****

**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.2**

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

Verificat:

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

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău – 2023**

**Scopul lucrării:** Studierea algoritmilor de cautare in graf si a diferitelor forme de pastrare si prelucrare a datelor. Elaborarea procedurii de cautare in adancime.

**Sarcini:**

1. Elaborati procedura cautarii in adancime intr-un graf arbitrar.
2. Elaborati un program cu urmatoarele posibilitati:

* Introducerea grafului in calculator.
* Parcurgerea grafului in adancime.
* Vizualizarea rezultatelor la display.

**Codul programului:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Declaram matricile pentru matricea de incidenta (MI) si matricea de adiacenta (MS), vectorul de culori (color),

// numarul de noduri (n), numarul de muchii (m) si modul in care se viziteaza nodurile (k).

int MI[100][100],

MS[100][100],

color[100],

n,

m,

k;

// Definim o structură abstractă "Data" și două structuri concrete care implementează această structură abstractă: "Stack" și "Queue".

// Aceste structuri abstracte vor fi folosite în funcția "DFS\_adj\_IVsit", pentru a implementa atât stiva cât și coada în același cod.

struct Data

{

int data[1000];

virtual void Push(int x) {}

virtual int Pop() {}

virtual int isEmpty() {}

};

struct Queue: Data

{

unsigned char q1, q2;

Queue() { q1 = q2 = 0; }

virtual void Push(int x){ data[q2++] = x; }

virtual int Pop() { return data[q1++]; }

virtual int isEmpty() { return q1 == q2; }

};

struct Stack: Data

{

unsigned char t;

Stack() { t=0; }

virtual void Push(int x){ data[t++] = x; }

virtual int Pop() { return data[--t]; }

virtual int isEmpty() { return !t; }

};

// Definim structura nodului unei liste de adiacenta.

struct node

{

int v;

node \*next;

node(int x, node \*t)

{

v = x;

next = t;

}

};

// Definim un pointer la un nod, numit "lnk", care va fi utilizat mai jos.

typedef node \*lnk;

lnk adj[100];

// Funcția "DFS\_adj\_IVsit" primește ca parametru un nod de start și vizitează toate nodurile atingibile de la acest nod, folosind fie o coadă, fie o stivă.

void DFS\_adj\_IVsit(int u)

{

Data\* toProceadj;

// Dacă variabila "k" este adevărată, vom folosi o stivă, altfel vom folosi o coadă.

if (k) toProceadj = new Stack;

else toProceadj = new Queue;

toProceadj->Push(u);

color[u] = 2;

// Cat timp stiva/ coada nu este goala, scoatem un nod din aceasta si il vizitam, apoi adaugam vecinii nevizitati in stiva/ coada.

do

{

u = toProceadj->Pop();

printf("%d ", u+1);

lnk tmp = adj[u];

while(tmp)

{

int v = tmp->v;

if (color[v] == 0)

{

color[v] = 2;

toProceadj->Push(v);

}

tmp = tmp->next;

}

} while (!toProceadj->isEmpty());

delete toProceadj;

}

// aplică DFS\_adj\_IVsit() pentru fiecare vârf nevizitat și afișează rezultatul.

void DFS\_adj\_Iterative()

{

printf("\nRaspunsul ");

for(int u=0; u<n; u++)

if(color[u]==0)

DFS\_adj\_IVsit(u);

printf("\n");

}

// citește matricea de incidență și o stochează în MI.

void FMI()

{

printf("Matricea de incidenta\n ");

for(int i=0; i<m; i++) {

printf("Introduce linia %d \n", i+1);

for(int j=0; j<n; j++)

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

}

}

// citește matricea de adiacență și o stochează în MS

void FMS()

{

printf("Matricea de adiacenta\n ");

for(int i=0; i<n; i++){

printf("Introduce linia %d\n",i+1);

for(int j=0; j<n; j++)

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

}

}

// lista de adiacență și o stochează sub formă de liste de adiacență în adj.

void FSS()

{

int tmp;

printf("Lista de adiacenta\n ");

for(int i=0; i<n; i++)

{

printf("Introduce arcul asociat cu varful %d\n", i+1);

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

scanf("%d", &tmp);

while(tmp)

{

adj[i] = new node(tmp-1, adj[i]);

scanf("%d", &tmp);

}

}

}

// convertește matricea de adiacență în lista de adiacență și o stochează în adj

void FMSvSS()

{

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

if(MS[i][j])

adj[i]=new node(j,adj[i]);

}

// convertește lista de adiacență în matricea de incidență și o stochează în MI.

void FSSvMI()

{

int m=0;

for(int i=0;i<n;i++)

{

lnk tmp=adj[i];

while(tmp)

{

MI[m][i]=-1;

MI[m++][tmp->v]=1;

tmp=tmp->next;

}

}

}

// convertește matricea de incidență în matricea de adiacență și o stochează în MS.

void FMIvMS()

{

int i,j,k,l;

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

{

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

{

if(MI[i][j]==-1) k=j;

if(MI[i][j]==1) l=j;

}

MS[k][l]=1;

}

}

// afișează matricea de incidență, matricea de adiacență și lista de adiacență.

void printm()

{

printf("Matricea de Incidenta\n");

for(int i=0;i<m;i++)

{

for(int j=0;j<n;j++)

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

printf("\n");

}

printf("Matricea de adiacenta\n");

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

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

printf("\n");

}

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

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

{

printf("%d : ",i);

lnk tmp=adj[i-1];

while(tmp)

{

printf("%d ",tmp->v+1);

tmp=tmp->next;

}

printf("0\n");

}

}

// citește datele de intrare (numărul de vârfuri și tipul reprezentării grafice) și apelează funcțiile corespunzătoare pentru a citi și a converti reprezentarea grafică. Apoi aplică DFS\_adj\_Iterative() și afișează rezultatul.

int main()

{

//clrscr();

printf("Introduceti numarul de varfuri si arce\n");

scanf("%d%d",&n,&m);

printf("Metoda de introducere\n1.Matrice de Incidenta\n2.Matricea de adiacenta\n3. Lista de adiacenta\n");

scanf("%d",&k);

switch(k)

{

case 1: FMI(); FMIvMS(); FMSvSS(); break;

case 2: FMS(); FMSvSS(); FSSvMI(); break;

case 3: FSS(); FSSvMI(); FMIvMS(); break;

}

printm();

printf("Alegeti: DFS - 1, BFS - 0\n");

scanf("%d",&k);

DFS\_adj\_Iterative();

return 0;

}

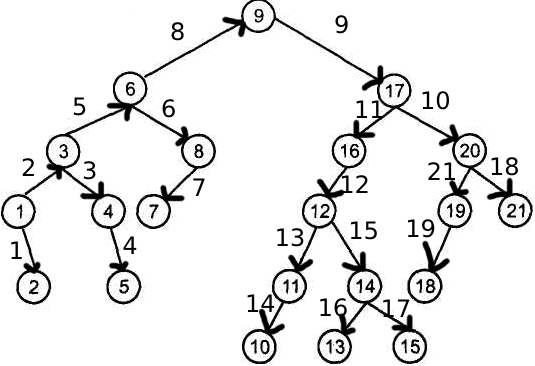


Figura 1. Graf

**Lista de adiacenta:**

**1|2\_3\_0**

**2|0**

**3|4\_6\_0**

**4|5\_0**

**5|0**

**6|8\_9\_0**

**7|0**

**8|7\_0**

**9|17\_0**

**10|0**

**11|10\_0**

**12|11\_0**

**13|0**

**14|13\_15\_0**

**15|0**

**16|12\_0**

**17|16\_20\_0**

**18|0**

**19|18\_0**

**20|19\_21\_0**

**21|0**

**Lista de adiacenta:**

0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**Lista de incidenta:**

-1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 -1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 -1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0

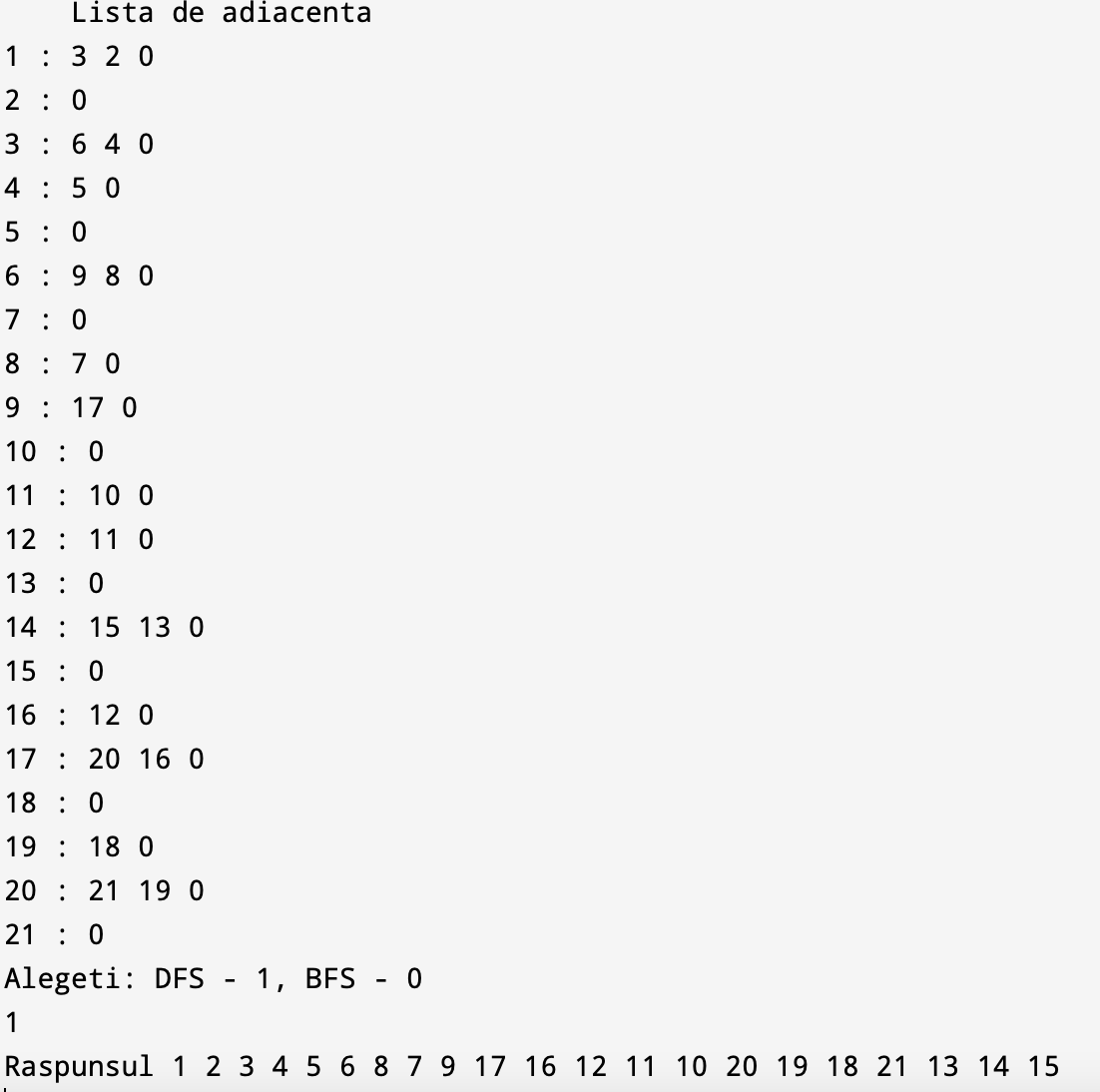
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

****

**Concluzie:**În urma executarii lucrării de laborator nr.2 , am înțeles că există mai multi algoritmi de căutare în graf și a diferitelor forme de păstrare și prelucrare a datelor, am înțeles că există mai multe tipuri de algoritmi de căutare, cum ar fi căutarea în lățime (BFS) și căutarea în adâncime (DFS). DFS (Depth-First Search) este un algoritm de căutare care explorează în profunzime toate nodurile unui graf. Algoritmul DFS poate fi implementat recursiv sau iterativ. Când algoritmul DFS explorează un nod, el vizitează primul nod nevizitat din lista sa de adiacență și continuă explorarea până când toate nodurile sunt vizitate.Pentru implementarea algoritmului DFS, se folosesc două structuri de date: o stivă și un vector de vizitare. Vectorul de vizitare este utilizat pentru a marca nodurile care au fost deja vizitate, iar stiva este utilizată pentru a păstra ordinea nodurilor care trebuie explorate.Prin urmare, procedura de căutare în adâncime (DFS) este o metodă eficientă de explorare a grafului și găsire a unui nod sau a unei căi între noduri. Cu toate acestea, trebuie să fim atenți să evităm buclele în graf atunci când implementăm algoritmul DFS.