# Drumuri minime în grafuri orientate. Algoritmul lui Dijkstra

Fie **G=(X,U)** şi **l:U🡪R+**. Se pune problema determinării unor drumuri de lungime minimă în acest graf. Rezolvarea ei are multiple aplicaţii practice. Considerând drept noduri diferite puncte dintr-un oraş, dacă ponderea **l(u),** unde **u=(xi,xj)**, reprezintă durata de trecere de la **xi** la **xj**, problema revine la determinarea drumurilor de durată minimă; dacă **l(u)** reprezintă costul transportului de la **xi** la **xj**, problema revine la determinarea drumurilor având costul de transport minim etc.

Pentru tratarea problemelor de minim vom asocia grafului **G** matricea costurilor în forma a), adică **C=(cij)nxn** definită astfel:

 **l(xi,xj) dacă (xi,xj)∈U**

**cij= 0 dacă i=j**

 **+∞ dacă (xi,xj)∉U**

**Problemă:** *Fiind dat un graf orientat* ***G=(X,U****), o funcţie* ***l:U🡪R+*** *şi un nod* ***x0****, să se determine pentru toate vârfurile* ***xi*** *pentru care există drum de la* ***x0*** *la* ***xi****, lungimea celui mai scurt drum şi unul dintre drumurile minime de la* ***x0*** *la* ***xi***.

 Algoritmul utilizrează metoda Greedy generând drumurile minime în ordinea crescătoare a lungimilor.

 **2 7 3**

 **1 1 2 1**

 **1 3 5**

 **4**

 **1 3**

 **6**

**Exemplu**: Pentru graful din figura alăturată, considerând nodul de plecare 1, se vor obţine în ordine:

D1=(1,2) de lungime 1

D2=(1,2,5) de lungime 2

D3=(1,2,5,3) de lungime 4

D4=(1,2,5,3,4) de lungime 5

De la 1 la 6 nu există drum.

Se porneşte din **x0**. Evident cel mai scurt drum de la **x0** la unul din celelalte vârfuri ale grafului este dat de arcul **(x0,xj)** de lungime minimă. Următorul drum în ordinea lungimilor va fi dat fie de un alt arc cu extremitatea iniţială **x0**, fie de un drum **(x0,xj,xp)**. Alegem în continuare drumuri în ordinea crescătoare a lungimilor, până când am determinat drumuri minime de la **x0** către toate vârfurile pentru care există drum pornind din **x0**. Pentru aceasta se consideră **S** mulţimea vârfurilor **xj∈X** pentru care am găsit drum minim de la **x0** la **xj**. Iniţial **S={x0}**. La fiecare pas, adăgăm în **S** acel nod **xk∈X-S** cu proprietatea că drumul minim de la **x0** la **xk** are cel mai mic cost dintre toate drumurile de la **x0** la **xp**, cu **xp∈X-S**. Pentru exemplul considerat **S** va avea pe rând următorul conţinut:

**S={1}**

**S={1,2}**

**S={1,2,5}**

**S={1,2,5,3}**

**S={1,2,5,3,4}**

Se observă că drumul de la **x0** la **xk** (nodul ce urmează să-l adăugăm în **S** la un moment dat) trece numai prin vârfuri din **S** (cu excepţia lui **xk**). Pentru a alege nodul **xk∈X-S** ce urmează a fi adăgat în **S** vom folosi un vector **D=(d1,d2,...,dn)** astfel încât:

 **lungimea drumului minim de la x0 la xi, dacă xi∈S**

**di=**

 **lungimea drumului minim de la x0 la xi ce foloseşte numai vârfuri din**

 **S, dacă xi∉S**

Iniţial **di=C(x0,i), (∀)i=1..n**, adică este linia **x0** din matricea costurilor. La un moment dat, adăgăm în **S** nodul **xk** cu proprietatea că **dk=min{dj | xj∈X-S}**. După adăgarea lui **xk** în **S** trebuie actualizate valorile lui **d** pentru elementele care nu sunt în **S**, deoarece este posibil ca drumul minim de la **x0** la unul dintre aceste noduri (folosind noduri din **S**) să folosească nodul **xk** pe care tocmai l-am adăgat. Drumul minim de la **x0** la **xj** ce foloseşte noduri din **S** (inclusiv **xk**) va fi de forma **(x0,…,xk,xj)**. Deci pentru **xj∈X-S**, **dj** se modifică după adăugarea lui **xk** la **S** numai dacă **dk+C(k,j)<dj**, caz în care **dj🡨 dk+C(k,j)**. În final, vectorul **d** va conţine costurile (lungimile) drumurilor minime de la **x0** la celelalte noduri; dacă pentru un nod **xj** nu există drum de la **x0** la **xj**, atunci **dj=∞**.

 Pentru a reţine şi drumurile minime (nu numai lungimile lor) vom considera un vector numit **T** (tată) care reţine indicele precedentului fiecărui nod în drumul minim de la **x0** la acel nod.

Iniţial:

 **0 dacă i=x0 sau C(x0,i)=∞**

 **Ti=**

 **x0 dacă C(x0,i)≠∞ şi i≠x0**

La fiecare actualizare de forma **dj🡨 dk+C(k,j)** vom avea şi o actualizare a vectorului **T** de forma **Tj🡨k**.

 Algoritmul se încheie când **S** conţine toate nodurile **xj** pentru care există drum de la **x0** la **xj**, deci fie când **S=X** (dacă există drumuri de la **x0** la toate celelalte noduri), fie când mulţimea **X-S** cuprinde numai noduri pentru care nu există drumuri pornind din **x0** la ele (caz în care **min{dj | xj∈X-S}=∞**). Pentru reprezentarea mulţimii **S** se poate folosi vectorul caracteristic **S** cu **n** componente definit astfel:

 **0 dacă xi∉S**

 **Si=**

1. **dacă xi∈S**

**Exemplu**: Considerăm graful anterior şi punctul **x0=1**. Cei trei vectori au valorile iniţiale:

 **D=(0, 1, ∞, ∞, 3, ∞)**

 **T=(0, 1, 0, 0, 1, 0)**

 **S=(1, 0, 0, 0, 0, 0)**

1. determinăm **min(D[i] | S[i]=0) => min=1,k=2 => S[2]🡨1**

Actualizăm distanţele la nodurile neselectate încă.

**S[3]=0; D[3]= ∞ > D[2]+C[2,3]=1+7=8 => D[3]🡨8, T[3]🡨2**

**S[4]=0; D[4]= ∞ D[2]+C[2,4]=1+∞** , nu se poate actualiza

**S[5]=0; D[5]=3 > D[2]+C[2,5]=1+1=2 => D[5]🡨2, T[5]🡨2**

**S[6]= ∞; D[6]= ∞ D[2]+C[2,6]=1+∞** , nu se poate actualiza

După primul pas configuraţia celor trei vectori este:

 **D=(0, 1, 8, ∞, 2, ∞)**

 **T=(0, 1, 2, 0, 2, 0)**

 **S=(1, 1, 0, 0, 0, 0)**

1. determinăm **min(D[i] | S[i]=0) => min=2, k=5 => S[5]🡨1**

Actualizăm distanţele la nodurile neselectate încă.

**S[3]=0; D[3]=8 > D[5]+C[5,3]=2+2=4 => D[3]🡨4, T[3]🡨5**

**S[4]=0; D[4]= ∞ D[5]+C[5,4]=2+∞**, nu se poate actualiza

**S[6]=0; D[6]= ∞ D[5]+C[5,6]=2+∞**, nu se poate actualiza

După al doilea pas configuraţia celor trei vectori este:

 **D=(0, 1, 4, ∞, 2, ∞)**

 **T=(0, 1, 5, 0, 2, 0)**

 **S=(1, 1, 0, 0, 1, 0)**

1. determinăm **min(D[i] | S[i]=0) => min=4, k=3 => S[3]🡨1**

Actualizăm distanţele la nodurile neselectate încă.

**S[4]=0; D[4]= ∞ > D[3]+C[3,4]=4+1=5 => D[4]🡨5, T[4]🡨3**

**S[6]=0; D[6]= ∞ D[3]+C[3,6]=4+∞**, nu se poate actualiza

După al treilea pas configuraţia celor trei vectori este:

 **D=(0, 1, 4, 5, 2, ∞)**

 **T=(0, 1, 5, 3, 2, 0)**

 **S=(1, 1, 1, 0, 1, 0)**

1. determinăm **min(D[i] | S[i]=0) => min=5, k=4 => S[4]🡨1**

Actualizăm distanţele la nodurile neselectate încă.

**S[6]=0; D[6]= ∞ D[4]+C[4,6]=5+∞**, nu se poate actualiza

După al patrulea pas configuraţia celor trei vectori este:

 **D=(0, 1, 4, 5, 2, ∞)**

 **T=(0, 1, 5, 3, 2, 0)**

 **S=(1, 1, 1, 1, 1, 0)**

1. determinăm **min(D[i] | S[i]=0) => min=∞** şi algoritmul se încheie pentru că nu există nici-un drum de la nodul 1 la nodul 6

Programul care implementează algoritmul lui Dijkstra este următorul:

**#include<stdio.h>**

**#define N 20**

**#define INF 1<<14**

**void citire(int c[N][N],int &n,int &xp)**

**{**

 **int i,j,x,y,z; FILE \*f=fopen(“graf.txt”,”r);**

 **fscanf(f,"%d %d",&n,&xp);** //numarul de noduri si nodul de plecare

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

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

 **c[i][j]=INF;**

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

 **while(!feof(f))**

 **{**

**fscanf("%d %d %d",&x,&y,&z);** //arcul si costul sau

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

 **}**

**fclose(f);**

**}**

**void minim(int S[N],int D[N],int n,int &q)**

**{**

 **int i; long m;**

 **m=2\*INF;**

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

 **if(!S[i]&&D[i]<m)**

 **{**

 **m=D[i]; q=i;**

 **}**

**}**

**void determinare\_drumuri(int c[N][N],int D[N],int T[N],int n)**

**{**

 **int S[N],i,x,j,k,ok;**

 //initializari

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

 **{**

 **D[i]=c[xp][i]; S[i]=0;**

 **if(c[xp][i]<INF) T[i]=xp;**

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

 **}**

 **S[xp]=1; T[xp]=0; ok=1; x=0;**

 **do{**

 **minim(S,T,n,k);** //determina nodul k aflat la distanta minima

 **x++;**

 **if(D[k]==INF||x==n)**

 **ok=0;** //nu mai pot fi construite drumuri minime

 **else**

 **{**

 //actualizam vectorii S,T si D

 **S[k]=1;**

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

 **if(!S[j]&&D[j]>D[k]+c[k][j])**

 **{**

 **D[j]=D[k]+c[k][j];**

 **T[j]=k;**

 **}**

 **}**

 **}while(ok);**

**}**

**void drum(int D[N],int T[N],int i)**

**{**

 **if(i)**

 **{**

 **drum(D,T,T[i]);**

 **printf("%d ",i);**

 **}**

**}**

**void afisare\_drumuri(int D[N],int T[N],int n)**

**{**

 **int i;**

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

 **if(i!=xp)**

 **if(D[i]==INF)**

 **printf("\nNu exista drum de la %d la %d\n",xp,i);**

 **else**

 **{**

 **printf("\nDrumul minim de la %d la %d: ",xp,i);**

 **drum(D,T,i); printf("\n");**

 **printf("\tLungimea drumului este %d\n",D[i]);**

 **}**

**}**

**void main()**

**{**

 **int c[N][N],D[N],T[N],n,xp;**

 **citire(c,n,xp); determinare\_drumuri(c,D,T,n);**

 **afisare\_drumuri(D,T,n);**

**}**

Datorită selectării minimelor, complexitatea algoritmului este **O(n2)**.