MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII

Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică

Tehnologia Informației

Raport

Disciplina: Matematica discretă

Lucrarea de laborator nr. 1

Tema: *"*Păstrarea gafului în memoria calculatorului*"*

Student: _____ **Raevschi Grigore TI-231**

Coordonator: _____ **Ciobanu Ecaterina, asist. univ.**

Chişinău 2024

Contents

Condiția problemei – Introducere

Lucrarea de laborator №1 Păstrarea grafului în memoria calculatorului.

Sarcina de bază:

1.De elaborate 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.Elaboraţi procedura de transformare dintr-o formă de reprezentare în alta.

3. Folosind procedurile mentionate, elaboraţi porogramul care va permite:

a)introducerea grafului repreyentat sub oricare din cele trei forme cu posibilităţi de corecţie a datelor;

b)păstrarea grafului în memoria externă în forma de listă de adiacenţă;

c)extragerea informaţiei in una din cele trei forme la imprimantă si display.

La prima lucrare de laborator am elaborat o aplicație în consolă ce permite păstrate grafului în memoria calculatorului conform teoriei grafurilor:

- Matricea de incidentă
- Matricea de adiacență
- Lista de adiacentă

Pentru dezvoltarea aplicației am ales limbajul de programare Java, și librăria JgraphT ce facilitează lucrul cu grafurile prin multitudinea de funcții oferite. Programul elaborat permite intrucerea nodurilor, afișarea grafului introdus în consolă, afișare lor conform teoriei grafurilor și ștergerea arcelor. Programul este intuitiv pentru utilizator prin faptul că conține un meniu de alegere.

Listing-ul programului

```
import org.jgrapht.Graph;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.graph.SimpleGraph;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Graf_reprezentare {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Declararea unui graf de numere intregi
        Graph<Integer, DefaultEdge> graph = new
SimpleGraph<>(DefaultEdge.class);

        while (true) {
            printMenu(); // afisare meniu
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    addEdge(graph);
                    break;
                case 2:
                    displayGraph(graph);
                    break;
                case 3:
                    displayIncidenceMatrix(graph);
                    break;
                case 4:
                    displayAdjacencyMatrix(graph);
                    break;
                case 5:
                    displayAdjacencyList(graph);
                    break;
                case 6:
                    deleteEdge(graph);
                    break;
                case 0:
                    System.out.println("STOP program (tasta 0 )!");
                    System.exit(0);
                    break;
                default:
                    System.out.println("Optiune incorecta");
            }
```

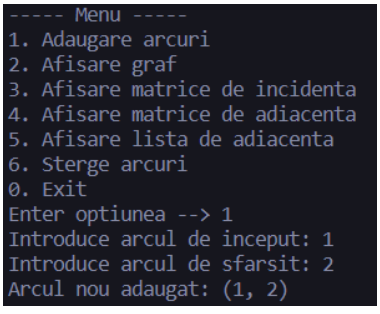
```
        }
    }

    // Afisare meniu
    private static void printMenu() {
        System.out.println("----- Menu -----");
        System.out.println("1. Adaugare arcuri");
        System.out.println("2. Afisare graf");
        System.out.println("3. Afisare matrice de incidenta");
        System.out.println("4. Afisare matrice de adiacenta");
        System.out.println("5. Afisare lista de adiacenta");
        System.out.println("6. Sterge arcuri");
        System.out.println("0. Exit");
        System.out.print("Enter optiunea --> ");
    }

    // Metoda pentru a adauga noduri noi
    private static void addEdge(Graph<Integer, DefaultEdge> graph) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce arcul de inceput: ");
        int source = scanner.nextInt(); // nodul sursa
        System.out.print("Introduce arcul de sfarsit: ");
        int target = scanner.nextInt(); // nodul destinatie

        graph.addVertex(source); // adaugam nodul sursa
        graph.addVertex(target); // adaugam nodul destinatieS
        graph.addEdge(source, target); // adaugam arcul dintre nodul sursa si
destinatie
        System.out.println("Arcul nou adaugat: (" + source + ", " + target +
")");
    }

    // Metoda pentru a sterge o legatura dintre noduri
    private static void deleteEdge(Graph<Integer, DefaultEdge> graph) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce arcul initial: ");
        int source = scanner.nextInt(); // declarre nodul sursa
        System.out.print("Introduce arcul final: ");
        int target = scanner.nextInt(); // declarare nodul destinatie

        // verificam daca graful contine arce
        if (graph.containsEdge(source, target)) {
            graph.removeEdge(source, target); // stergem
            System.out.println("Arcul sters: (" + source + ", " + target +
")");
        } else {
            System.out.println("Arcul nu a fost gasit: (" + source + ", " +
target + ")");
        }
```

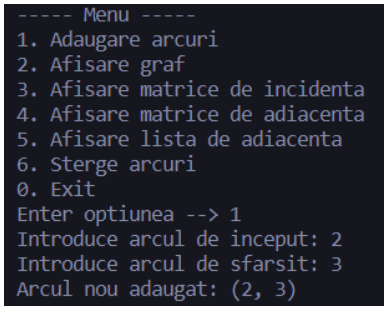
```
    }

    // Afisare graf
    private static void displayGraph(Graph<Integer, DefaultEdge> graph) {
        System.out.println("Graful: " + graph);
    }

    // Metoda de afisare a matricei de incidenta
    // 1, dacă nodul i este extremitatea finală a arcului j;
    // -1, dacă nodul i este extremitatea inițială a arcului j;
    // 0, dacă nodul i nu este extremitate a arcului j.
    private static void displayIncidenceMatrix(Graph<Integer, DefaultEdge>
graph) {
        System.out.println("Matricea de incidenta:");
        // Declaram o lista
        List<DefaultEdge> edges = new ArrayList<>(graph.edgeSet());
        // Parcugem nodurile grafului
        for (Integer vertex : graph.vertexSet()) {
            // Parcurgem arcurile
            for (DefaultEdge edge : edges) {
                // Declaram nodul sursa si destinatie
                Integer source = graph.getEdgeSource(edge);
                Integer target = graph.getEdgeTarget(edge);
                // Verificam conditia
                if (vertex.equals(source)) {
                    System.out.print("-1 ");
                } else if (vertex.equals(target)) {
                    System.out.print("1 ");
                } else {
                    System.out.print("0 ");
                }
            }
            System.out.println();
        }
    }

    // Metoda de afisare a matricei de adiacenta
    private static void displayAdjacencyMatrix(Graph<Integer, DefaultEdge>
graph) {
        System.out.println("Matricea de adiacenta:");
        // Parcurgem fiecare varf din arc
        for (Integer source : graph.vertexSet()) {
            for (Integer target : graph.vertexSet()) {
                // Verificam daca exista un arc intre varful sursa si
destinatie
                System.out.print(graph.containsEdge(source, target) ? "1 " :
"0 ");
            }
            System.out.println();
```

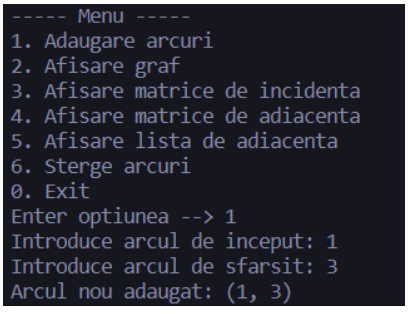
```
        }
    }

    // Metoda de afisare a listei de adiacenta
    private static void displayAdjacencyList(Graph<Integer, DefaultEdge>
graph) {
        System.out.println("Lista de adiacenta:");
        // Iterarea dupa noduri
        for (Integer vertex : graph.vertexSet()) {
            System.out.print(vertex + ": ");
            // Declareare unei liste pentru stocarea listei
            List<Integer> neighbors = new ArrayList<>();
            // Iterarea dupa arce
            for (DefaultEdge edge : graph.edgesOf(vertex)) {
                // Verificam daca nodul curent este sursa sau destinatia
arcului
                Integer neighbor = graph.getEdgeSource(edge).equals(vertex) ?
graph.getEdgeTarget(edge)
                        : graph.getEdgeSource(edge);
                // Adaugam elementele in lista
                neighbors.add(neighbor);
            }
            // Afisarea listei in ordine crescatoare
            neighbors.sort(Integer::compareTo);
            for (Integer neighbor : neighbors) {
                System.out.print(neighbor + " ");
            }
            // Afisam 0 la sfarsit
            System.out.print("0");
            System.out.println();
        }
    }
}
```

Rezultatul în consolă

1. Introducerea nodurilor , nodurile se introduc în perechi câte 2 , nodul sursă și destinație





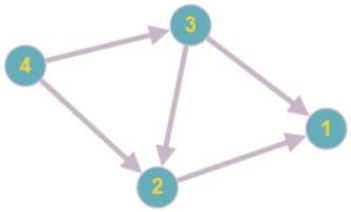


2. Afișare grafului

```
----- Menu -----
1. Adaugare arcuri
2. Afisare graf
3. Afisare matrice de incidenta
4. Afisare matrice de adiacenta
5. Afisare lista de adiacenta
6. Sterge arcuri
0. Exit
Enter optiunea --> 2
Graful: ([1, 2, 3], [{1,2}, {2,3}, {1,3}])
```

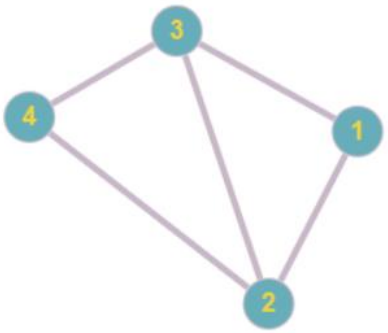
3. Afișarea matricei de incidenta

```
Graful: ([1, 2, 3, 4], [{1,2}, {2,3}, {1,3}, {3,4}, {2,4}])
----- Menu -----
1. Adaugare arcuri
2. Afisare graf
3. Afisare matrice de incidenta
4. Afisare matrice de adiacenta
5. Afisare lista de adiacenta
6. Sterge arcuri
0. Exit
Enter optiunea --> 3
Matricea de incidenta:
-1 0 -1 0 0
1 -1 0 0 -1
0 1 1 -1 0
0 0 0 1 1
```



4. Afișarea matricei de adiacenta

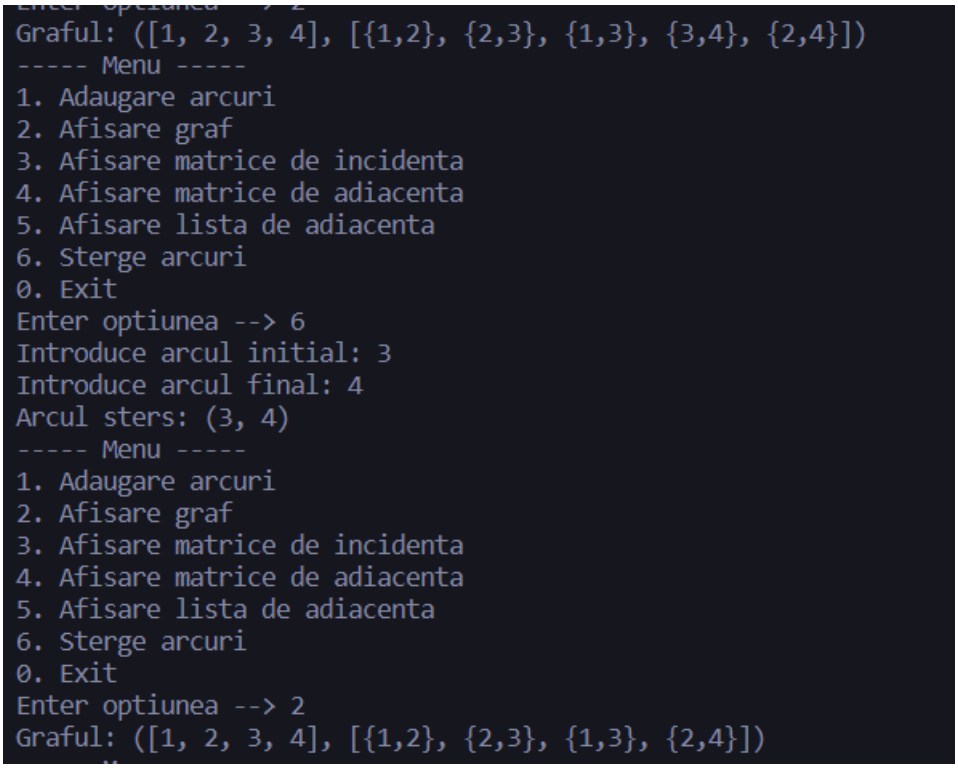
```
----- Menu -----
1. Adaugare arcuri
2. Afisare graf
3. Afisare matrice de incidenta
4. Afisare matrice de adiacenta
5. Afisare lista de adiacenta
6. Sterge arcuri
0. Exit
Enter optiunea --> 4
Matricea de adiacenta:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
```



5. Afișarea listei de adiacență

```
----- Menu -----
1. Adaugare arcuri
2. Afisare graf
3. Afisare matrice de incidenta
4. Afisare matrice de adiacenta
5. Afisare lista de adiacenta
6. Sterge arcuri
0. Exit
Enter optiunea --> 5
Lista de adiacenta:
1: 2 3 0
2: 1 3 4 0
3: 1 2 4 0
4: 2 3 0
```

6. Ștergere arcuri



Concluzie

În rezultatul elaborării acestui program am fost familiarizat cu elemente din teoria grafurilor, am elaborat pentru prima dată o aplicație în java utilizând librăria JgraphT. Am întâmpinat mai multe probleme la elaborarea listei de adiacentă. Acesta laborator m-a ajutat să înțeleg mai bine modul cum un graf este păstrat în memoria calculatorului.