Ministerul Educaţiei al Republicii Moldova

Universitatea Tehnică a Moldovei



**RAPORT**

*La Matematica Discretă*

*Lucrarea de laborator nr. 4-5*

*A efectua*t Popriţac Ion

Gr. C-131

*A verificat* Marusic G.

2014

# LUCRAREA DE LABORATOR Nr. 4

# TEMA: ALGORITMUL DETERMINĂRII GRAFULUI DE ACOPERIRE

1. **SCOPUL LUCRĂRII:**

### Studierea algoritmului de determinare a grafului de acoperire şi elaborarea programelor care vor realiza acest algoritm.

1. **NOTE DE CURS**

## Noţiune de graf de acoperire

Fie *H* un subgraf care conţine toate vârfurile unui graf arbitrar *G*. Dacă pentru fiecare componentă de conexitate a lui *G* subgraful *H* va defini un arbore atunci *H* se va numi graf de acoperire (scheletul sau carcasă) grafului G. Este evident că graful de acoperire există pentru oricare graf: eliminând ciclurile din fiecare componentă de conexitate, adică eliminând muchiile care sunt în plus, vom ajunge la graful de acoperire.

Se numeşte graf aciclic orice graf care nu conţine cicluri. Pentru un graf arbitrar *G* cu *n* vârfuri şi *m* muchii sunt echivalente următoarele afirmaţii:

1. *G* este arbore;
2. *G* este un graf conex şi *m* = *n - 1*;
3. *G* este un graf aciclic şi *m = n - 1*;
4. oricare două vârfuri distincte (diferite) ale lui *G* sunt unite printr-un lanţ simplu care este unic;
5. *G* este un graf aciclic cu proprietatea că, dacă o pereche oarecare de vârfuri neadiacente vor fi unite cu o muchie, atunci graful obţinut va conţine exact un ciclu.

**Consecinţă**: numărul de muchii pentru un graf arbitrar G, care va fi necesar a fi eliminate spre a obţine un graf de acoperire nu depinde de ordinea eliminării lor şi este egal cu

*m(G)-n(G)+k(G)*,

unde *m(G), n(G)* şi *k(G)* sunt numărul de muchii, vârfuri şi componente conexe, respectiv.

Numărul *s(G) = m(G)-n(G)+ k(G)* se numeşte *rang* *ciclic* sau număr *ciclomatic* al grafului *G*. Numărul *r(G) = n(G)-k(G)* – *rang cociclomatic* sau *număr cociclomatic.*

Deci, *s(G)+r(G)=m(G)*.

Este adevărată următoarea afirmaţie: orice subgraf a unui graf arbitrar *G* se conţine într-un graf de acoperire a grafului *G*.

## Algoritmul de determinare a grafului de acoperire

Există mai mulţi algoritmi de determinare a grafului de acoperire. Algoritmul de mai jos nu este un algoritm-standard, ci este unul elaborat în bază algoritmului de căutare în lărgime. Esenţa algoritmului constă în aceea că folosind două fire de aşteptare în unul din care sunt înscrise (pe rând) numerele vârfurilor adiacente cu vârfurile din celălalt FA (ca şi în cazul căutării în lărgime), vor fi eliminate muchiile dintre vârfurile unui FA şi toate muchiile în afară de una dintre fiecare vârf al FA curent şi vârfurile din FA precedent. în cazul În care ambele FA vor deveni vide procedura se va termina.

Pentru a nu admite ciclarea şi ca să fim siguri că au fost prelucrate toate componentele conexe se va utiliza marcarea vârfurilor. Dacă după terminarea unui ciclu ordinar nu au mai rămas vârfuri nemarcate procedura ia sfârşit, în caz contrar în calitate de vârf iniţial se va lua oricare din vârfurile nemarcate.

**Descrierea algoritmului:**

1. Se vor declara două FA (FA1 şi FA2) vide.
2. Se va lua în calitate de vârf iniţial un vârf arbitrar al grafului.
3. Se va introduce vârful iniţial în firul de aşteptare vid FA1 şi se va marca acest vârf.
4. Se vor introduce în FA2 toate vârfurile adiacente cu vârfurile din FA1 şi se vor marca. Dacă FA2 este vid se va trece la p.7, în caz contrar - la p. 4.
5. Se vor elimina toate muchiile care leagă vârfurile din FA2.
6. Pentru toate vârfurile din FA2 vor fi eliminate toate muchiile în afară de una care leagă vârful dat cu vârfurile din FA1.
7. Se vor schimba cu numele FA1 şi FA2 (FA1 va deveni FA2 şi invers).
8. Dacă există cel puţin un vârf nemarcat se va lua în calitate de vârf iniţial oricare din acestea şi se va trece la p.1, altfel
9. STOP.

Graful obţinut este graful de acoperire.

1. **SARCINA DE BAZĂ**
2. Elaboraţi organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.
3. Utilizând procedurile de introducere a grafului în memoria CE din lucrarea Nr. 1, elaboraţi un program cu următoarele facilităţi:

* introducerea grafului care este dat sub formă de matrice de incidenţă, adiacenţă sau listă de adiacenţă;
* determinarea grafului de acoperire, pornind de la un vârf arbitrar;
* extragerea informaţiei la display sau imprimantă în oricare din formele numite.

4. ÎNTREBĂRI DE CONTROL

1. Ce este un graf aciclic şi prin ce se deosebeşte el de un arbore?
2. Definiţi noţiunile de arbore şi graf de acoperire.
3. Care vor fi transformările ce vor fi efectuate într-un graf arbitrar pentru a obţine graful de acoperire?
4. Care este esenţa algoritmului de determinare a grafului de acoperire?
5. Evidenţiaţi etapele de bază ale algoritmului de determinare a grafului de acoperire.

Listing-ul 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]);

}

}

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();

}

Concluzie: În urma efectuării lucrarii de laborator am studiat algoritmul de determinare a grafului de acoperire. Cu ajutorul acestui algoritm am elaborat programa care îl realizează. Am elaborat organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.