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. 3**

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 algoritmului de căutare în lărgime;

2. Elaborarea programului de căutare în lărgime.

# **2. Sarcina**

1. Elaboraţi procedura care va realiza algoritmul de parcurgere a grafului în lărgime.

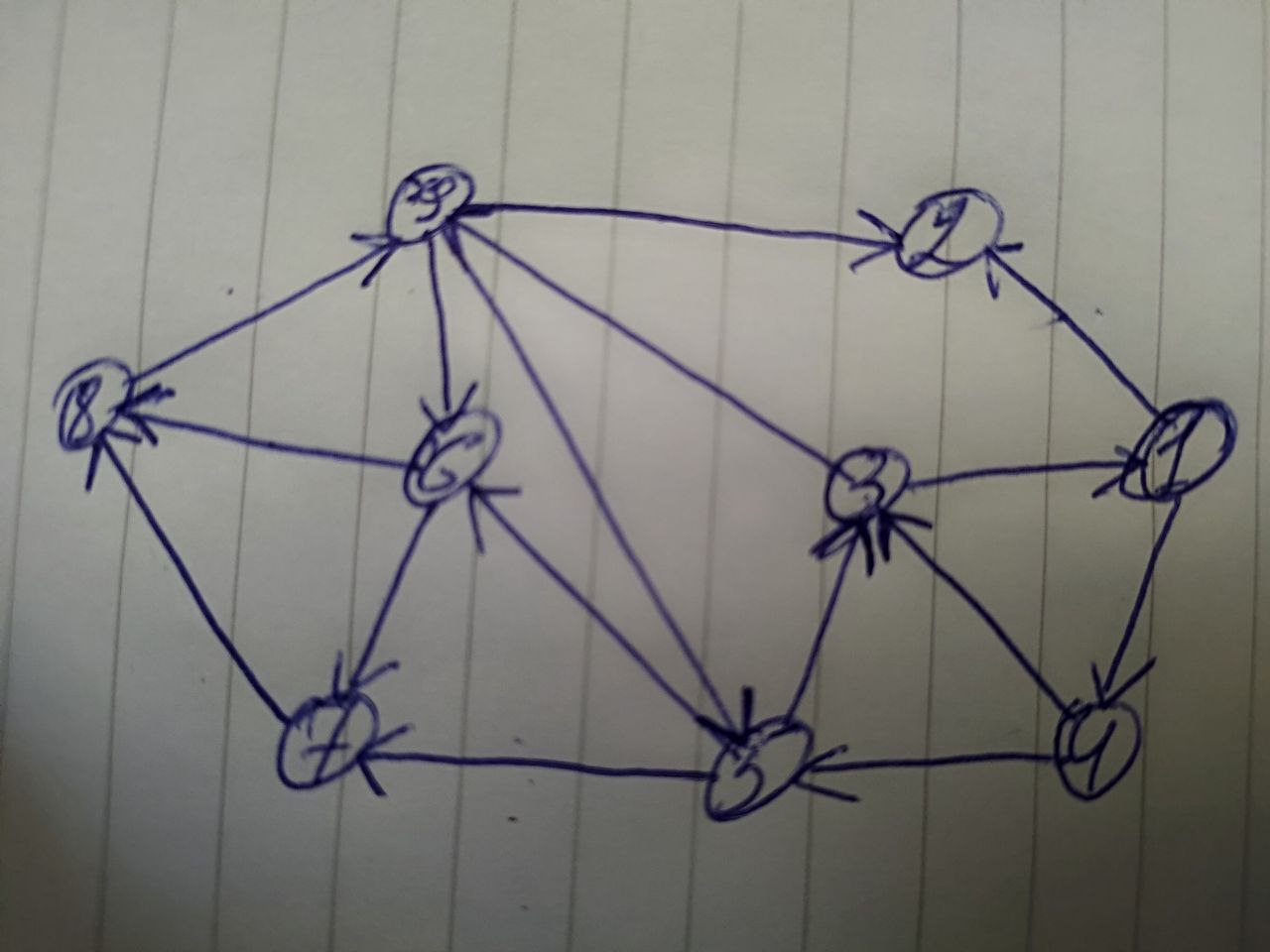
2. Folosind procedurile din lucrările precedente, elaboraţi programul care va permite:

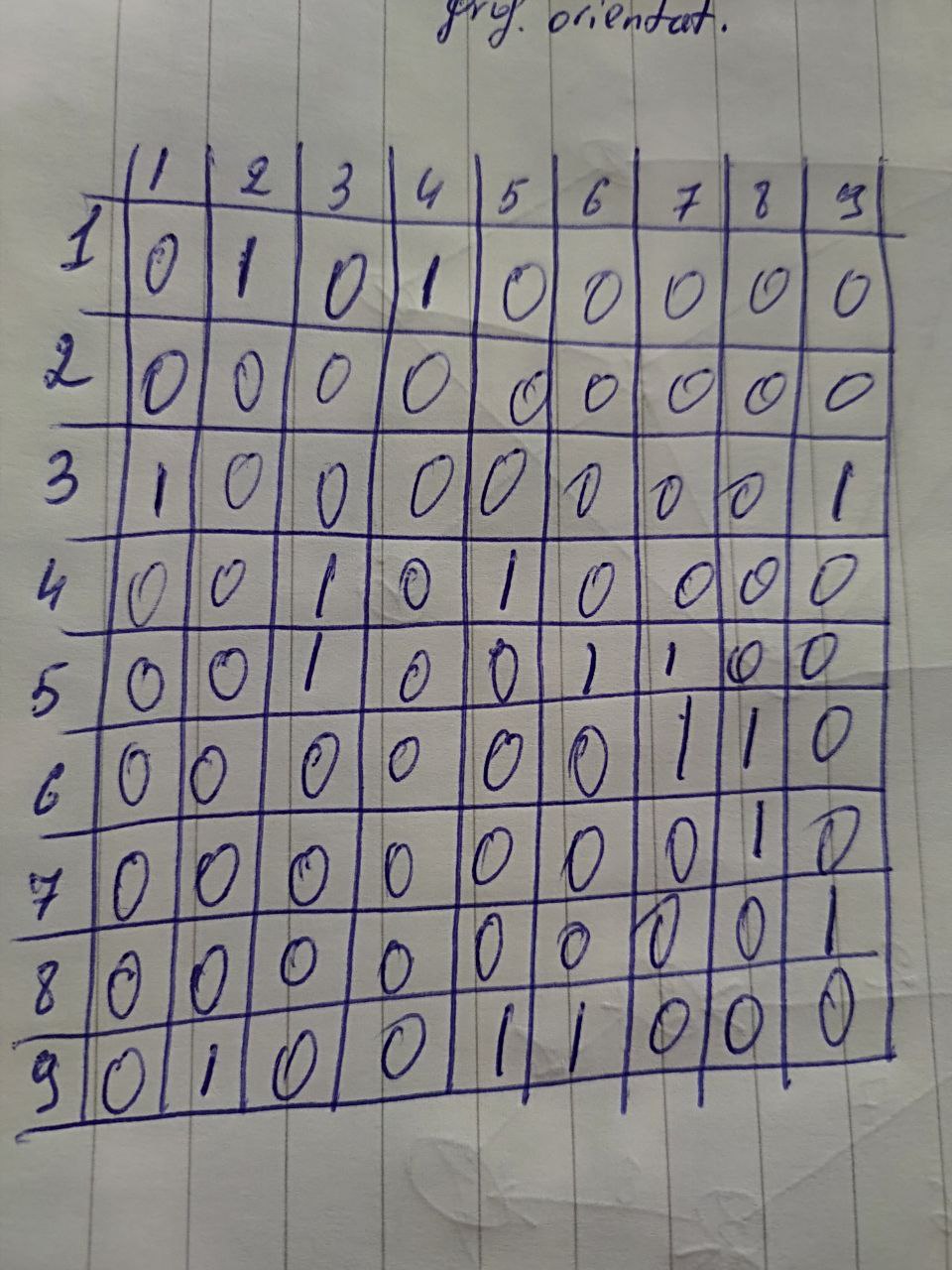
➢ introducerea grafului în memoria calculator;

➢ parcurgerea grafului în lărgime;

➢ vizualizarea rezultatelor la display;

• matrice de adiacenţă (graf orientat).





**Codul:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

// Structura pentru un nod în listă de adiacență

struct Node {

    int vertex;

    struct Node\* next;

};

// Structura pentru o listă de adiacență

struct AdjList {

    struct Node\* head;

};

// Structura pentru graf

struct Graph {

    int numVertices;

    struct AdjList\* array;

};

// Prototipuri funcții

struct Node\* createNode(int vertex);

struct Graph\* createGraph(int numVertices);

void addEdge(struct Graph\* graph, int start, int end);

void breadthFirstSearch(struct Graph\* graph, int startVertex);

void displayMenu();

int main() {

    int option, start, end;

    struct Graph\* graph = NULL;

    do {

        displayMenu();

        scanf("%d", &option);

        switch (option) {

            case 1:

                // Introducere grafului în memorie

                printf("Introduceti numarul de noduri in graf: ");

                scanf("%d", &end);

                graph = createGraph(end);

                printf("Introduceti 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(graph, start, end);

                }

                break;

            case 2:

                // Parcurgere graf în lățime

                if (graph == NULL) {

                    printf("Graful nu a fost introdus inca. Va rugam sa introduceti graful.\n");

                } else {

                    printf("Introduceti nodul de start pentru parcurgerea in latime: ");

                    scanf("%d", &start);

                    breadthFirstSearch(graph, start);

                }

                break;

            case 3:

    // Vizualizare rezultate

    if (graph == NULL) {

        printf("Graful nu a fost introdus inca. Va rugam sa introduceti graful.\n");

    } else {

        printf("Parcurgerea in latime a grafului este: \n");

        for (int i = 1; i <= graph->numVertices; ++i) {

            printf("Nodul %d: ", i);

            struct Node\* temp = graph->array[i].head;

            while (temp != NULL) {

                printf("%d ", temp->vertex);

                temp = temp->next;

            }

            printf("\n");

        }

    }

    break;

            case 0:

                printf("Programul se inchide.\n");

                break;

            default:

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

        }

    } while (option != 0);

    // Eliberare memorie alocată pentru graf

    if (graph != NULL) {

        for (int i = 1; i <= graph->numVertices; ++i) {

            struct Node\* current = graph->array[i].head;

            while (current != NULL) {

                struct Node\* temp = current;

                current = current->next;

                free(temp);

            }

        }

        free(graph->array);

        free(graph);

    }

    return 0;

}

// Creare nod în listă de adiacență

struct Node\* createNode(int vertex) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->vertex = vertex;

    newNode->next = NULL;

    return newNode;

}

// Creare graf cu un număr specific de noduri

struct Graph\* createGraph(int numVertices) {

    struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

    graph->numVertices = numVertices;

    graph->array = (struct AdjList\*)malloc(numVertices \* sizeof(struct AdjList));

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

        graph->array[i].head = NULL;

    }

    return graph;

}

// Adăugare muchie în graf

void addEdge(struct Graph\* graph, int start, int end) {

    struct Node\* newNode = createNode(end);

    newNode->next = graph->array[start].head;

    graph->array[start].head = newNode;

    // Dacă graful este neorientat, trebuie să adăugăm și muchia în sens invers

    newNode = createNode(start);

    newNode->next = graph->array[end].head;

    graph->array[end].head = newNode;

}

// Parcurgere în lățime a grafului

void breadthFirstSearch(struct Graph\* graph, int startVertex) {

    int\* visited = (int\*)malloc(graph->numVertices \* sizeof(int));

    for (int i = 1; i <= graph->numVertices; ++i) {

        visited[i] = 0;

    }

    // Coada pentru BFS

    int\* queue = (int\*)malloc(graph->numVertices \* sizeof(int));

    int front = -1, rear = -1;

    // Adăugăm nodul de start în coadă și îl marcam ca vizitat

    queue[++rear] = startVertex;

    visited[startVertex] = 1;

    printf("Rezultatele parcurgerii in latime: ");

    while (front != rear) {

        int currentVertex = queue[++front];

        printf("%d ", currentVertex);

        // Parcurgem vecinii nodului curent

        struct Node\* temp = graph->array[currentVertex].head;

        while (temp != NULL) {

            int adjVertex = temp->vertex;

            if (!visited[adjVertex]) {

                queue[++rear] = adjVertex;

                visited[adjVertex] = 1;

            }

            temp = temp->next;

        }

    }

    printf("\n");

    // Eliberare memorie

    free(visited);

    free(queue);

}

// Afișarea meniului

void displayMenu() {

    printf("\n--------Lista de optiuni--------\n"

           "1. Introducere grafului in memoria calculator\n"

           "2. Parcurgere grafului in latime\n"

           "3. Vizualizare rezultatelor la display\n"

           "0. Iesire din program\n"

           "Alegeti optiunea: ");

}

Inițializare: Se pregătesc structurile de date necesare pentru algoritm:  
  
Se alocă un vector numit visited pentru a marca nodurile vizitate.  
Se alocă o coadă pentru a stoca nodurile care urmează să fie procesate.  
Se adaugă nodul de start în coadă și se marchează ca vizitat.

Parcurgerea în lățime (BFS): Se desfășoară următoarele operații repetate până când coada devine goală:  
  
Se extrage un nod din fața cozii și se afișează.  
Se parcurg toți vecinii nevizitați ai nodului extras și aceștia sunt adăugați în coadă pentru procesare ulterioară.  
Nodurile sunt marcate ca vizitate pentru a evita prelucrarea repetată.

Coada este utilizată pentru a stoca nodurile care urmează să fie vizitate.  
Vectorul de vizitare este folosit pentru a marca nodurile care au fost deja vizitate.  
Complexitatea algoritmului:

Finalizare: După ce toate nodurile accesibile au fost vizitate și coada devine goală, algoritmul se încheie.  
Această procedură de parcurgere în lățime oferă o modalitate eficientă de a explora un graf, începând de la un nod de start și vizitând toți vecinii acestuia, apoi vecinii vecinilor, și așa mai departe, într-un mod sistematic.

# **3. Concluzii**

În concluzie, implementarea algoritmului de parcurgere în lărgime a unui graf într-un program reprezintă o etapă esențială în dezvoltarea sistemelor ce implică analiza și manipularea datelor grafice. Această implementare permite explorarea eficientă a conexiunilor dintre noduri și găsirea unor soluții optime într-o varietate de domenii, inclusiv rețele informatice, analiza rețelelor sociale și optimizarea rutelor în logistică. Prin intermediul acestui program, utilizatorii pot interacționa cu grafurile într-un mod intuitiv și pot extrage informații semnificative pentru luarea deciziilor sau rezolvarea problemelor complexe. Astfel, algoritmul de parcurgere în lărgime și aplicația rezultată aduc valoare adăugată în gestionarea și analiza datelor grafice.