Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatica și Microelectronica

Catedra Matematica

**RAPORT**

la lucrarea de laborator nr. 2

Disciplina: Matematica discretă

**Tema*:* Algoritmi de determinare a drumului minim-maxim cu ajutorul algoritmului Ford și Bellman-Kalaba.**

Elaborat: st.gr.AI-151 Brăduleac Vadim

Verificat: Dohotaru Leonid

Chişinău 2016

**Scopul lucrării**

Studierea algoritmilor de determinare a drumurilor minime într-un graf. Elaborarea programelor de determinare a drumului minim într-un graf ponderat.

**Sarcina lucrării**

1. Elaboraţi procedura introducerii unui graf ponderat;
2. Elaboraţi procedurile determinării drumului minim;
3. Realizaţi un program cu următoarele funcţii:

* introducerea grafului ponderat cu posibilităţi de analiză sintactică şi semantică şi de corectare a informaţiei;
* determinarea drumului minim;
* extragerea informaţiei la display şi printer (valoarea drumului minim şi succesiunea vârfurilor care formează acest drum).

**Consideraţii teoretice**

* *Noţiune de drum minim*

Pentru un graf orientat *G = (X,U)* se va numi *drum* un şir de vârfuri *D* = *(x0, x1,..., xr)* cu proprietatea că *(x0, x1)*, *(x1, x2)*,..., *(xr-1, xr)* aparţin lui *U*, deci sunt arce ale grafului şi extremitatea finală a arcului precedent coincide cu extremitatea iniţială a arcului următor.

Vârfurile *x0* şi *xr* se numesc extremităţile drumului *D*. Lungimea unui drum este dată de numărul de arce pe care le conţine. Dacă vârfurile *x0, x1,..., xr* sunt distincte două câte două drumul *D* este elementar.

Adeseori, fiecărui arc (muchii) i se pune în corespondenţă un număr real care se numeşte *ponderea* (lungimea) arcului. Lungimea arcului *(xi, xj)* se va nota *w(i,j)*, iar în cazul în care un arc este lipsă ponderea lui va fi considerată foarte mare (pentru calculator cel mai mare număr pozitiv posibil). În cazul grafurilor cu arce ponderate (grafuri ponderate) se va considera lungime a unui drum suma ponderilor arcelor care formează acest drum. Drumul care uneşte două vârfuri concrete şi are lungimea cea mai mică se va numi *drum minim* iar lungimea drumului minim vom numi *distanţă*. Vom nota distanţa dintre *x* şi *t* prin *d(x, t)*, evident, *d(x,x)=0*.

## *Algoritmul lui Ford pentru detrminarea drumului minim*

Permite determinarea drumului minim care începe cu un vârf iniţial *xi* până la oricare vârf al grafului *G*. Dacă prin *Lij* se va nota ponderea arcului *(xi, xj)* atunci algoritmul conţine următorii paşi:

1. Fiecărui vârf *xj* al grafului *G* se va ataşa un număr foarte mare *Hj(∞)*. Vârfului iniţial i se va ataşa *Ho = 0*;
2. Se vor calcula diferenţele *Hj - Hi* pentru fiecare arc *(xi, xj)*. Sunt posibile trei cazuri:
3. *Hj - Hi < Lij,*
4. *Hj - Hi = Lij,*
5. *Hj - Hi > Lij.*

Cazul "*c*" permite micşorarea distanţei dintre vârful iniţial şi *xj* din care cauză se va realiza *Hj = Hi + Lij*.

Pasul 2 se va repeta atâta timp cât vor mai exista arce pentru care are loc inegalitatea “c”. La terminare, etichetele *Hi* vor defini distanţa de la vârful iniţial până la vârful dat *xi*.

1. Acest pas presupune stabilirea secvenţei de vârfuri care va forma drumul minim. Se va

pleca de la vârful final *xj* spre cel iniţial. Predecesorul lui *xj* va fi considerat vârful *xi* pentru care va avea loc *Hj - Hi = Lij*. Dacă vor exista câteva arce pentru care are loc această relaţie se va alege la opţiune.

* *Algoritmul Bellman - Calaba*

Permite determinarea drumului minim dintre oricare vârf al grafului până la un vârf, numit vârf final.

Etapa iniţială presupune ataşarea grafului dat *G* a unei matrice ponderate de adiacenţă, care se va forma în conformitate cu următoarele:

1. *M(i,j)* = *Lij*, dacă există arcul *(xi, xj)* de pondere *Lij*;
2. *M(i,j)* = ∞, unde ∞ este un număr foarte mare (de tip întreg maximal pentru calculatorul dat), dacă arcul *(xi, xj)* este lipsă;
3. *M(i,j)* = *0*, dacă *i = j*.

La etapa a doua se va elabora un vector *V0* în felul următor:

1. *V0(i) = Lin*, dacă există arcul *(xi, xn)*, unde *xn* este vârful final pentru care se caută drumul minim, *Lin* este ponderea acestui arc;
2. *V0(i) =* ∞, dacă arcul *(xi, xn)* este lipsă;
3. *V0(i) = 0*, dacă *i = j*.

* *Algoritmul constă în calcularea iterativă a vectorului V în conformitate cu următorul procedeu:*

1. *Vk(i) = min{Vk-1; Lij+Vk-1(j)}*, unde *i = 1, 2,…, n - 1, j = 1, 2,..., n*; *i<>j;*
2. *Vk(n) = 0*.

Când se va ajunge la *Vk = Vk-1* - STOP.

Componenta cu numărul *i* a vectorului *Vk* cu valoarea diferită de zero ne va da valoarea minimă a drumului care leagă vârful *i* cu vârful *n*.

**Sarcina Individuală Varianta 3.**

F(x1)={x2,x3,x4}, F(x2)={x3,x5}, F(x3)={x4,x5,x6}, F(x4)={x5,x6}, F(x5)={x7,x8,x9}, F(x6)={x5,x8,x9}, F(x7)={x10}, F(x8)={x7,x10}, F(x9)={x8,x10}, F(x10)=ø

**Listingul programului**

#include <stdio.h>

#include <conio.h>

#include <windows.h>

#define N 50

int t\_lista[N][N], t\_pondere[N][N], v\_drum[N], n\_virfuri, v\_initial, v\_final, v\_intermediar, pon\_max=0, item, ultimul\_rind;

int menu(){

int i;

meniu:

printf("\n");

printf(" ##############################################################################\n");

printf(" # #\n");

printf(" # Alegeti algoritmul dorit: #\n");

printf(" # #\n");

printf(" # 1.Algoritmul Bellman Kalaba (drum minim) #\n");

printf(" # #\n");

printf(" # 2.Algoritmul Bellman Kalaba (drum maxim) #\n");

printf(" # #\n");

printf(" # 3.Algoritmul Ford (drum minim) #\n");

printf(" # #\n");

printf(" # 4.Algoritmul Ford (drum maxim) #\n");

printf(" # #\n");

printf(" ##############################################################################\n");

scanf("%d",&i);

while(i!=1 && i!=2 && i!=3 && i!=4){

printf(" Ati ales optiune gresita!!!\n");

Sleep(2500);

system("cls");

goto meniu;

}

return(i);

}

void citire\_lista\_adiacenta(int t[N][N], int x){

int i, j, k;

printf(" Introduceti lista de adiacenta a grafului:\n");

for(i=1; i<=x; i++){

for(j=1; j<=x; j++){

printf("F(X%d)[%d]=",i,j);

scanf("%d",&t[i][j]);

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

if(t[i][k]==t[i][j] && k!=j){

printf("Aceste arc deja a fost introdus!\n");

t[i][j]=0;

j--;

continue;

}

if(t[i][j]<0 || t[i][j]>x || t[i][j]==i){

printf("Eroare la valoare!\n");

t[i][j]=0;

j--;

continue;

}

if(t[i][j]==0) break;

}

}

}

void citire\_pondere\_lista(int t1[N][N], int t2[N][N], int x){

int i, j;

printf("\n Introduceti ponderea pentru urmatoarele arce:\n");

for(i=1; i<=x; i++)

for(j=1; j<=x; j++){

if(t1[i][j]==0) break;

printf("P(X%d,X%d)=",i,t1[i][j]);

scanf("%d",&t2[i][t1[i][j]]);

if(pon\_max<t2[i][t1[i][j]]) pon\_max=t2[i][t1[i][j]];

if(t2[i][t1[i][j]]<0){

printf("Ponderea nu poate fin negativa!\n");

j--;

}

}

for(i=1; i<=x; i++)

for(j=1; j<=x; j++)

if(t2[i][j]==0 && i!=j){

if(item==1 || item==3)

t2[i][j]=3\*pon\_max;

if(item==2 || item==4)

t2[i][j]=(-3)\*pon\_max;

}

}

void golire(int t[N][N]){

int i, j;

for(i=0; i<N; i++)

for(j=0; j<N; j++)

t[i][j]=0;

}

void golire\_vector(int v[N]){

int i;

for(i=0; i<N; i++)

v[i]=0;

}

int minim\_vector(int v[N], int \*pozitia, int x){

int i;

int minim=v[1];

\*pozitia=1;

for(i=2; i<=x; i++)

if(v[i]<minim){

minim=v[i];

\*pozitia=i;

}

return(minim);

}

int maxim\_vector(int v[N], int \*pozitia, int x){

int i;

int maxim=v[1];

\*pozitia=1;

for(i=2; i<=x; i++)

if(v[i]>maxim){

maxim=v[i];

\*pozitia=i;

}

return(maxim);

}

void adunare\_vectori(int v1[N], int v2[N], int v3[N], int x){

int i;

for(i=1; i<=x; i++)

v3[i]=v1[i]+v2[i];

}

int comparare\_vectori(int v1[N], int v2[N], int x){

int i, j;

j=0;

for(i=1; i<=x; i++)

if(v1[i]==v2[i]) j++;

if(j==x) return(1);

else return(0);

}

void transpunere\_matrice(int t[N][N], int x){

int t2[N][N], i, j;

for(i=1; i<=x; i++)

for(j=1; j<=x; j++)

t2[i][j]=t[j][i];

for(i=1; i<=x; i++)

for(j=1; j<=x; j++)

t[i][j]=t2[i][j];

}

void inversare\_vector(int v[N], int x){

int i, j=1, vec[N];

golire\_vector(vec);

for(i=x; i>=1; i--)

if(v[i]!=0){

vec[j]=v[i];

j++;

}

for(i=1; i<=x; i++)

v[i]=vec[i];

}

void completare\_liste\_min(int t[N][N], int x){

int i, j, a, vec[N];

virf\_final1:

system("cls");

if(item==1)

printf("\n Care este virful fixat(final)?\n");

if(item==3)

printf("\n Care este virful fixat(initial)?\n");

scanf("%d",&v\_final);

if(v\_final<=0 || v\_final>n\_virfuri){

printf("\n Ati introdus un virf gresit, mai introduceti odata!\n");

fflush(stdin);

Sleep(4000);

goto virf\_final1;

}

for(j=1; j<=x; j++)

t[x+1][j]=t[j][v\_final];

for(j=1; j<=x; j++){

adunare\_vectori(t[x+1],t[j],vec,x);

t[x+2][j]=minim\_vector(vec,&a,x);

}

i=x+2;

while((comparare\_vectori(t[i-1],t[i],x))==0){

for(j=0; j<=x; j++){

adunare\_vectori(t[i],t[j],vec,x);

t[i+1][j]=minim\_vector(vec,&a,x);

}

i++;

}

ultimul\_rind=i;

}

void completare\_liste\_max(int t[N][N], int x){

int i, j, a, vec[N];

virf\_final2:

system("cls");

if(item==2)

printf("\n Care este virful fixat(final)?\n");

if(item==4)

printf("\n Care este virful fixat(initial)?\n");

scanf("%d",&v\_final);

if(v\_final<=0 || v\_final>n\_virfuri){

printf("\n Ati introdus un virf gresit, mai introduceti odata!\n");

fflush(stdin);

Sleep(4000);

goto virf\_final2;

}

for(j=1; j<=x; j++)

t[x+1][j]=t[j][v\_final];

for(j=1; j<=x; j++){

adunare\_vectori(t[x+1],t[j],vec,x);

t[x+2][j]=maxim\_vector(vec,&a,x);

}

i=x+2;

while((comparare\_vectori(t[i-1],t[i],x))==0){

for(j=0; j<=x; j++){

adunare\_vectori(t[i],t[j],vec,x);

t[i+1][j]=maxim\_vector(vec,&a,x);

}

i++;

}

ultimul\_rind=i;

}

void afisare\_rezultat(int t[N][N], int x){

int i;

for(i=1; i<=x; i++)

printf("\tX%d",i);

printf("\n");

for(i=1; i<=x; i++){

if(t[ultimul\_rind][i]==(3\*pon\_max)){

printf("\tinf");

continue;

}

if(t[ultimul\_rind][i]<0){

printf("\t-inf");

continue;

}

printf("\t%d",t[ultimul\_rind][i]);

}

}

void completare\_drum\_minim(int t[N][N], int v[N], int x){

int i=1, pozitia, vec[N], v1=v\_initial, v2=v\_final;

v[1]=v\_initial;

while(v[i]!=v\_final){

adunare\_vectori(t\_pondere[ultimul\_rind],t\_pondere[v[i]],vec,x);

minim\_vector(vec,&pozitia,x);

i++;

v[i]=pozitia;

}

v\_initial=v1;

v\_final=v2;

}

void completare\_drum\_maxim(int t[N][N], int v[N], int x){

int i=1, pozitia, vec[N], v1=v\_initial, v2=v\_final;

v[1]=v\_initial;

while(v[i]!=v\_final){

adunare\_vectori(t\_pondere[ultimul\_rind],t\_pondere[v[i]],vec,x);

maxim\_vector(vec,&pozitia,x);

i++;

v[i]=pozitia;

}

v\_initial=v1;

v\_final=v2;

}

void afisare\_drum(int v[N], int x){

int i;

for(i=1; i<=x; i++){

if(v[i+1]==0){

printf("X%d.",v[i]);

break;

}

else

printf("X%d, ",v[i]);

}

}

while(r!=n)

{r=0;

max=-10000;

k=1;

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

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

if(b[i][j]+b[q][j]>max)

max=b[i][j]+b[q][j];

v[k]=max;

max=-10000;

k++;}

q++;

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

b[q][k]=v[k];

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

if(b[q-1][j]==b[q][j])

r++;

}

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

if(b[q][j]<-500) exist[j]=1;

if(z!=h&&exist[z]!=1)

{

printf("\n");

printf("Drumurile de valoare maxima dmax(x%d,x%d) sunt:\n",z,h);

printf("\n");

max=-9999;

t=1;

d[1][t]=z;

t++;

i\_h=z;

while(i\_h!=h)

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

if(b[i\_h][j]+b[q][j]>max&&i\_h!=j)

{max=b[i\_h][j]+b[q][j];

d[1][t]=j;}

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

if(b[i\_h][j]+b[q][j]==max&&i\_h!=j&&d[1][t]!=j)

d[2][t]=j;

max=-9999;

i\_h=d[1][t];

t++;

}

for(i=1;i<t;i++)

{c[i]=d[1][i];

if(d[2][i]>0)

{g2=i;

l=d[2][i];}

p++;

}

if(g2!=0)

{

for(i=1;i<g2;i++)

d[2][i]=d[1][i];

max=-9999;

t=g2+1;

int i\_h2;

i\_h2=l;

while(i\_h2!=h)

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

if(b[i\_h2][j]+b[q][j]>max&&i\_h2!=j)

{max=b[i\_h2][j]+b[q][j];

d[2][t]=j;}

max=10000;

i\_h2=d[2][t];

t++;}

if(g2!=0)

printf("\tdmax(x%d,x%d)= %d =( ",z,h,b[q][z]);

for(j=1;j<t;j++)

if(d[2][j]>0)

printf("x%d ",d[2][j]);

if(d[2][1]>0)

printf(")\n",z,h);

printf("\n");

}

printf("\tdmax(x%d,x%d)= %d =( ",z,h,b[q][z]);

for(i=1;i<p;i++)

printf("x%d ",c[i]);

printf(")\n");

printf("\n");

for(i=1;i<15;i++)

{

d[1][i]=0;

d[2][i]=0;

}

for(i=1;i<=15;i++)

c[i]=0;

p=1;

g2=0;

l=0;

printf("\n");

}else{

printf("Drumuri din x%d in x%d NU EXISTA\n",z,h);

for(i=1;i<15;i++)

{

d[1][i]=0;

d[2][i]=0;

}

for(i=1;i<=15;i++)

c[i]=0;

p=1;

g2=0;

l=0;

printf("\n");

}

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

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

a[i][j]=0;

q=0;

r=0;

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

exist[j]=0;

}

getch();

}

int main()

{int a[40][40],b[40][40],c[40],v[40],d[40][40],aux[40],la[40][40],t,num,t\_i,z,i,j,s=0,n,m,mi=1,min,h,h\_j,h\_i,max,p=1,k,q,g=0,l,x\_i,x\_j,r;

int\*\* pd;

system("Color 21");

pd=(int\*\*)malloc(40\*sizeof(int\*));

for (i=0;i<40;i++){

pd[i]=(int\*)malloc(40\*sizeof(int));

}

printf(" \t\t\t\tDati numarul de virfuri: ");

scanf("%d",&n);

printf("\n");

for(x\_i=1;x\_i<=n;x\_i++)

for(x\_j=1;x\_j<=n;x\_j++)

la[x\_i][x\_j]=-9;

for(x\_i=1;x\_i<=n+1;x\_i++)

for(x\_j=1;x\_j<=n;x\_j++)

{

m99:system("cls");

printf("\n");

printf(" \t\t\t\tLista de adiacenta: \n");

printf("Х");

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

printf("НННН");

printf("ё");

printf("\n");

printf("і x ");

printf("|F(x)");

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

printf(" ");

printf("|\n");

printf("ННННН");

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

printf("НННН");

printf("\n");

main(){

int i, j;

golire(t\_lista);

golire(t\_pondere);

golire\_vector(v\_drum);

item=menu();

numarul\_virfuri:

system("cls");

printf("\n Care este numarul virfurilor?\n");

scanf("%d",&n\_virfuri);

if(n\_virfuri<=0 || n\_virfuri>45){

printf("\n Ati introdus un numar gresit, mai introduceti odata!\n");

fflush(stdin);

Sleep(4000);

goto numarul\_virfuri;

}

citire\_lista\_adiacenta(t\_lista,n\_virfuri);

system("cls");

citire\_pondere\_lista(t\_lista,t\_pondere,n\_virfuri);

system("cls");

switch(item){

case(1):

completare\_liste\_min(t\_pondere,n\_virfuri);

break;

case(2):

completare\_liste\_max(t\_pondere,n\_virfuri);

break;

case(3):

transpunere\_matrice(t\_pondere,n\_virfuri);

completare\_liste\_min(t\_pondere,n\_virfuri);

break;

case(4):

transpunere\_matrice(t\_pondere,n\_virfuri);

completare\_liste\_max(t\_pondere,n\_virfuri);

break;

}

if(item==1)

printf(" Ponderea drumurilor minime de la toate virfurile la virful final X%d este:\n\n",v\_final);

if(item==2)

printf(" Ponderea drumurilor maxime de la toate virfurile la virful final X%d este:\n\n",v\_final);

if(item==3)

printf("\n Ponderea drumurilor minime de la virful initial X%d la toate virfurile este:\n\n",v\_final);

if(item==4)

printf("\n Ponderea drumurilor maxime de la virful initial X%d la toate virfurile este:\n\n",v\_final);

afisare\_rezultat(t\_pondere,n\_virfuri);

printf("\n\n Pentru a continua apasati un buton...\n");

getch();

virf\_initial:

system("cls");

if(item==1 || item==2)

printf("\n Care este virful initial?\n");

if(item==3 || item==4)

printf("\n Care este virful final?\n");

scanf("%d",&v\_initial);

if(v\_initial==v\_final || v\_initial<=0 || v\_initial>n\_virfuri){

printf("\n Ati introdus un virf gresit, mai introduceti odata!\n");

fflush(stdin);

Sleep(4000);

goto virf\_initial;

}

if(item==1 || item==3){

for(i=1; i<=n\_virfuri; i++)

t\_pondere[i][i]=3\*pon\_max;

completare\_drum\_minim(t\_pondere,v\_drum,n\_virfuri);

if(item==3){

v\_intermediar=v\_initial;

v\_initial=v\_final;

v\_final=v\_intermediar;

}

printf("\n Drumul minim din virful X%d in virful X%d este:\n",v\_initial,v\_final);

}

if(item==2 || item==4){

for(i=1; i<=n\_virfuri; i++)

t\_pondere[i][i]=(-3)\*pon\_max;

completare\_drum\_maxim(t\_pondere,v\_drum,n\_virfuri);

if(item==4){

v\_intermediar=v\_initial;

v\_initial=v\_final;

v\_final=v\_intermediar;

}

printf("\n Drumul maxim din virful X%d in virful X%d este:\n",v\_initial,v\_final);

}

if(item==3 || item==4)

inversare\_vector(v\_drum,n\_virfuri);

afisare\_drum(v\_drum,n\_virfuri);

printf("\n");

getch();

}

**Rezultatele**

Figura 1 – Algoritmul Belman-Kalaba

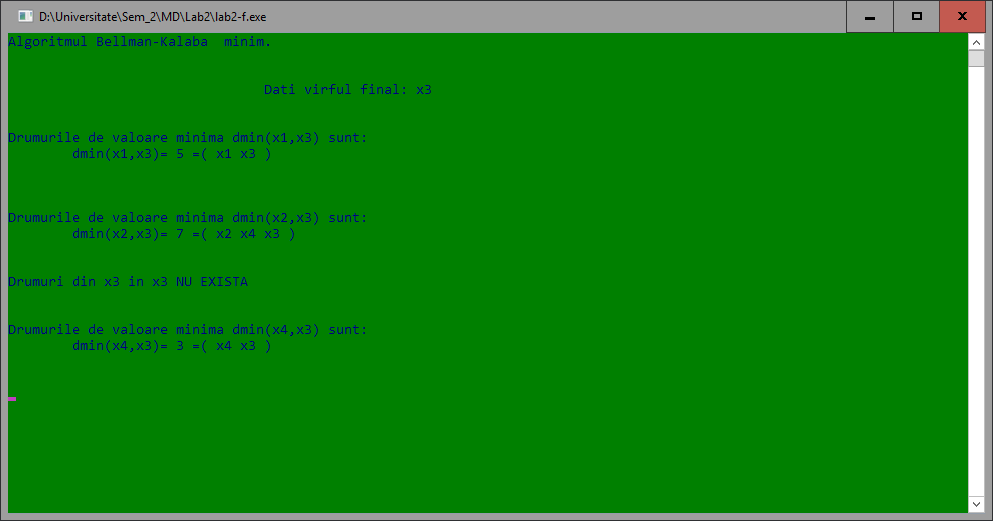
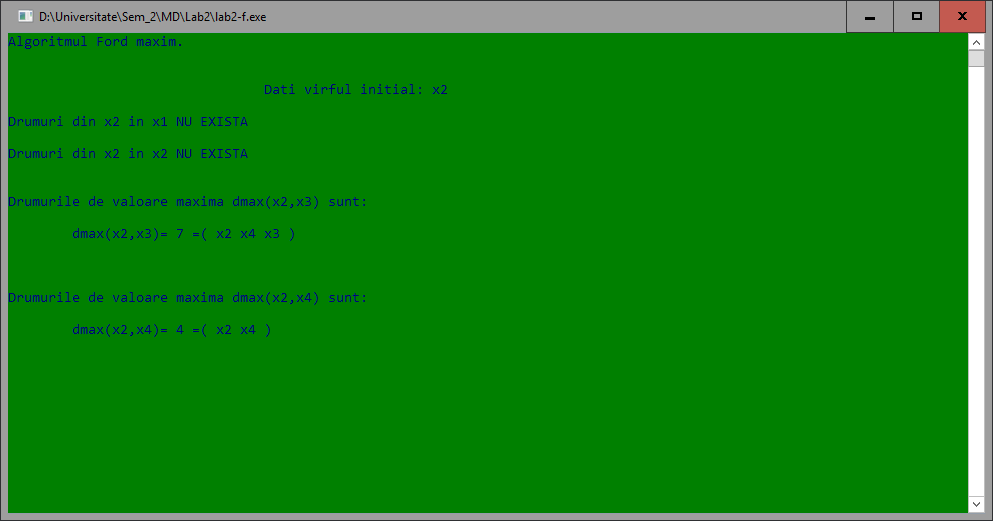


Figura 2 – Algoritmul Ford



**Concluzie**

Efectuând aceasta lucrare am făcut cunoştinţă: cu noţiunea de graf şi valoare a drumului, algoritmii Bellman Kalaba şi Ford de determinare a drumului de valoare minima întru-n graf având dat un vârf iniţial sau unul final respectiv şi am implementat aceşti algoritmi în limbajul de programare C.