1 0 1 1 1 1 0 0   
1 1 0 1 1 1 0 0   
1 1 1 0 1 1 0 0   
1 1 1 1 0 1 0 0   
1 1 1 1 1 0 0 0   
0 0 0 0 0 0 0 1   
0 0 0 0 0 0 1 0**

**int a[50][50],b[50][50],n,m,k;int viz[50];**

**void df(int s,int k)**

**{**

**viz[k]=1;**

**if(k!=s) b[s][k]=1;**

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

**if(!viz[i] && a[k][i])**

**df(s,i);}**

**int main()**

**{ for(int i=1;i<=n;i++)**

**{**

**df(i,i);**

**}**

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

**{**

**for(int j=1;j<=n;j++) g<<b[i][j]<<" ";**

**g<<endl;**

**}}**

**Roy Floyd drum de cost minim cu afisare drum**

**void RoyFloyd() //**

**{for(int k=1;k<=n;k++)**

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

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

**if(a[i][j]>a[i][k]+a[k][j])**

**a[i][j]=a[i][k]+a[k][j];**

**}}**

**void descompun\_drum(int i, int j) // descompunere drum de la i la j prin k**

**{int ok=0,k=1;**

**while(k<=n&&!ok) //**

**{**

**if(i!=k&&j!=k)**

**if(a[i][j]==a[i][k]+a[k][j])**

**{**

**descompun\_drum(i,k);**

**descompun\_drum(k,j);**

**ok=1; // se poate realiza descompunerea**

**}**

**k++;**

**}**

**if(!ok)**

**{**

**g<<j; }//cand “drumul” nu mai poate fi descompus afisez extremitatea finala**

**}**

**void scriu\_drum(int ni,int nf)**

**{**

**if(a[ni][nf]< inf)**

**{**

**g<< a[ni][nf];// cost drum minim**

**descompun\_drum(ni,nf);**

**}**

**else**

**g<<” nu exista lant “ ;**

**}**

**Se da un graf neorientat cu n varfuri si m muchii, citit prin vectorul muchiilor si apoi un varf k.**

**Sa se afiseze pe linii separate distantele minime de la varful k la celelalte varfuri ale grafului si lanturile de lungime minima corespunzatoare acestor distante. Distanta de la un varf la altul se considera a fi numarul de muchii din cel mai scurt lant care uneste cele doua noduri.**

**Se va folosi un algoritm de tip breadth first.**

**Exemplu:**

**Fisierul de intrare:**

**12 13**

**1 2**

**1 3**

**1 4**

**2 8**

**3 6**

**3 7**

**4 6**

**4 5**

**8 9**

**7 9**

**7 10**

**6 10**

**12 11**

**8**

**Fisierul de iesire:**

**1 2 8 2 1**

**2 1 8 2**

**3 3 8 2 1 3**

**4 3 8 2 1 4**

**5 4 8 2 1 4 5**

**6 4 8 2 1 3 6**

**7 2 8 9 7**

**8 0 8**

**9 1 8 9**

**10 3 8 9 7 10**

**11 -**

**12 –**

**void bf(int k)**

**{**

**int st,dr,j;**

**st=dr=1;**

**x[1]=k;**

**viz[k]=1;**

**d[k]=0;**

**while(st<=dr)**

**{**

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

**if(a[x[st]][j]==1 && viz[j]==0)**

**{**

**dr++;**

**x[dr]=j;**

**viz[j]=1;**

**d[j]=d[x[st]]+1;// vectorul cu distantele**

**t[j]=x[st];// tatal nodului j**

**}**

**st++;**

**}}**

**void lant(int k)**

**{ if(t[k]!=0) lant(t[k]);**

**fout<<k<<" ";}**

**Se da un graf neorientat cu n varfuri si m muchii prin lista muchilor si un varf k. Afisati toate ciclurile elementare care au ca extremitate initiala si finala varful k.**

**Exemplu:   
date.in   
8 10 1   
1 2   
1 4   
2 3   
3 4   
3 5   
5 6   
7 8   
4 6   
1 3   
2 4   
date.out   
1 2 3 1   
1 2 3 4 1   
1 2 3 5 6 4 1   
1 2 4 1   
1 2 4 3 1   
1 2 4 6 5 3 1   
1 3 2 1   
1 3 2 4 1   
1 3 4 1   
1 3 4 2 1   
1 3 5 6 4 1   
1 3 5 6 4 2 1   
1 4 2 1   
1 4 2 3 1   
1 4 3 1   
1 4 3 2 1   
1 4 6 5 3 1   
1 4 6 5 3 2 1**

**int a[50][50],n,m,k;**

**int X[50],viz[50];**

**void back(int k, int pas)**

**{**

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

**if(!viz[i] && a[X[pas-1]][i])**

**{**

**X[pas]=i;**

**viz[i]=1;**

**if(X[pas]==k && pas>3) afis(pas);**

**else back(k,pas+1);**

**viz[i]=0;**

**}}**

**int main()**

**{**

**int x,y;**

**f>>n>>m>>k;**

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

**{**

**f>>x>>y;**

**a[x][y]=a[y][x]=1; } X[1]=k; back(k,2); }**

**Se da un graf neorientat cu n varfuri si m muchii prin lista muchilor. Construiti matricea lanturilor folosind parcurgerea in adancime.   
Exemplu:**

**date.in   
8 9   
1 2   
1 4   
2 3   
3 4   
3 5   
5 6   
7 8   
4 6   
date.out   
0 1 1 1 1 1 0 0   
1 0 1 1 1 1 0 0   
1 1 0 1 1 1 0 0   
1 1 1 0 1 1 0 0   
1 1 1 1 0 1 0 0   
1 1 1 1 1 0 0 0   
0 0 0 0 0 0 0 1   
0 0 0 0 0 0 1 0**

**int a[50][50],L[50][50],n,m,k;int viz[50];**

**void df(int s,int k)**

**{**

**viz[k]=1;**

**if(k!=s) L[s][k]=1;**

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

**if(!viz[i] && a[k][i])**

**df(s,i);}**

**int main()**

**{ for(int i=1;i<=n;i++)**

**{**

**df(i,i);**

**}**

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

**{**

**for(int j=1;j<=n;j++) g<<l[i][j]<<" ";**

**g<<endl;**

**}}**

**Se citeste un graf neorientat cu n noduri si m muchii dat prin vectorul muchiilor.   
- Construiti matricea lanturilor folosind algoritmul Roy-Warshall.   
- Folosind matricea lanturilor afisati componentele conexe.   
Exemplu:   
date.in   
7 4   
1 2   
1 3   
2 5   
6 7   
date.out   
1 2 3 5   
4   
6 7**

**int k,m,n,x[100],a[100][100],viz[100];**

**void rw()**

**{int i, j, k;**

**for(k=1;k<=n;k++)**

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

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

**if(i!=j)**

**if(a[i][j]==0)**

**a[i][j]=a[i][k]\*a[k][j];**

**}**

**void afis() /// afisare componente conexe**

**{for(int i=1;i<=n;i++)**

**if(!viz[i])**

**{ g<<i<<" ";**

**viz[i]=1;**

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

**if(a[i][j]) { g<<j<<" "; viz[j]=1;}**

**g<<endl;**

**}**

**}**

**int main()**

**{citire();**

**rw();**

**afis();**

**return 0;**

**}**

**Se da un graf neorientat cu n varfuri si m muchii. Determinati numarul minim de culori cu care se pot colora varfurile grafului astfel incat oricare doua varfuri adiacente sa aiba culori diferite. Numerotand apoi culorile necesare, realizati o colorare a varfurilor grafului dat si afisati pentru fiecare culoare nodurile care se coloreaza cu ea.**

**Exemplu:   
date.in   
9 13   
1 2   
1 4   
2 3   
3 4   
3 5   
5 6   
7 8   
4 6   
7 9   
8 9   
1 3   
2 4   
date.out   
4   
culoarea 1: 1 5 7   
culoarea 2: 2 6 8   
culoarea 3: 3 9   
culoarea 4: 4**

**int a[50][50],n,m;**

**int x[50],viz[50];**

**int culori(int s)**

**{**

**int st,dr,i,x[50],j,c=1,maxx,t,gasit;**

**st=dr=1;**

**viz[s]=1;**

**x[1]=s;**

**while(st<=dr)**

**{**

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

**{**

**if(a[x[st]][i] && !viz[i])//vecinii nodului st nevizitati**

**{**

**dr++;**

**x[dr]=i;**

**maxx=0;//calculam culoarea de numar maxim pentru vecinii vizitati ai nodului i**

**for(int k=1; k<=n; k++)**

**if(a[i][k] && viz[k]>maxx) maxx=viz[k];**

**t=1;//gasim numarul culorii pentru nodul i**

**do {**

**gasit=0;**

**for(int k=1; k<=n; k++)**

**if(a[i][k] && viz[k]==t) gasit=1;**

**if(gasit) t++;**

**}**

**while(t<=maxx && gasit);**

**if(gasit) viz[i]=maxx+1;**

**else viz[i]=t;**

**}**

**}**

**st++;**

**}**

**for(i=1;i<=dr;i++) if(viz[x[i]]>c) c=viz[x[i]];**

**return c;**

**}**

**int main()**

**{**

**……………**

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

**if(!viz[i])**

**{ c=culori(i);//numarul de culori pentru component conexa**

**if(c>maxx) maxx=c;**

**}**

**g<<maxx<<"\n";**

**//afisare noduri pe culori**

**for(int i=1;i<=maxx;i++)**

**{**

**g<<"culoarea "<<i<<": ";**

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

**if(viz[j]==i) g<<j<<" ";**

**g<<endl;**

**}**

**f.close();**

**g.close(); return 0;}**

**GRAFURI ORIENTATE**

**Cuplaj maxim in graf bipartite---**infoarena.ro/problema/cuplaj

**Se dă un graf neorientat bipartit G = (V = (L, R), E). Un cuplaj în G este o submulţime de muchii M astfel încât pentru toate vârfurile v din V, există cel mult o muchie în M incidentă în v. Un cuplaj maxim este un cuplaj de cardinalitate maximă.**

**Cerinţa**

**Dându-se un graf neorientat bipartit G să se determine un cuplaj maxim.**

**Date de intrare**

**Fişierul de intrare cuplaj.in conţine pe prima linie trei numere naturale N, M şi E, unde N reprezintă cardinalul mulţimii L iar M cardinalul mulţimii R. Pe următoarele E linii se vor afla câte două numere naturale, separate între ele printr-un spaţiu, u şi v, cu semnificaţia că există muchie de la nodul u din L la nodul v din R.**

**Date de ieşire**

**În fişierul de ieşire cuplaj.out veţi afişa pe prima linie un singur număr reprezentând cuplajul maxim. Pe fiecare din următoarele linii veţi afişa câte două numere x şi y, separate între ele prin spaţiu, cu semnificaţia că nodul x din L a fost cuplat cu nodul y din R.**

ifstream fin("cuplaj.in");

ofstream fout("cuplaj.out");

**struct muchie {int x,y;};**

**int C[10001][10001];//matricea capacitatilor**

**int t1,t2,n;//cardinalele si nr total de noduri**

**int T[10001],X[10001];//vector TATA din BFS, coada din BFS**

**muchie S[10001];//solutia**

**int sk;//lungimea solutiei**

**int BFS() //returneaza 1 daca gaseste drum de crestere de la 0 la n**

**{**

**int s=1,d=1,j;**

**memset(T,0,sizeof(T));**

**memset(X,0,sizeof(X));**

**X[1]=0;T[0]=-1;**

**while(s<=d)**

**{**

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

**if(T[j]==0 && C[X[s]][j]>0)**

**{**

**X[++d]=j;**

**T[j]=X[s];**

**if(j==n)**

**return 1;**

**}**

**s++;**

**}**

**return 0;**

**}**

**void cuplaj\_maxim()**

**{**

**int j;**

**while(BFS())//cat timp mai gaseste drumuri de crestere**

**{**

**j=n;**

**while(j!=0)//inverseaza arcele**

**{ C[T[j]][j]=0;**

**C[j][T[j]]=1;**

**j=T[j];**

**}**

**}**

**}**

**int main()**

**{**

**int i,x,y,m;**

**fin>>t1>>t2>>m;**

**n=t1+t2;**

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

**{**

**fin>>x>>y;**

**C[x][t1+y]=1;**

**}**

**for(i=1;i<=t1;i++) C[0][i]=1;**

**n++;**

**for(i=1;i<=t2;i++) C[t1+i][n]=1;**

**cuplaj\_maxim();**

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

**if(C[n][i]==1) S[++sk].y=i;**

**for(i=1;i<=sk;i++)**

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

**if(C[S[i].y][j]) S[i].x=j;**

**fout<<sk<<endl;**

**for(i=1;i<=sk;i++) fout<<S[i].x<<" "<<S[i].y-t1<<endl;**

**return 0;**}

**Se da un graf orientat cu n varfuri si m arce avand arcele etichetate cu costuri numere naturale.   
Se citesc apoi doua varfuri x si y. Afisati drumul de cost minim de la varful x la varful x trecand prin varful y, precum si costul acestui drum.   
Exemplu:   
date.in   
12 21 (n,m)   
1 2 20 (arcele si costurile)   
1 3 35   
1 7 20   
2 4 30   
3 4 40   
3 6 40   
3 8 80   
4 5 25   
5 6 5   
6 8 30   
6 9 10   
7 8 15   
7 11 100   
8 9 40   
8 10 30   
8 11 35   
9 10 30   
10 12 25   
11 12 10   
10 5 15   
5 1 10   
1 10 (x,y)   
date.out   
1 7 8 10 5 1   
90**

**void dijkstra(int s)**

**{ int i,j,k,minn;**

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

**{**

**d[i]=c[s][i];**

**if(i!=s && d[i]!=inf) t[i]=s;**

**else t[i]=0;**

**p[i]=0;**

**}**

**p[s]=1;**

**for(k=1;k<n;k++)**

**{ minn=inf;**

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

**if(!p[i] && d[i]<minn)**

**{ minn=d[i]; j=i;**

**}**

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

**if(!p[i] && d[i]>d[j]+c[j][i])**

**{ d[i]=d[j]+c[j][i];**

**t[i]=j;**

**}**

**p[j]=1;**

**}**

**}**

**void drum(int i)**

**{ if(t[i]) drum(t[i]);**

**g<<i<<" ";**

**}**

**int main()**

**{ int i,v;**

**citire();**

**dijkstra(x);**

**v=d[y];**

**drum(t[y]);**

**dijkstra(y);**

**v=v+d[x];**

**drum(x);**

**g<<endl<<v;**

**Se da un graf orientat cu n vârfuri si m arce prin lista arcelor si un varf k. Calculati cate varfuri are componenta tare conexa in care se afla varful k.   
Exemplu:   
date.in   
7 10 (n,m)   
1 2   
1 3   
2 6   
3 6   
3 2   
6 1   
3 4   
4 5   
4 7   
7 4   
3 (k)   
date.out   
4**

**int k,l,m,n,a[100][100],pf[100],ps[100];**

**fstream f("date.in",ios::in);**

**fstream g("date.out",ios::out);**

**void dfsuc(int nod)**

**{int k;**

**pf[nod]=1;**

**for(k=1;k<=n;k++)**

**if(a[nod][k]==1 && pf[k]==0)**

**dfsuc(k);**

**}**

**void dfpred(int nod)**

**{int k;**

**ps[nod]=1;**

**for(k=1;k<=n;k++)**

**if(a[k][nod]==1 && ps[k]==0)**

**dfpred(k);**

**}**

**void citire()**

**{int x,y;**

**f>>n>>m;**

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

**{f>>x>>y;**

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

**}**

**f>>k>>l;**

**}**

**int main()**

**{citire();**

**dfsuc(k);**

**dfpred(k);**

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

**if(pf[i]\*ps[i]==0) pf[i]=ps[i]=0;**

**int t=0;**

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

**if(pf[i]) t++;**

**g<<t;**

**tema**

**Se da un graf orientat cu n vârfuri si m arce prin lista arcelor si doua varfuri distincte k si l.**

**Determinati daca varfurile k si l se afla in aceeasi componenta tare conexa a grafului.**

**Exemplu:**

**7 10 (n,m)**

**1 2**

**1 3**

**2 6**

**3 6**

**3 2**

**6 1**

**3 4**

**4 5**

**4 7**

**7 4**

**3 4 -> nu**

**3 6 -> da**

**Se da un graf orientat cu n vârfuri si m arce prin lista arcelor si un varf k.**

**Afisati varfurile din componenta tare conexa in care se afla varful k.**

**Exemplu:**

**date.in**

**7 10 (n,m)**

**1 2**

**1 3**

**2 6**

**3 6**

**3 2**

**6 1**

**3 4**

**4 5**

**4 7**

**7 4**

**3 (k)**

**date.out**

**1 2 3 6**

**Se citeste un graf orientat cu n noduri si m arce dat prin lista arcelor.**

**- Construiti matricea drumurilor folosind algoritmul Roy-Warshall.**

**- Folosind matricea drumurilor afisati componentele tare conexe.**

**Exemplu:**

**date.in**

**7 10**

**1 2**

**1 3**

**2 6**

**3 6**

**3 2**

**6 1**

**3 4**

**4 5**

**4 7**

**7 4**

**date.out**

**1 2 3 6**

**4 7**

**5**

**int k,m,n,x[100],a[100][100],viz[100];**

**void citire()**

**{//completez matricea de adiacenta }**

**void rw()// ALGORITMUL LUI ROY WARSHALL=> matricea drumurilor**

**{int i, j, k;**

**for(k=1;k<=n;k++)**

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

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

**if(i!=j)**

**if(a[i][j]==0)**

**a[i][j]=a[i][k]\*a[k][j];**

**}**

**void afis() // afisare componemte tare conexe**

**{for(int i=1;i<=n;i++)**

**if(!viz[i])**

**{ g<<i<<" ";**

**p[i]=1;**

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

**if(a[i][j]\*a[j][i]==1) { g<<j<<" "; viz[j]=1;}**

**g<<endl;**

**}**

**}**

**int main()**

**{citire();**

**rw();**

**afis();**

**f.close();**

**g.close();**

**return 0;**

**}**

**Se citeste un graf orientat cu n noduri si m arce dat prin vectorul arcelor. Sa se afiseze varfurile care compun cea mai mare componenta tare conexa a grafului citit (cea care are cele mai multe varfuri).**

**Exemplu:**

**date.in**

**6 7**

**1 2**

**2 1**

**3 4**

**1 5**

**4 5**

**5 6**

**6 4**

**date.out**

**4 5 6**

**int a[100][100],n,m,nrc,suc[100],pred[100];**

**void citire()**

**{int x,y;**

**f>>n>>m;**

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

**{f>>x>>y;**

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

**}**

**}**

**void dfsuc(int nod, int nrc)**

**{int k;**

**suc[nod]=nrc;**

**for(k=1;k<=n;k++)**

**if(a[nod][k]==1 && suc[k]==0)**

**dfsuc(k,nrc);**

**}**

**void dfpred(int nod,int nrc)**

**{int k;**

**pred[nod]=nrc;**

**for(k=1;k<=n;k++)**

**if(a[k][nod]==1 && pred[k]==0)**

**dfpred(k,nrc);**

**}**

**int main()**

**{int k,mk=0,c,mc;**

**citire();**

**nrc=1;**

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

**if(suc[i]==0)**

**{dfsuc(i,nrc);**

**dfpred(i,nrc);**

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

**if(suc[j]!=pred[j])**

**suc[j]=pred[j]=0;**

**nrc++;**

**}**

**for(int i=1;i<nrc;i++)**

**{k=0;**

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

**if(suc[j]==i) k++;**

**if(k>mk)**

**{**

**mk=k;**

**mc=i;**

**}**

**}**

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

**if(suc[j]==mc) g<<j<<" ";**

**f.close();**

**g.close();**

**return 0;**

**}**

**Se citeste un graf orientat cu n varfuri si m arce prin lista arcelor. Se da numar natural k mai mic decat n si k varfuri ale grafului. Afisati toate drumurile elementare care au ca extremitate initiala varful 1, ca extremitate finala varful n si care trec prin cele k varfuri citite in ordinea in care au fost citite.**

**date.in**

**7 13 3 (n,m,k)**

**1 2**

**1 3**

**2 3**

**2 6**

**2 5**

**3 1**

**3 5**

**3 6**

**4 5**

**4 7**

**5 7**

**7 6**

**6 4**

**2 6 4 (cele k varfuri)**

**date.out**

**1 2 3 6 4 5 7**

**1 2 3 6 4 7**

**1 2 6 4 5 7**

**1 2 6 4 7**

**int a[50][50],n,m,k;**

**int x[50],p[50],viz[50],b[50];**

**void afis(int n)**

**{**

**for(int i=1;i<=n;i++) fout<<x[i]<<" ";**

**fout<<endl;**

**}**

**int bun(int pas)**

**{**

**if(p[x[pas]]>=2)**

**for(int i=1;i<=k;i++)**

**if(p[b[i]]>0 && p[b[i]]<p[x[pas]] && !viz[b[i]]) return 0;**

**return 1;**

**}**

**int sol(int pas)**

**{ for(int i=1;i<=k;i++)**

**if(!viz[b[i]]) return 0;**

**return 1; }**

**void back(int k, int pas)**

**{**

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

**if(!viz[i] && a[x[pas-1]][i])**

**{**

**x[pas]=I; viz[i]=1;**

**if(bun(pas))**

**if(x[pas]==n && sol(pas)) afis(pas);**

**else back(k,pas+1);**

**viz[i]=0;**

**}**

**}**

**int main()**

**{**

**int x1,y2;**

**fin>>n>>m>>k;**

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

**{**

**fin>>x1>>x2;**

**a[x1][x2]=1;**

**}**

**for(int i=1;i<=k;i++)**

**{**

**fin>>b[i];// cele k noduri citite**

**p[b[i]]=i;**

**}**

**x[1]=1;**

**viz[1]=1;**

**back(k,2);}**

**Se citeste un graf orientat cu n varfuri si m arce prin lista arcelor. Se citeste apoi un numar natural k mai mic decat n.**

**a) Afisati toate subgrafurile cu k varfuri ale subgrafului dat.**

**b) Afisati numarul maxim de arce ale unui subgraf cu k varfuri ale grafului dat.**

**c) Afisati cate dintre subgrafurile cu k varfuri ale subgrafului dat sunt complete.**

**Exemplu:**

**date.in**

**5 9**

**1 2**

**1 3**

**3 4**

**4 1**

**2 5**

**5 3**

**3 2**

**4 2**

**2 1**

**4**

**date.out**

**varfuri: 1 2 3 4**

**arce: (1,2) (1,3) (2,1) (3,2) (3,4) (4,1) (4,2)**

**varfuri: 1 2 3 5**

**arce: (1,2) (1,3) (2,1) (2,5) (3,2) (5,3)**

**varfuri: 1 2 4 5**

**arce: (1,2) (2,1) (2,5) (4,1) (4,2)**

**varfuri: 1 3 4 5**

**arce: (1,3) (3,4) (4,1) (5,3)**

**varfuri: 2 3 4 5**

**arce: (2,5) (3,2) (3,4) (4,2) (5,3)**

**maxim de arce: 7**

**numar de subgrafuri complete: 1**

**int A[100][100], x[100],k,n,m,maxa=0,cc=0;**

**void citire()**

**{**

**int i,j,p;**

**fin>>n>>m;**

**for(p=1;p<=m;p++)**

**{**

**fin>>i>>j;**

**A[i][j]=1;**

**}**

**fin>>k;**

**}**

**void afis(int k)**

**{**

**int i,j,c=0,ok=1;**

**fout<<"varfuri: ";**

**for(i=1;i<=k;i++) fout<<x[i]<<" ";**

**fout<<endl<<"arce: ";**

**for(i=1;i<=k;i++)**

**for(j=1;j<=k;j++)**

**if(A[x[i]][x[j]])**

**{**

**fout<<"("<<x[i]<<","<<x[j]<<") ";**

**c++;**

**}**

**fout<<endl;**

**if(c>maxa) maxa=c;**

**for(i=1;i<=k;i++)**

**for(j=i+1;j<=k;j++)**

**if(A[x[i]][x[j]]+A[x[j]][x[i]]==0) ok=0;**

**if(ok) cc++;**

**}**

**void back(int k, int pas)**

**{**

**for(int i=x[pas-1]+1;i<=n;i++)**

**{**

**x[pas]=i;**

**if(pas==k) afis(k);**

**else back(k,pas+1);**

**}**

**}**

**int main()**

**{**

**citire();**

**back(k,1);**

**fout<<"maxim de arce: "<<maxa<<endl;**

**fout<<"numar de subgrafuri complete: "<<cc;}**

**Se citeste un graf orientat cu n varfuri si m arce, dat prin vestorul arcelor. Se citeste apoi un numar natural k mai mic decat n. Sa se afiseze nodurile care se afla la distanta k se nodul 1. Distanta de la un nod x la un nod y este egala cu numarul de arce din care este comviz cel mai scurt drum de la nodul x la nodul y.**

**Ex: Pentru graful din imaginea alaturata, daca se citeste k=2 se afiseaza varfurile 2, 3 si 6.**

**void bf(int r)**

**{int s,d,i;**

**s=1;d=1;**

**p[r]=1;**

**x[1]=r;**

**niv[r]=0;**

**while(s<=d)**

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

**if(!p[i] && a[x[s]][i])**

**{d++;**

**x[d]=i;**

**p[i]=1;**

**niv[i]=niv[x[s]]+1;**

**}**

**s++;**

**}**

**}**

**void main()**

**{citire();**

**bf(1);**

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

**if(niv[i]==k) g<<i<<" ";**

**}**

**Intr-o fabrica exista n sectii numerotate de la 1 la n. Sectia 1 este sectia de intrare a materiilor prime, iar sectia n cea de livrare a produselor finite. Celelalte n-2 sectii sunt sectii de productie care primesc produse intermediare de la alte sectii si livreaza la randul lor produse intermediare altora. O sectie nu poate livra un produs decat dupa ce ii sosesc produsele intermediare de la sectiile de care depinde ea. Se cunosc cele m perechi de sectii (i,j) cu semnificatia ca sectia i furnizeaza produse sectiei j si, de asemea, se cunoaste pentru fiecare astfel de pereche durata transportului intre cele 2 sectii. Se cere sa se determine durata productiei, adica timpul scurs de la inceperea functionarii sectiei 1 si pana la sosirea produselor la sectia n. (se considera ca timpul de prelucrare in fiecare sectie este 0)**

**int a[100][100],n,m,min=-16000;**

**void citire()**

**{int c,x,y,i;**

**f>>n>>m;**

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

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

**if(i==j) a[i][j]=0;**

**else a[i][j]=min;**

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

**{f>>x>>y>>c;**

**a[x][y]=c;**

**}**

**}**

**void rf()**

**{for(int k=1;k<=n;k++)**

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

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

**if(a[i][j]<a[i][k]+a[k][j] && a[i][k]!=min && a[k][j]!=min)**

**a[i][j]=a[i][k]+a[k][j];**

**}**

**void main()**

**{citire();**

**rf();**

**g<<a[1][n]<<endl;**

**}**

**Se da un graf orientat cu n varfuri si m arce, citit prin vectorul arcelor. Sa se afiseze toate circuitele euleriene pe care le are graful.**

**void circuit(int k)**

**{for(int i=1;i<=n;i++)**

**{x[k]=i;**

**if(k==1 ||a[x[k-1]][x[k]]==1)**

**{a[x[k-1]][x[k]]=0;**

**if(k==m+1) afis();**

**else back(k+1);**

**a[x[k-1]][x[k]]=1;**

**}}}**

**Se citeste un graf orientat cu n noduri si m arce dat prin vectorul arcelor. Sa se determine daca graful contine circuite.**

**int circuit(int k)**

**{int s=1,d=1;**

**x[1]=k;**

**for(int i=1;i<=n;i++) p[i]=0;**

**while(s<=d)**

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

**if(a[x[s]][i]) {d++;**

**x[d]=i;**

**if(p[k]) return 1;//am circuit**

**p[i]=1;**

**}**

**s++;**

**} return 0; }**

**int main()**

**{int c=0;**

**citire();**

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

**if(circuit(i)) c=1;**

**if(c==1) cout<<"exista";**

**else cout<<"nu exista";**

**}**

**Se citesc 2 grafuri orientate, unul cu n noduri si m arce, iar celalalt cu k varfuri si l arce, ambele date prin vectorul arcelor. Sa se determine daca al doilea graf este subgraf al primului.**

**Drumuri de lungime minima intre oricare 2 noduri - algoritmul Roy-Floyd**

**Se citeste un graf orientat cu n noduri si m arce dat prin vectorul arcelor. Sa se calculeze lungimea minima a drumului dintre oricare doua noduri din graf.**

**Ex: Pentru graful alaturat matricea lungimilor minime este:**

**0 2 = = 1 =**

**1 0 = = 2 =**

**3 3 0 2 2 1**

**1 3 1 0 2 2**

**2 1 = = 0 =**

**2 2 2 1 1 0**

**(= reprezinta faptul ca nu exista drum)**

**void citire()**

**{ int i,j,x,y;**

**f>>n>>m;**

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

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

**if(i==j) c[i][j]=0;**

**else c[i][j]=max;**

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

**{ f>>x>>y;**

**c[x][y]=1; }}**

**void rf()**

**{ int i,j,k;**

**for(k=1;k<=n;k++)**

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

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

**if(c[i][j]>c[i][k]+c[k][j])**

**c[i][j]=c[i][k]+c[k][j];}**

**void afis()**

**{ int i,j;**

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

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

**if(c[i][j]==max) g<<"- ";**

**else g<<c[i][j]<<" ";**

**g<<endl; }}**

**int main()**

**{ citire(); afis(); rf(); g<<endl; afis();**

**}**

**algoritmul lui Dijkstra**

**oid citire()**

**{ int i,j,x,y,cost;**

**f>>n>>m;**

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

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

**if(i==j) c[i][j]=0;**

**else c[i][j]=max;**

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

**{ f>>x>>y>>cost;**

**c[x][y]=cost;**

**}**

**}**

**void dijkstra(int s)**

**{ int i,j,k,min;**

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

**{ d[i]=c[s][i];**

**if(i!=s && d[i]!=max) t[i]=s;**

**}**

**p[s]=1;**

**for(k=1;k<n;k++)**

**{ min=max;**

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

**if(!p[i] && d[i]<min) { min=d[i]; j=i;**

**}**

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

**if(!p[i])**

**if(d[i]>d[j]+c[j][i])**

**{ d[i]=d[j]+c[j][i];**

**t[i]=j;**

**}**

**p[j]=1;**

**}**

**}**

**void drum(int i)**

**{ if(t[i]) drum(t[i]);**

**g<<i<<" ";**

**}**

**void main()**

**{ citire();**

**f>>s;**

**dijkstra(s);**

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

**if(i!=s)**

**{g<<s<<"->"<<i<<": ";**

**drum(i);**

**g<<"="<<d[i]<<endl;**

**}**

**}**