MINISTERUL EDUCAŢIEI și CERCETĂRII al REPUBLICII MOLDOVA UNIVERSITATEA TEHNICĂ a MOLDOVEI   
FACULTATEA CALCULATOARE, INFORMATICǍ şi MICROELECTRONICǍ Departamentul Informatică și Ingineria Sistemelor

**Lucrare de laborator nr. 2**

La Matematica Discretă

Tema: ***Păstrarea grafurilor în memoria calculatorului.***

***Matricea drumurilor***

A realizat Chistol Maxim, st. gr. IA-233  
A verificat Vladimir Melnic

**Chișinău 2024**

**Cuprins**

[**1. Scopul lucrării** 3](#_Toc158886713)

[**2. Sarcina** 4](#_Toc158886714)

[**3. Concluzii** 13](#_Toc158886715)

# **1. Scopul lucrării**

1. Studierea algoritmilor de căutare în graf şi a diferitor forme de păstrare şi prelucrare a datelor.

2. Elaborarea procedurii de căutare în adâncime.

# **2. Sarcina**

1. Elaborați procedura căutării în adâncime într-un graf.

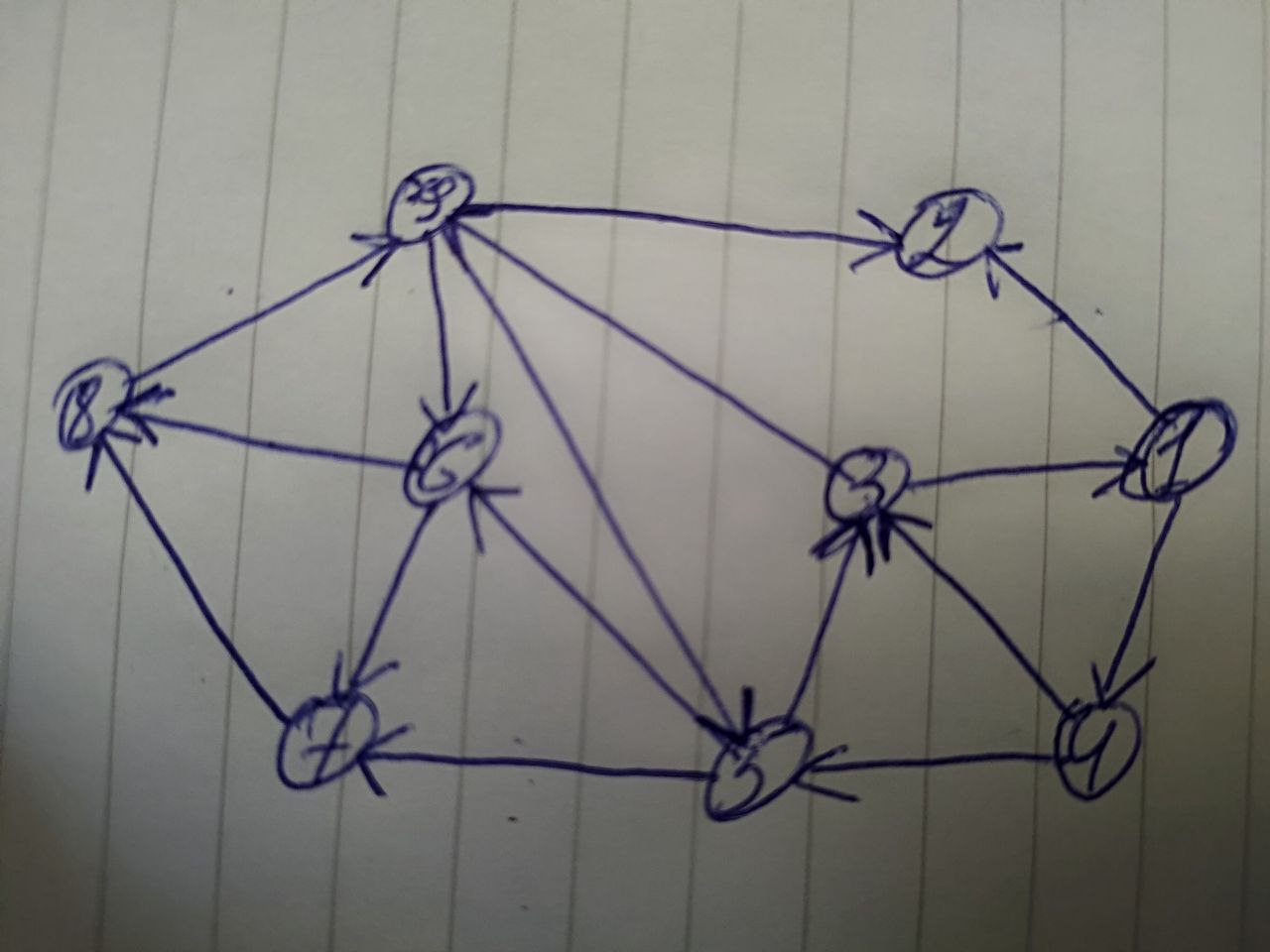
2. Elaboraţi un program cu următoarele posibilităţi:

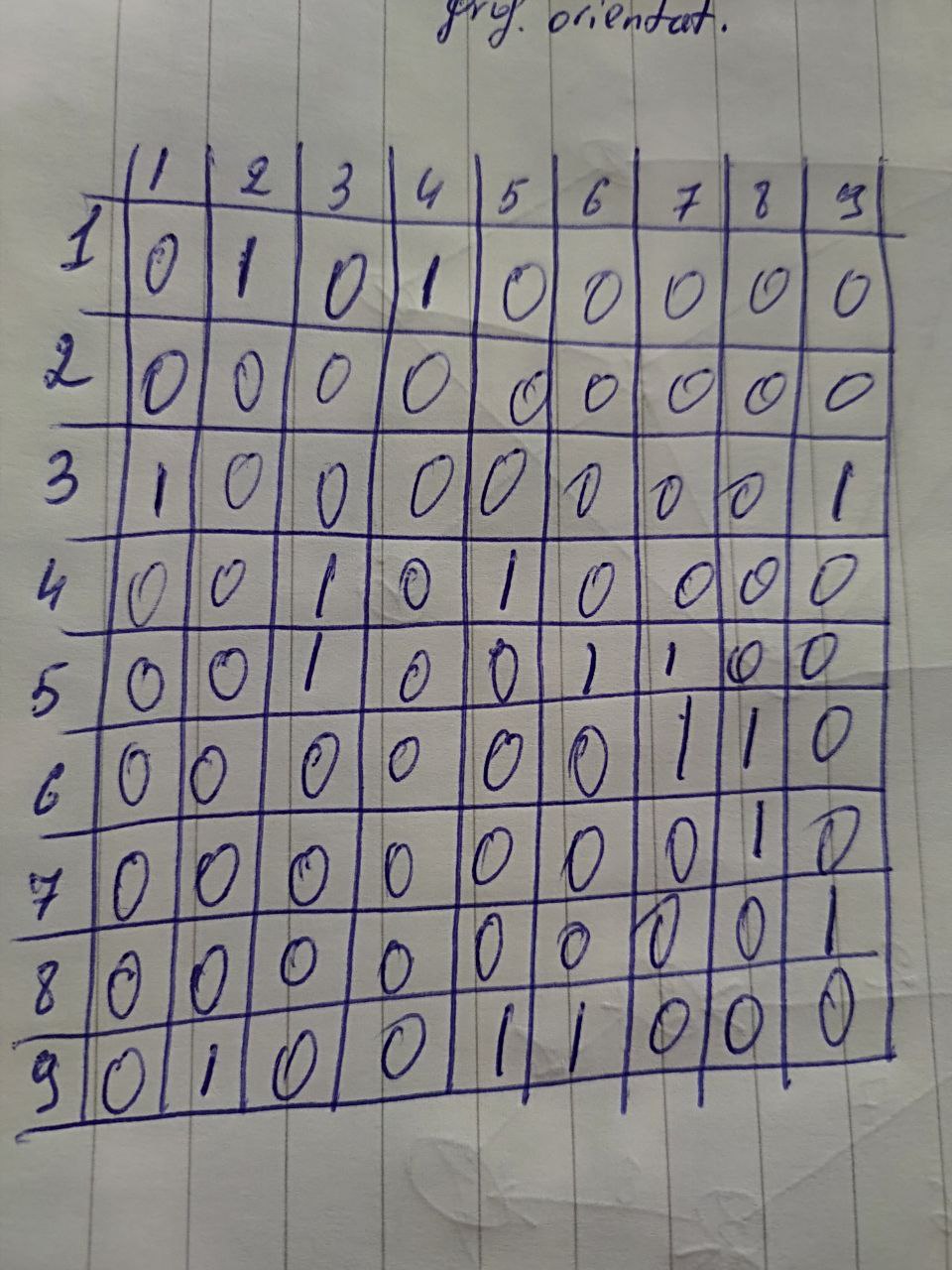
• introducerea grafului în calculator;

• parcurgerea grafului în adâncime;

• vizualizarea rezultatelor la display;

• matrice de adiacenţă (graf orientat).





#include <stdio.h>

#define MAX\_VERTICES 100

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

int visited[MAX\_VERTICES];

int numVertices;

// Functie pentru adaugarea unei muchii in graf

void addEdge(int start, int end) {

    adjacencyMatrix[start][end] = 1;

    adjacencyMatrix[end][start] = 1;

}

// Functie pentru parcurgerea in adancime a unui nod

void DFS(int vertex) {

    visited[vertex] = 1;

    printf("%d ", vertex);

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

        if (adjacencyMatrix[vertex][i] && !visited[i]) {

            DFS(i);

        }

    }

}

// Functie pentru afisare matrice

void displayMatrix() {

    printf("Matricea adiacentei:\n");

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

        for (int j = 0; j < numVertices; j++) {

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

        }

        printf("\n");

    }

}

// Functie pentru afisarea meniului

void displayMenu() {

    printf("\n1. Introducere graf");

    printf("\n2. Parcurgere graf in adancime");

    printf("\n3. Afisare matrice adiacenta");

    printf("\n0. Iesire");

    printf("\nAlegeti optiunea: ");

}

int main() {

    int option, start, end;

    do {

        displayMenu();

        scanf("%d", &option);

        switch (option) {

            case 1:

                // Introducere graf

                printf("\nIntroduceti numarul de noduri in graf: ");

                scanf("%d", &numVertices);

                // Inițializare matrice adiacentă

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

                    for (int j = 0; j < numVertices; j++) {

                        adjacencyMatrix[i][j] = 0;

                    }

                }

                // Adăugare muchii

                printf("\nIntroduceti muchiile (perechi de noduri separate prin spatii, -1 pentru a opri): ");

                while (1) {

                    scanf("%d %d", &start, &end);

                    if (start == -1 || end == -1) {

                        break;

                    }

                    addEdge(start, end);

                }

                break;

            case 2:

                // Parcurgere graf in adancime

                printf("Parcurgerea in adancime a grafului.. ");

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

                    visited[i] = 0;

                }

                int m;

                printf("\n\n Introduceti varful: ");

                scanf("%d",&m);

                        DFS(m);

                printf("\n");

                break;

            case 3:

                // Afisare matrice adiacenta

                displayMatrix();

                break;

            case 0:

                printf("\nProgramul se inchide...");

                break;

            default:

                printf("\nOptiune invalida! Va rugam sa alegeti din nou.");

        }

    } while (option != 0);

    return 0;

}

Explicatie Cod:

 // Parcurgere graf in adancime

                printf("Parcurgerea in adancime a grafului: ");

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

                    visited[i] = 0;

                }

                int m;

                printf("\n\n Introduceti varful: ");

                scanf("%d",&m);

                        DFS(m);

1. `for (int i = 0; i < numVertices; i++) { visited[i] = 0; }`: Acesta este for care initializează un vector de vizitare (`visited`) pentru toate nodurile grafului. Vectorul `visited` este folosit pentru a ține evidența nodurilor care au fost deja vizitate în timpul parcurgerii. Valorile inițiale sunt setate la 0 pentru a indica că niciun nod nu a fost vizitat încă.

2. `for (int i = 0; i < numVertices; i++) { if (!visited[i]) { DFS(i); } }`: Acesta este for care parcurge toate nodurile grafului. Dacă nodul curent (`i`) nu a fost vizitat (`!visited[i]`), atunci este apelată funcția `DFS(i)` pentru a începe o parcurgere în adâncime de la acel nod.

3. `DFS(i)`: Această funcție este presupusă a fi o implementare a parcurgerii în adâncime (DFS - Depth-First Search). Detaliile implementării efective a funcției nu sunt furnizate în fragmentul de cod dat, dar de obicei, o implementare DFS implică explorarea unui nod, marcarea lui ca vizitat și apoi recursiv explorarea tuturor vecinilor nevizitați ai nodului curent.

void DFS(int vertex) {

    visited[vertex] = 1;

    printf("%d ", vertex);

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

        if (adjacencyMatrix[vertex][i] && !visited[i]) {

            DFS(i);

        }

    }

}

for (int i = 0; i < numVertices; i++) {: Acest for parcurge toți vecinii nodului curent, unde numVertices reprezintă numărul total de noduri în graf.

if (adjacencyMatrix[vertex][i] && !visited[i]) { DFS(i); }: În interiorul If, se verifică dacă există o muchie între nodul curent (vertex) și nodul i, și dacă nodul i nu a fost vizitat încă (!visited[i]). Dacă aceste condiții sunt îndeplinite, se apelează recursiv funcția DFS pentru a explora în adâncime nodul i.

if (adjacencyMatrix[vertex][i] && !visited[i]): În interiorul if , verificăm dacă există o muchie între vârful vertex și vârful i și dacă vârful i nu a fost încă vizitat. adjacencyMatrix[vertex][i] este adevarata dacă există o muchie între vârfurile vertex și i, iar !visited[i] este adevarat dacă vârful i nu a fost încă vizitat.

Funcția DFS se oprește de la a explora o anumită ramură a grafului atunci când întâlnește un nod care a fost deja vizitat.

Acest lucru este verificat prin !visited[i].

Dacă nodul i a fost deja vizitat, funcția DFS nu se va apela pe ea însăși pentru acel nod, oprirea astfel recursivitatea pe acea cale.

# **3. Concluzii**

Scopul lucrării a fost de a studia algoritmii de căutare în graf, concentrându-se pe căutarea în adâncime (DFS). Sarcina principală a fost elaborarea unei proceduri de căutare în adâncime și implementarea unui program care să permită introducerea unui graf, parcurgerea acestuia în adâncime și vizualizarea rezultatelor.

Prin realizarea acestei sarcini, am dobândit o înțelegere mai profundă a modului în care algoritmii de căutare în graf funcționează și am aplicat aceste cunoștințe în practică prin dezvoltarea unei implementări în limbajul de programare C. Programul oferă utilizatorului flexibilitatea de a introduce un graf, de a-l parcurge în adâncime și de a vizualiza rezultatele, facilitând astfel explorarea și înțelegerea structurilor de graf.