****

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Calancea Cătălin**

**MI-222**

**Raport**

**pentru lucrarea de laborator Nr.1**

***la cursul de “Matematica Discreta”***

Verificat:

**Orindas Victoria,** *asist. univ.*

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău – 2023**

**Scopul lucrării:** Studierea metodelor de definire a unui graf:matrice de incidenta, matrice de adiacenta,liste. Elaborarea unor procedure de introducere, extragere si transformare a diferitelor forme de reprezentare interna a grafurilor cu scoaterea rezultatelor la display si imprimanta.

**Sarcini:**

**1. Să elaborați procedura introducerii unui graf în memoria calculatorului în formă de matrice de incidenţă, matrice de adiacenţă şi listă de adiacenţă cu posibilităţi de analiză a corectitudinii.**

**2. Să elaborați proceduri de transformare dintr-o formă de reprezentare în alta.**

**3. Folosind procedurile menţionate să elaborați programul care va permite:**

**• introducerea grafului reprezentat sub oricare din cele trei forme cu posibilităţi de**

**corecţie a datelor;**

**• păstrarea grafului în memoria în formă de listă de adiacenţă;**

**• extragerea informaţiei într-una din cele trei forme la display**

**• transformarea grafului dintr-o representare în alta: ex din Matrice de adiacenta să transformați în Lista sau Matrice de Incidența, și viceversa.**

**4. La apărarea lucrării trebuie să aratați cum lucrează programul, să introduceți un graf în forma selectată de profesor, și să transformați în alte două forme. Să explicați cum se transfomă fiecare formă de pastrare a grafului în alta.**

**Codul programului:**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <locale.h>

//declararea si definirea unei structuri pentru //lista de adiacenta

typedef struct \_node

{

int value;

struct \_node \*next;

}

node;

void MIin(); //introduceti matricea de incidenta

void MSin(); //introduceti matricea de adiacenta

void SSin(); //introduceti lista de adiacenta

//functii de input

void readArray(int \*, int, int); //Matrice

node \*readList(); //Lista

//functii de output

void printArray(int \*, int, int); //Matrici

void printList(node \*); //Lista

//functii de transformare

//m. incidenta -> m. adiacenta

void MI\_MS(int \*, int \*, int, int);

//m. adiacența -> lista

void MS\_SS(int \*, int, node \*\*);

//lista -> m. incidenta

void SS\_MI(int \*, int , node \*\*);

//auxiliar

// numararea elementele listei de adiacenta

int countInList(node \*\*, int);

//eliberarea memoriei pentru un sir din lista

void destroy(node \*);

int main()

{

setlocale(0,"Eng");

int choice;

printf("Alegeti optiunea potrivitt:\n");

printf("1 - Convertire din matricea de incidenta\n");

printf("2 - Convertire din matricea de adiacenta\n");

printf("3 - Convertire din lista de adiacenta\n");

printf("0 - Inchide programul\n");

printf("Ati ales: "); scanf("%d", &choice);

switch(choice)

{

case 1: MIin(); break;

case 2: MSin(); break;

case 3: SSin(); break;

default: printf("\t Programul se inchide...\n"); break;

}

getch();

return 0;

}

//definirea functiilor...

void MIin()

{

int M, N, i, choice;

int \*A, \*B;

node \*\*arrayOfList;

printf("Specificati dimensiunile matricei:\n");

printf("M="); scanf("%d", &M);

printf("N="); scanf("%d", &N);

A=(int \*)malloc(M\*N\*sizeof(int));

readArray(A, M, N);

printf("-----------------------------------n");

printf("Matricea de incidenta:\n");

printArray(A, M, N);

printf("----------------------------------\n");

printf("Alegeti varianta de reprezentare:\n");

printf("1 - Convertiti in matricea de adiacenta\n");

printf("2 - Convertiti in lista de adiacenta\n");

scanf("%d", &choice);

switch(choice)

{

case 1: printf("Convertire in matrice de adiacenta\n");

B=(int \*)calloc(N\*N, sizeof(int)); //alocarea memoriei pentru MA

MI\_MS(A, B, M, N); //transformare

printf("\nMatricea de adiacenta primita:\n");

printf("----------------------------------\n");

printArray(B, N, N); //output matricei rezultate

printf("----------------------------------\n");

free(A);

free(B);

break;

case 2: B=(int \*)calloc(N\*N, sizeof(int)); //alocarea memoriei pentru MS intermediar

MI\_MS(A, B, M, N); //transformare

free(A);

printf("Convertire in lista de adiacenta\n");

arrayOfList=(node \*\*)malloc(N\*sizeof(node \*)); //alocarea memoriei pentru LA

MS\_SS(B, N, arrayOfList);

printf("\nLista de adiacenta primita:\n");

printf("----------------------------------\n");

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

{

printf("%d:", i+1);

printList(\*(arrayOfList+i));

}

printf("----------------------------------\n");

free(B);

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

break;

default:printf("\nAlegere gresita...\n"); free(A);

}

}

void MSin()

{

int M, N, i, choice;

int \*A;

node \*\*arrayOfList;

printf("Specificati numarul de varfuri:\n");

printf("N="); scanf("%d", &N);

A=(int \*)malloc(N\*N\*sizeof(int));

readArray(A, N, N);

printf("----------------------------------\n");

printf("Matricea adiacenta:\n");

printArray(A, N, N);

printf("----------------------------------\n");

printf("Alegeti varianta de reprezentare:\n");

printf("1 - Convertire in lista de adiacenta\n");

printf("2 - Convertire in matrice de incidenta\n");

scanf("%d", &choice);

switch(choice)

{

case 1:printf("ПConvertire in lista de adiacenta\n");

arrayOfList=(node \*\*)malloc(N\*sizeof(node \*));

MS\_SS(A, N, arrayOfList);

printf("\nLista de adiacenta primita:\n");

printf("----------------------------------\n");

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

{

printf("%d:", i+1);

printList(\*(arrayOfList+i));

}

free(A);

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

break;

case 2:arrayOfList=(node \*\*)malloc(N\*sizeof(node \*));

MS\_SS(A, N, arrayOfList);

M=countInList(arrayOfList, N);

free(A);

printf("Conversie in matricea de incidenta\n");

A=(int \*)calloc(M\*N, sizeof(int));

SS\_MI(A, N, arrayOfList);

printf("\nMatricea de incidenta primita::\n");

printf("----------------------------------------\n");

printArray(A, M, N);

printf("----------------------------------------\n");

free(A);

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

break;

default:printf("\nAlegere gresita...\n"); free(A);

}

}

void SSin()

{

int M, N, i, choice;

int \*A, \*B;

node \*\*arrayOfList;

printf("\n\t+ Numarul de varfuri in graf: "); scanf("%d", &N);

arrayOfList=(node \*\*)malloc(N\*sizeof(node \*)); // alocarea memoriei pentru LA

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

{

printf("Varf #%d\n", i+1);

\*(arrayOfList+i)=readList();

}

printf("----------------------------------\n");

printf("Lista adiacentei\n");

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

{

printf("%d:", i+1);

printList(\*(arrayOfList+i));

}

printf("----------------------------------\n");

M=countInList(arrayOfList, N);

printf("Alegeti in ce forma s afisati:\n");

printf("1 - Convertire in matrice de incidenta\n");

printf("2 - Convertire in matrice de adiacenta\n");

scanf("%d", &choice);

switch(choice)

{

case 1:printf("Conversie in matrice de incidenta\n");

A=(int \*)calloc(M\*N, sizeof(int));

SS\_MI(A, N, arrayOfList);

printf("\nMatricea de incidenta primita:");

printf("\n--------------------------------------\n");

printArray(A, M, N);

printf("----------------------------------------\n");

free(A);

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

break;

case 2: A=(int \*)calloc(M\*N, sizeof(int));

SS\_MI(A, N, arrayOfList);

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

B=(int \*)calloc(N\*N, sizeof(int));

printf("Conversie in matrice de adiacenta\n");

MI\_MS(A, B, M, N);

printf("\nMatricea de adiacenta primita:");

printf("\n--------------------------------------\n");

printArray(B, N, N);

printf("----------------------------------------\n");

free(A);

free(B);

break;

default:printf("\nAlegere gresita\n");

for(i=0; i<N; i++) destroy(\*(arrayOfList+i));

free(arrayOfList);

}

}

void readArray(int \*A, int M, int N)

{

int i, j;

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

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

{

printf("A[%d][%d]=", i+1, j+1);

scanf("%d",(A+i\*N+j));

}

}

node \*readList()

{

int i=0, semn;

int sizeOfNode=sizeof(node);

node \*A=NULL, \*R1, \*R2;

printf("%d : ", ++i); scanf("%d",&semn);

if(semn)

{

R1=(node \*)malloc(sizeOfNode);

R1->value=semn; R1->next=NULL;

A=R1;

printf("%d : ", ++i); scanf("%d",&semn);

}

while(semn)

{

R2=(node \*)malloc(sizeOfNode);

R2->value=semn; R2->next=NULL;

R1->next=R2; R1=R2;

printf("%d : ", ++i); scanf("%d",&semn);

}

printf("\n");

return A;

}

void printArray(int \*A, int M, int N)

{

int i, j;

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

{

for(j=0; j<N; j++) printf("%3d", \*(A+i\*N+j));

printf("\n");

}

}

void printList(node \*R)

{

int i=0, semn;

int sizeOfNode=sizeof(node);

while(R)

{

printf("%2d", R->value);

R=R->next;

}

printf("%2d\n", 0);

}

void MI\_MS(int \*A, int \*B, int M, int N)

{

int i, j, a, b;

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

{

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

if(\*(A+i\*N+j)==2) a=b=j;

else

{

if(\*(A+i\*N+j)==-1) a=j;

if(\*(A+i\*N+j)==1) b=j;

}

\*(B+a\*N+b)=1;

}

}

void MS\_SS(int \*A, int N, node \*\*arrayOfList)

{

int i, j, k;

int sizeOfNode=sizeof(node);

node \*R1, \*R2;

for(i=0; i<N; i++) //gasirea primul 1 din sir

{

j=0; k=0;

while(!k&&(j<N))

{

if(\*(A+i\*N+j)) k=j+1;

j++;

}

//k=0 - nu in linia 1, altfel k - numarul //primul 1 din rand (numarand de la 1)

if(k)

{

R1=(node \*)malloc(sizeOfNode);

R1->value=k; R1->next=NULL;

\*(arrayOfList+i)=R1;

//continuam sa explorezi linia

while(j<N)

{

if(\*(A+i\*N+j))

{

R2=(node \*)malloc(sizeOfNode);

R2->value=j+1; R2->next=NULL;

R1->next=R2; R1=R2;

}

j++;

}

}

else \*(arrayOfList+i)=NULL;

}

}

void SS\_MI(int \*A, int N, node \*\*arrayOfList)

{

int i, j, k=0;

node \*R;

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

{

R=\*(arrayOfList+i);

while(R)

{

if((R->value)-1 == i) \*(A+k\*N+i)=2;

else

{

\*(A+k\*N+i)=-1;

\*(A+k\*N+(R->value)-1)=1;

}

R=R->next;

k++;

}

}

}

int countInList(node \*\*arrayOfList, int N)

{

int i, k=0;

node \*R;

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

{

R=\*(arrayOfList+i);

while(R)

{

R=R->next;

k++;

}

}

return k;

}

void destroy(node \*R)

{

node \*TMP;

while(R)

{

TMP=R;

R=R->next;

free(TMP);

}

}

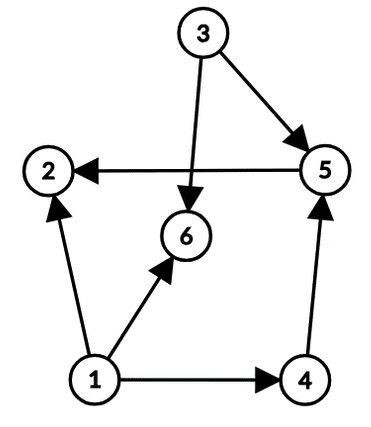


Figura 1. Graf antisimetric

**Lista de adiacenta:**

**1|2\_6\_4\_0**

**2|0**

**3|5\_6\_0**

**4|5\_0**

**5|2\_0**

**6|0**

Lista de adiacenta:

**0 1 0 1 0 1**

**0 0 0 0 0 0**

**0 0 0 0 1 1**

**0 0 0 0 1 0**

**0 1 0 0 0 0**

**0 0 0 0 0 0**

Lista de incidenta:

-1 1 0 0 0 0

-1 0 0 0 0 1

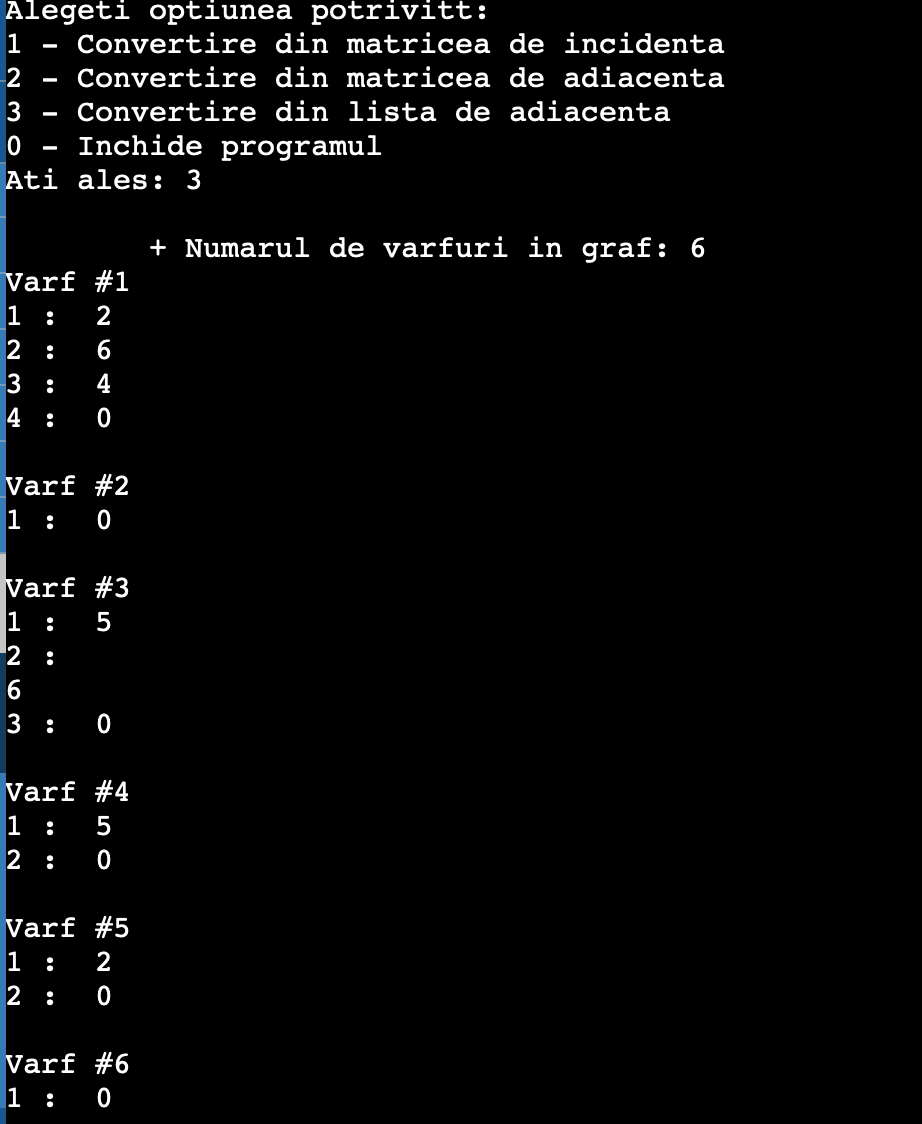
-1 0 0 1 0 0

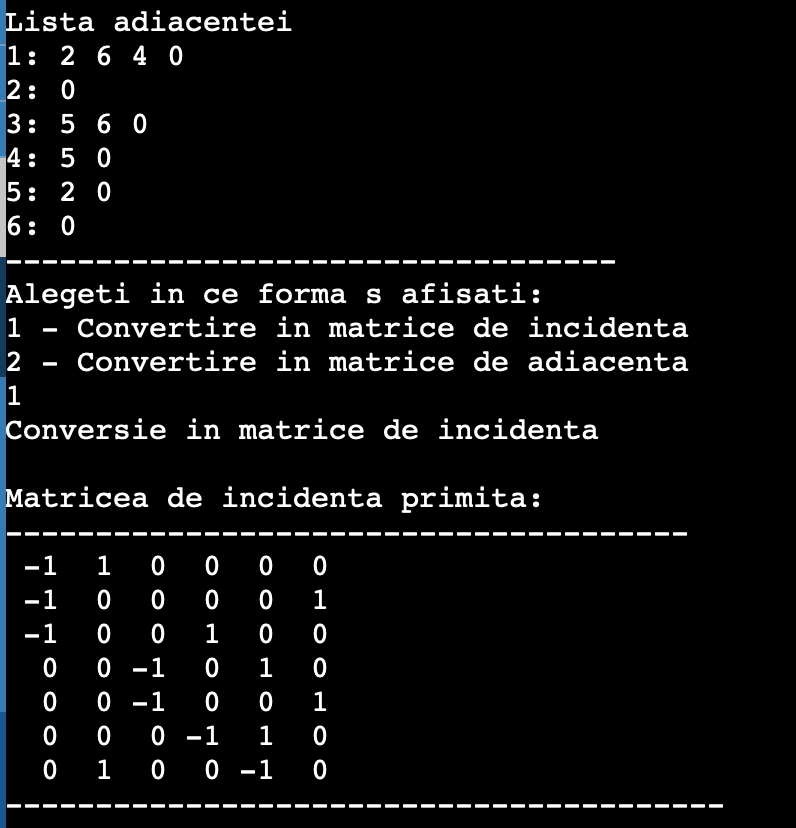
0 0 -1 0 1 0

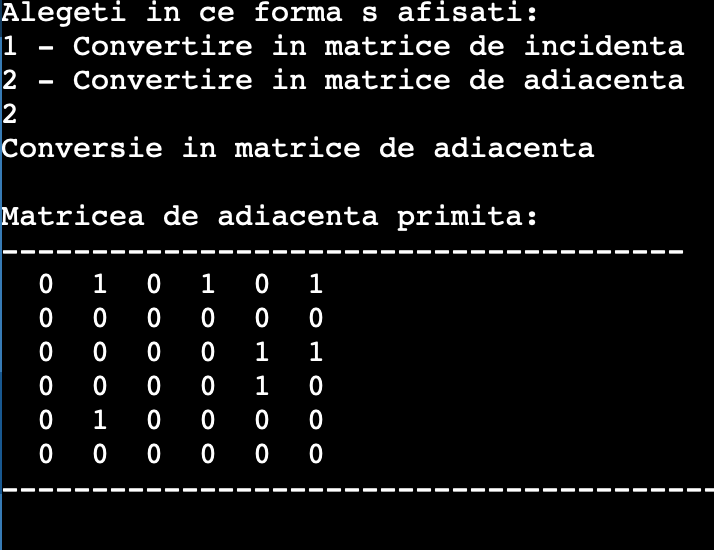
0 0 -1 0 0 1

0 0 0 -1 1 0

0 1 0 0 -1 0

****

****

****

**Concluzie:**În urma executarii lucrării de laborator nr.1 , am înțeles că există mai multe forme de reprezentare internă a unui graf, cum ar fi matricea de incidentă, matricea de adiacență și liste. Fiecare dintre aceste forme de reprezentare are avantaje și dezavantaje, iar alegerea uneia depinde de contextul în care este utilizată. Pentru a putea lucra cu grafuri, am elaborat proceduri pentru introducerea, extragerea și transformarea datelor în diferite forme de reprezentare. Aceste proceduri permit manipularea grafurilor într-un mod eficient și ușor de înțeles. În final, am concluzionat că reprezentarea grafurilor este o componentă esențială în problemele de grafuri și că este important să înțelegem și să folosim aceste metode în mod corespunzător pentru a putea aborda cu succes problemele ce implică grafuri.