TEOREME DIN TEORIA GRAFURILOR

Teorema 1: Dandu-se un graf cu n noduri avand matricea de adiacenta A, atunci elementele Aij^k din A^k reprezinta numarul tuturor drumurilor de lungime k dintre i si j.

Demonstratie

Aij^1 reprezinta toate drumurile de lungime 1 dintre i si j.
Presupunem ca afirmatia e adevarata pentru k-1, adica Aij^{k-1} reprezinta numarul drumurilor de lungime k-1 dintre i si j.
$Aijk=Ai1^{k-1}*A1j+Ai2^{k-1}*A2j+Ai3^{k-1}*A3j+….+Ain^{k-1}*Anj$
$Aip^{k-1}*Apj$ – reprezinta numarul drumurilor de lungime k ce trec prin nodul p, din care rezulta ca Aij^k reprezinta numarul drumurilor de lungime k dintre i si j.

Teorema 2: Daca intre doua noduri a si b exista un lant cu lungimea mai mare strict decat n-1, atunci sigur exista un lant de lungime mai mica sau egala cu n-1 intre a si b

Demonstratie

$a – xj_1 – xj_2 – xj_3 - …..-xj_{p-1} – b$ lant ce are lungimea p>n .
$a , xj_1 , xj_2 , xj_3 , …..,xj_{p-1} , b$ €{x1,x2,…,xn}
$a , xj_1 , xj_2 , xj_3 , …..,xj_{p-1} , b$ sunt in numar de p+1 → exista cel putin doua care se repeata si acestea nu pot fi alaturate. Putem sa le eliminam noduri pana cand toate sunt distincte doua cate doua. In final drumul este elementar cu cel mult n noduri, adica drumul are lungimea cel mult n.

Consecinta: Daca intre a si b nu exista drum de lungime n, atunci intre cele doua noduri nu exista drumuri .

Demonstratie

Daca ar exista drum de lungime mai mare strict decat n, atunci exista drum de lungime mai mica sau egala cu n.

Definitie: Se numeste matrice a drumurilor intr-un graf, o matrice patratica de ordin n (n numarul de noduri),astfel incat Dij=1 daca exista drum intre i si j si Dij=0 daca nu exista drum intre i si j.

$B=A^1+A^2+…+A^n$, atunci Bij reprezinta numarul tuturor drumurilor dintre I si j care au lungimile mai mici sau egale cu n.

Dij=1, daca Bij≠0 si Dij=0 daca Bij=0

Aflarea matricei D:

```
#include <iostream>
#include<fstream>
#include<conio.h>
#include<ctype.h>
using namespace std;
int n,m,i,j,k,p,a[100][100],b[100][100],d[100][100],e[100][100],x,y;

void citire()
{
   ifstream f("arce.txt");
   f>>n>>m;
   for(i=1;i<=m;i++)
   {
      f>>x>>y;a[x][y]=b[x][y]=a[y][x]=b[y][x]=1;
   }
   f.close();
}

void calcul()
{
  {

  for(k=1;k<=n-1;k++)
     for(i=1;i<=n;i++)
       for(j=1;j<=n;j++)
       {   e[i][j]=b[i][j];
          for(p=1;p<=n;p++)
            e[i][j]=e[i][j]+b[i][p]*a[p][j];
       }
   for(int s=1;s<=n;s++)
         for(int t=1;t<=n;t++)
            b[s][t]=e[s][t];

  }
```

```
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           if(e[i][j]!=0) d[i][j]=1;
                else d[i][j]=0;
}

void afisare()
{
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
       cout<<b[i][j]<<' ';
 cout<<endl;
}
}


int main()
{   char ch;
   citire();
   calcul();
   afisare();
    ch=getchar();
    return 0;
}
```

Este de ordinal $O(n^4)$

Algoritmul Roy-Warshall are ordinul $O(n^3)$

Algoritmul Roy-Warshall de determinare a matricei drumurilor

Matricea drumurilor se obţine aplicând matricei de adiacenţă transformări succesive. Vom spune că există drum de la nodul i la nodul j, dacă găsim un nod k (diferit de i, j) cu proprietatea că există drum de la i la k şi drum de la j la k. Astfel:

- un element a[i, j] care este 0, devine 1, dacă există un nod k astfel încât a[i][k]=1 şi a[k][j]=1. Pentru a găsi arcele nodului k, trebuie parcurse pe rând în varianta k toate nodurile 1, 2, …, n.

```
for(k=1;k<=n-1;k++)
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
        if(!a[i][j])
           a[i][j]=b[i][k]*a[k][j];
```

Teoreme:

1) Numărul minim de muchii necesare pentru ca un graf neorientat să fie conex este: n-1,unde n reprezintă numărul de noduri.
2) Un graf conex cu n noduri şi n-1 muchii este aciclic şi maximal în raport cu această proprietate.
3) Un graf aciclic cu n noduri şi n-1 muchii este conex şi maximal în raport cu această proprietate.
4) Dacă un graf neorientat conex are n noduri şi m muchii, numărul de muchii care trebuie eliminate pentru a obţine un graf parţial conex aciclic este: m-n+1.
5) Dacă un graf are n noduri şi m muchii şi p componente conexe, numărul de muchii care trebuie eliminat pentru a obţine un graf parţial aciclic(arbore) este egal cu: m-n+p.
6) Pentru a obţine dintr-un graf neorientat conex 2 componente conexe numărul minim de muchii care trebuie eliminate este egal cu gradul minim din graf.
7) Numărul maxim de muchii dintr-un graf cu n noduri şi p componente conexe este [(n-p)(n-p+1)]/2.

1)----------

G=(N,M) N={1,2,….,n} Pp. ca G conex si are p muchii cu p<n-1

n=2 →m=n-1=1

Pp. ca daca N are n-1 noduri atunci m=n-2

Fie G a.i. N are n noduri si e conex. Luam G' care are nodurile N-{n} . Numarul minim de muchii pentru G' este n-2 pentru a fi conex. Evident ca G trebuie sa aiba minim n-1 muchii pentru a fi conex.

2)----------

G=(N,M) N={1,2,….,n} Pp. ca G conex si n-1 muchii.

Daca G este cyclic insemna ca avem x1,x2,…,xp,x1 ciclu cu p>=1. Eliminam muchia [xp,x1] eliminam ciclul si graful ramane conex. Contrazice faptul ca G trebuie sa aiba minim n-1 muchii. El este acyclic maximal, adica orice muchie adaugam el devine cyclic.(evident)

3)----------

G=(N,M) N={1,2,….,n} Pp. ca G aciclic si n-1 muchii.
Daca G este acyclic si are n-1 muchii. Daca n-ar fi conex inseamna ca exista cel putin un nod izolat. Eliminam toate nodurile isolate in numar de p>=1 si obtinem un subgraf cu n-p noduri n-1 muchii acyclic si conex →are cel putin n-p-1 muchii>=n-1 →p=0 →G conex
4)----------
Evident ca eliminam m-n+1 muchii pentru a obtine un graf partial conex.
m-(m-n+1)=n-1
5)----------
p component conexe
m1,m2,…,mp numarul de muchii pentru comp conexe c1, c2 s.a.m.d mp
Eliminam n1-m1+1,….np-mp+1
Daca eliminam n-m+p obtinem un graf partial acyclic cu p component conexe
6)----------
n-m+2 numarul minim de muchii pentru a ramane acyclic cu 2 comp conexe

) Pentru a obţine dintr-un graf neorientat conex 2 componente conexe numărul minim de muchii care trebuie eliminate este egal cu gradul minim din graf.

7)----------

7) Numărul maxim de muchii dintr-un graf cu n noduri şi p componente conexe este [(n-p)(n-p+1)]/2.

c1,c2,..,cp n1+n2+…+np=n

n1(n1-1)/2+n2(n2-1)/2+…+np(np-1)/2=

Algoritmul Roy-Floyd pentru determinarea matricei costurilor minime intr-un graf ponderat

Fie **G=(V, E)** un graf neorientat, unde V are n elemente (n noduri) si E are m elemente (m muchii) memorat prin matricea ponderilor. Se cere ca pentru doua noduri x,y citite sa se determine lungimea minima a lantului de la x la y. Fie **G=(V, E)** un graf neorientat, unde V are n elemente (n noduri) si E are m elemente (m muchii) memorat prin matricea ponderilor. Se cere ca pentru doua noduri x,y citite sa se determine lungimea minima a lantului de la x la y.
Algoritmul:
-se genereaza matricea ponderilor:
a[i,j]=0 daca i=j
a[i,j]=c daca exista muchie de cost c intre i si j
a[i,j]=pinf daca nu exista muchie intre i si j
pinf=constanta cu o valoare foarte mare
-se incearca pentru oricare pereche de noduri i,j sa se obtina "drumuri" mai scurte prin noduri intermediare k ($k \in 1 \ldots n$).

Acest lucru se determina comparand "lungimea" lantului a[i,j] cu lungimea lantului care trece prin k si daca:

a[i,j] > a[i,k]+a[k,j] atunci se atribuie: a[i,j] $\leftarrow$ a[i,k]+a[k,j]

Astfel generarea matricii drumurilor optime se realizeaza cu urmatoarea secventa:

```
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];
```

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

```
using namespace std;
const int pinf=1000;  //pentru plus infinit
int a[20][20],n,m;


void citire_cost()
{
 int i,j,x,y,c;
ifstream f("roy.txt");
 f>>n>>m;
 //initializare matrice
 for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
      if(i==j)
            a[i][j]=0;
       else
            a[i][j]=pinf;
  for(i=1;i<=m;i++)
     {f>>x>>y>>c;
     a[x][y]=a[y][x]=c;}
}

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

void genarare_matrice_drumuri_optime()
{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) //realizeaza descompunerea portiunii de la
i la j prin k
{int g=0,k=1;
 while(k<=n&&!g)
           {if(i!=k&&j!=k)
              if(a[i][j]==a[i][k]+a[k][j])
                       {descompun_drum(i,k);
```

```
                    descompun_drum(k,j);
                    g=1;} //g marcheaza daca se poate realiza descompune
          k++;
          }
  if(!g)
    cout<<j<<" ";   //cand "drumul" nu mai poate fi descompus afisez
extremitatea finala

}

void scriu_drum(int nodini,int nodfin) // functia primeste ca parametri cele
doua noduri pt care se determina optimul
{if(a[nodini][nodfin]<pinf)
    {cout<<"lantul de la "<<nodini<<" la "<<nodfin<<" are lungimea
"<<a[nodini][nodfin];
     cout<<endl<<"un drum optim este: "<<endl;
     cout<<nodini<<" ";
     descompun_drum(nodini,nodfin); // apeleaza functia care afiseaza
efectiv lantul
    }
 else
    cout<<"nu exista drum de la "<<nodini<<" la "<<nodfin;
}


int main()
{int x,y;
citire_cost();
cout<<endl<<"matricea ponderilor "<<endl;
afisare_mat();
genarare_matrice_drumuri_optime();
cout<<endl<<"matricea drumurilor optime "<<endl;
afisare_mat();
cout<<endl<<"Se determina drumul minim intre varfurile x si y "<<endl;
cout<<"x=";
cin>>x;
cout<<"y=";
cin>>y;
scriu_drum(x,y);
getch();
}
```

Algoritmul Floyd-Warshall pentru determinarea matricei costurilor minime intr-un graf ponderat

Avem o problema clasica de grafuri. Se da un graf orientat cu N noduri, memorat prin matricea ponderilor. Determinati pentru orice pereche de noduri x si y lungimea minima a drumului de la nodul x la nodul y, afisandu-se matricea drumurilor minime. Prin lungimea unui drum intelegem suma costurilor arcelor care-l alcatuiesc. Problema se gaseste pe infoarena.
Aceasta problema ar putea fi rezolvata prin algoritmi clasici pentru determinarea unui drum de cost minim intre doua noduri in complexitate O(n^3*logn) – cel mai optim. Totusi, algoritmul Floyd-Warshall (sau chiar Roy-Floyd) rezolva problema prin programare dinamica in complexitate n^3. Consideram toate varfurile intermediare k(k=1,2,â€¦,n) si verificam pentru fiecare pereche de varfuri din graf x,y daca trecand prin k se optimizeaza costul de la x la y; in caz afirmativ, costul drumului de la x la y se modifica. Observam ca dupa pasul k in matricea costurilor, pe pozitia (x,y), se afla costul drumului de cost minim de la x la y care trece numai prin varfuri intermediare din multimea {1,2,â€¦,k}.

```
#include<iostream>
#include<fstream>
 using namespace std;
void read(),solve(),show();
int n,a[105][105];
int main()
{
   read();
   solve();
   show();
   return 0;
}
void read()
{
   int j;
   ifstream f(“arce.txt”);

   f>>n;
   for(int i = 0; i < n; i++)
      for(j = 0; j < n; j++)
         f>>a[i][j];
```

```
}

void solve()
{
   int i,j;
   for(int k = 0; k < n; k++)
      for(i = 0; i < n; i++)
         for(j = 0; j < n; j++) //daca putem actualiza
            if(a[i][k] && a[k][j] && i != j && (a[i][j] > a[i][k] + a[k][j] ))
               a[i][j] = a[i][k] + a[k][j];
}
void show()
{
   int j; //afisam matricea costurilor
   for(int i = 0; i < n; i++,printf("\n"))
  {      for(j = 0 ; j < n; j++)
          cout<<a[i][j]<<"  ";
          cout<<endl
}
}
```

Arbori partiali de cost minim

1 Algoritmul lui Kruskal

Arborele partial de cost minim poate fi construit muchie cu muchie, dupa urmatoarea metoda a lui Kruskal (1956): se alege intai muchia de cost minim, iar apoi se adauga repetat muchia de cost minim nealeasa anterior si care nu formeaza cu precedentele un ciclu. Alegem astfel X–1 muchii. Este usor de dedus ca obtinem in final un arbore. Este insa acesta chiar arborele partial de cost minim cautat? Se ordoneaza crescator dupa cost muchiile.

```
#include <iostream>
#include<fstream>
using namespace std;

struct arc
{
   int x,y,c;
}aux;
int i,j,n,m,c_min,cost,v,p,arb[100];
arc arce[100];
void citeste()
{
   ifstream f("graf.txt");
   f>>n>>m>>v;
   for(i=1;i<=m;i++)
    {
         f>>arce[i].x>>arce[i].y>>arce[i].c;

    }
    f.close();

}

void sortare()
{
 for(i=1;i<=m-1;i++)
   for(j=i+1;j<=m;j++)
     if(arce[i].c>arce[j].c)
 {char ch;
    aux=arce[i];arce[i]=arce[j];arce[j]=aux;
 }


}

void kruskal()
{int mx=1,mare,mic;

arb[arce[1].x]=arb[arce[1].y]=1;
```

```
i=2;c_min=arce[1].c;
while(i<=n-1)
{
  if(arb[arce[i].x]==0&&arb[arce[i].y]==0)
  {
     mx++;
     arb[ arb[arce[i].x]]=arb[arce[i].y]=mx;c_min=c_min+arce[i].c;
  }
  else
  if(arb[arce[i].x]==0&&arb[arce[i].y]!=0)
  {
     arb[arce[i].x]=arb[arce[i].y];c_min=c_min+arce[i].c;
  }
else
  if(arb[arce[i].x]!=0&&arb[arce[i].y]==0)
  {
     arb[arce[i].y]=arb[arce[i].x];c_min=c_min+arce[i].c;
  }
  else
  if(arb[arce[i].x]!=arb[arce[i].y])
  {  if(arb[arce[i].x]>arb[arce[i].y]) {mare=arb[arce[i].x];mic=arb[arce[i].y];}
            else {mare=arb[arce[i].y];mic=arb[arce[i].x];}
               for(int t=1;t<=n;t++)
                  if(arb[t]==mic) arb[t]=mare;

  c_min=c_min+arce[i].c;
  }
  i++;
}

}


void tipar()
{
cout<<"COSTUL MINIM ESTE: "<<c_min<<endl;
for(i=1;i<=n-1;i++)
cout<<'('<<arce[i].x<<','<<arce[i].y<<")="<<arce[i].c<<endl;
}
```

```
int main()
{
   citeste();
   sortare();
   kruskal();
   tipar();
   return 0;
}
```

2 Algoritmul lui Prim

Se pleaca de la un nod initial. Se adauga la graful arborele format un alt nod, care sa reprezinte muchia de cost minim dintre nodurile arborelui si nodurile care nu au fost atasate. Algoritmul se termina in n-1 pasi

```
#include <iostream>
#include<fstream>
#include<conio.h>
using namespace std;

int i,j,n,m,c,c_min,min1,v,p1,p2,a[50][50],s[100],t[100],x,y,poz,k;
const int pinf=60000;
char ch;
void citeste()
{    ifstream f("graf.txt");
   f>>n>>m>>v;
   for(i=1;i<=m;i++)
    {
         f>>x>>y>>c;
        a[x][y]=a[y][x]=c;
    }

    f.close();
}


void prim()
{   c_min=0;
```

```
   for(k=1;k<=n-1;k++)
   {  min1=pinf;
      for(i=1;i<=n;i++)
      if(s[i]!=0)
      {
         for(j=1;j<=n;j++)
            if(s[j]==0&&min1>a[i][j]&&a[i][j]!=0)
            {
               min1=a[i][j];p1=i;p2=j;

            }

      }
      cout<<p1<<','<<p2<<endl;ch=getchar();
      t[p1]=p2;c_min=c_min+min1;s[p2]=1;
   }

}

int main()
{
   citeste();
   s[v]=1;
 prim();
 cout<<"COSTUL MINIM ESTE: "<<c_min<<endl;

   return 0;
}.
```

2 Algoritmul lui Lee

Se face o parcurgere a unui arbore fara sa-l constrim pana cand gasim solutia

Exemplu: Problema LABIRINTULUI

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream f("labirint.in");
ofstream g("labirint.out");
void bordare (int m, int n,int a[100][100])
{
    int k;
    for (k=0;k<=n;k++)
    {
        a[0][k]=0;
    }
    for (k=0;k<=n;k++)
    {
        a[n+1][k]=0;
    }
    for (k=n+1;k>=1;k--)
    {
        a[k][m+1]=0;
    }
    for (k=m+1;k>=1;k--)
    {
        a[k][0]=0;
    }
}
int main ()
{
    int a[100][100],b[100][100]={0},d1[]={0,1,0,-
1},d2[]={-1,0,1,0},l[1000],c[1000],x,y,x1,y1,i,j,m,n,k;
    f>>m>>n;
    for (i=1;i<=m;i++)
    {
        for (j=1;j<=n;j++)
        {
```

```
            f>>a[i][j];
        }
    }
    bordare (m,n,a);
    for (i=1;i<=m;i++)
    {
        for (j=1;j<=n;j++)
        {
            if (a[i][j]==-1) {l[0]=i;c[0]=j;}
        }
    }
    b[l[0]][c[0]]=a[l[0]][c[0]];
    i=j=0;
    while (i<=j)
    {
        x=l[i];
        y=c[i];
        for (k=0;j<4;k++)
        {
            x1=x+d1[k];
            y1=y+d2[k];
            if (a[x1][y1]!=0)
            {
                if (b[x1][y1]==0)
                {
                    b[x1][y1]=a[x1][y1]+b[x][y];
                    j++;
                    l[j]=x1;
                    c[j]=y1;
                }
                else if (b[x][y]+a[x1][y1]<b[x1][y1])
                {
                    b[x1][y1]=b[x][y]+a[x1][y1];
                    j++;
                    l[j]=x1;c[j]=y1;
                }
            }
        }
        i++;
    }
```

```
    for (i=0;i<=m+1;i++)
    {
        for (j=0;j<=n+1;j++)
        {
            cout<<b[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

1. Parcurgerea arborilor binari

RSD – preordine
SRD – inordine ---------- parcurgere in latime
SDR –postordine ---------- parcurgere in latime

RSD	

#include <iostream>
#include <fstream>
using namespace std;
 	
int n,v, s[100],d[100],t[100];
void citire()
{
ifstream f("arb.txt");
f>>n>>v;
for(int i=1;i<=n;i++)
  f>>s[i];
for(int i=1;i<=n;i++)
  f>>d[i];
}
 	
void rsd(int v)
{
  cout<<v<<',';
  if(s[v]!=0)rsd(s[v]);
  if(d[v]!=0) rsd(d[v]);
 	
}
	Parcurgere in adancime
	#include <iostream>
	#include <fstream>
	using namespace std;
	
	int n,v, s[100],d[100],st[100],viz[100],k,a;
	void citire()
	{
	ifstream f("arb.txt");
	f>>n>>v;
	for(int i=1;i<=n;i++)
	  f>>s[i];
	for(int i=1;i<=n;i++)
	  f>>d[i];
	}
	
	void rsd()
	{
	 k=1;
	 st[k]=v;viz[v]=1;
	 while(k>0)

```
int main ()
{
   citire();
   for(int i=1;i<=n;i++)
   cout<<s[i]<<',';
   cout<<endl;
for(int i=1;i<=n;i++)
   cout<<d[i]<<',';
   rsd(v);
}
```

```
 {
    a=st[k];k--;cout<<a<<",";
    if(d[a]!=0&&viz[d[a]]==0)
{k++;st[k]=d[a];viz[d[a]]=1;}
    if(s[a]!=0&&viz[s[a]]==0)
{k++;st[k]=s[a];viz[s[a]]=1;}
 }
}

int main ()
{
   citire();
      rsd();
}
```