**MINISTERUL EDUCAŢIEI, CULTURII ȘI CERCETĂRII**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică şi Microelectronică**

**Tehnologia Informației**

**Raport**

**Disciplina:**Matematica discretă

**Lucrarea de laborator nr. 3**

**Tema:** *“*ALGORITMUL DETERMINĂRII GRAFULUI DE ACOPERIRE*”*

|  |  |  |
| --- | --- | --- |
| **Student:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Raevschi Grigore TI-231** |
| **Coordonator:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Asist, uni Ciobanu Ecaterina** |
|  |  |  |

**Chişinău 2024**

Cuprins

[Cuprins 2](#_Toc159430510)

[Condiția problemei – Introducere 3](#_Toc159430511)

[Listing-ul programului 4](#_Toc159430512)

[Rezultatul în consolă 7](#_Toc159430513)

[Concluzie 10](#_Toc159430514)

# Condiția problemei – Introducere

Elaboraţi organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.

Utilizând procedurile de introducere a grafului în memoria CE din lucrarea Nr. 1, elaboraţi un program cu următoarele facilităţi:

* introducerea grafului care este dat sub formă de matrice de incidenţă, adiacenţă sau listă de adiacenţă;
* determinarea grafului de acoperire, pornind de la un vârf arbitrar;
* extragerea informaţiei la display sau imprimantă în oricare din formele numite.

# Listing-ul programului

import org.jgrapht.Graph;

import org.jgrapht.graph.DefaultEdge;

import org.jgrapht.graph.SimpleGraph;

import java.util.\*;

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 7:

deleteVertex(graph);

break;

case 8: // New case for depth-first traversal

depthFirstTraversal(graph);

break;

case 9: // New case for breadth-first traversal

breadthFirstTraversal(graph);

break;

case 10: // New case for determining the covering graph

determineCoveringGraph(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 noi");

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("7. Sterge nod");

System.out.println("8. Parcurgerea grafului in adancime");

System.out.println("9. Parcurgerea grafului in lungime");

System.out.println("10. Determinare graf de acoperire");

System.out.println("0. Exit");

System.out.print("Enter optiunea --> ");

}

private static void addEdge(Graph<Integer, DefaultEdge> graph) {

Scanner scanner = new Scanner(System.in);

System.out.print("Dati numarul de noduri ce vrei sa citesti: ");

int numVertices = scanner.nextInt();

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

System.out.print("enter nodul: ");

int vertex = scanner.nextInt();

graph.addVertex(vertex);

System.out.println("nodul adaugat: " + vertex);

}

System.out.print("Dati numarul de arcuri ce vrei sa citesti: ");

int numEdges = scanner.nextInt();

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

System.out.print("arcul de inceput: ");

int source = scanner.nextInt();

System.out.print("nodul final: ");

int target = scanner.nextInt();

if (!graph.containsEdge(source, target)) {

graph.addEdge(source, target);

System.out.println("arcul adaugat cu succes: (" + source + ", " + target + ")");

} else {

System.out.println("Arcul (" + source + ", " + target + ") există deja în graf.");

}

if (source == target) {

if (!graph.containsEdge(source, source)) {

graph.addEdge(source, source);

System.out.println("arcul adaugat cu succes: (" + source + ", " + source + ")");

} else {

System.out.println("Arcul (" + source + ", " + source + ") există deja în graf.");

}

}

}

}

// 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 + ")");

}

}

// Metoda pentru a sterge o legatura dintre noduri

private static void deleteVertex(Graph<Integer, DefaultEdge> graph) {

Scanner scanner = new Scanner(System.in);

System.out.print("Introduce nodul initial: ");

int source = scanner.nextInt(); // citire nod necesar de sters

// verificam daca graful contine nodul introdus

if (graph.containsVertex(source)) {

graph.removeVertex(source); // stergem nodul

System.out.println("Nodul sters: (" + source + ")");

} else {

System.out.println("Nodul nu a fost gasit: (" + source + ")");

}

}

// 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 iniţială a arcului j;

// 0, dacă nodul i nu este extremitate a arcului j.

// Metoda pentru afișarea matricei de incidență

private static void displayIncidenceMatrix(Graph<Integer, DefaultEdge> graph) {

System.out.println("Matricea de incidență:");

// Inițializați o matrice pentru a stoca matricea de incidență

int[][] incidenceMatrix = new int[graph.vertexSet().size()][graph.vertexSet().size()];

// Parcurgeți fiecare vârf din graf

int vertexIndex = 0;

for (Integer vertex : graph.vertexSet()) {

// Parcurgeți fiecare muchie din graf

int edgeIndex = 0;

for (DefaultEdge edge : graph.edgeSet()) {

// Obțineți nodurile sursă și destinație ale muchiei

Integer source = graph.getEdgeSource(edge);

Integer target = graph.getEdgeTarget(edge);

// Verificați dacă vârful curent este extremitatea inițială sau finală a muchiei și actualizați matricea de incidență corespunzător

if (vertex.equals(source)) {

incidenceMatrix[vertexIndex][edgeIndex] = -1;

} else if (vertex.equals(target)) {

incidenceMatrix[vertexIndex][edgeIndex] = 1;

} else {

incidenceMatrix[vertexIndex][edgeIndex] = 0;

}

edgeIndex++;

}

vertexIndex++;

}

// Afișați matricea de incidență

for (int[] row : incidenceMatrix) {

for (int value : row) {

System.out.print(value + " ");

}

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();

}

}

// Method for depth-first traversal

private static void depthFirstTraversal(Graph<Integer, DefaultEdge> graph) {

Scanner scanner = new Scanner(System.in);

System.out.println("dati nodul de pornire: ");

int startVertex = scanner.nextInt();

Set<Integer> visited = new HashSet<>();

depthFirstTraversalHelper(graph, startVertex, visited);

}

private static void depthFirstTraversalHelper(Graph<Integer, DefaultEdge> graph, int currentVertex,

Set<Integer> visited) {

if (!visited.contains(currentVertex)) {

System.out.println(currentVertex + " ");

visited.add(currentVertex);

for (DefaultEdge edge : graph.edgesOf(currentVertex)) {

Integer neighbor = graph.getEdgeSource(edge).equals(currentVertex) ? graph.getEdgeTarget(edge)

: graph.getEdgeSource(edge);

depthFirstTraversalHelper(graph, neighbor, visited);

}

}

}

// Method for breadth-first traversal

private static void breadthFirstTraversal(Graph<Integer, DefaultEdge> graph) {

Scanner scanner = new Scanner(System.in);

System.out.println("dati nodul de pornire: ");

int startVertex = scanner.nextInt();

Set<Integer> visited = new HashSet<>();

Queue<Integer> queue = new LinkedList<>();

visited.add(startVertex);

queue.add(startVertex);

while (!queue.isEmpty()) {

int currentVertex = queue.poll();

System.out.println(currentVertex + " ");

for (DefaultEdge edge : graph.edgesOf(currentVertex)) {

Integer neighbor = graph.getEdgeSource(edge).equals(currentVertex) ? graph.getEdgeTarget(edge)

: graph.getEdgeSource(edge);

if (!visited.contains(neighbor)) {

visited.add(neighbor);

queue.add(neighbor);

}

}

}

}

// Metoda determinarii grafului de acoperire

private static void determineCoveringGraph(Graph<Integer, DefaultEdge> graph) {

Scanner scanner = new Scanner(System.in);

System.out.println("Nodul de startare:");

int startVertex = scanner.nextInt();

Set<Integer> visited = new HashSet<>();

Queue<Integer> queue = new LinkedList<>();

visited.add(startVertex);

queue.add(startVertex);

// construim graful de acoperire

Graph<Integer, DefaultEdge> coveringGraph = new SimpleGraph<>(DefaultEdge.class);

coveringGraph.addVertex(startVertex);

while (!queue.isEmpty()) {

int currentVertex = queue.poll();

for (DefaultEdge edge : graph.edgesOf(currentVertex)) {

Integer neighbor = graph.getEdgeSource(edge).equals(currentVertex) ? graph.getEdgeTarget(edge)

: graph.getEdgeSource(edge);

if (!visited.contains(neighbor)) {

visited.add(neighbor);

queue.add(neighbor);

coveringGraph.addVertex(neighbor);

coveringGraph.addEdge(currentVertex, neighbor);

}

}

}

System.out.println("Graficul de acoperire pornind de la nodul " + startVertex + " este:");

System.out.println(coveringGraph);

displayAdjacencyList(coveringGraph);

displayIncidenceMatrix(coveringGraph);

displayAdjacencyMatrix(coveringGraph);

}

}

# Rezultatul în consolă

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

A screenshot of a computer program

Description automatically generated

Figura 1 (citirea nodurilor)

1. Citirea arcelor, se citesc a câte 2 nodul start și nodul final

A screenshot of a computer program

Description automatically generated

Figura 1-1 (citirea arcelor)

1. Afișare grafului

A screenshot of a computer

Description automatically generated

Figura 1-2 (afisarea)

1. Determinare graf de acoperire

A screenshot of a computer program

Description automatically generated

Figura 1-3 (graf de acoperire/lista de adiacenta pe baza lui)

1. Matricea de incidență pe baza grafului de acoperire

A screenshot of a computer screen

Description automatically generated

Figura 1-4 (matricea de incidență)

1. Matricea de adiacență pe baza grafului de acoperire

A screen shot of a computer

Description automatically generated

Figura 1-5 (matricea de adiacență)

# Concluzie

Prin implementarea algoritmului și programului pentru determinarea grafului de acoperire, cu posibilitatea de a porni procedura din oricare vârf al grafului, am reușit să dezvoltăm o soluție eficientă și flexibilă pentru analiza grafurilor. Organigrama algoritmului și programul procedurii ne-au ghidat într-un mod clar și structurat, iar utilizarea procedurilor de introducere a grafului în memoria calculatorului ne-a furnizat multiple modalități de a reprezenta și manipula datele grafului.

Prin intermediul programului elaborat, utilizatorii au facilități esențiale, precum introducerea grafului în diverse forme (matrice de incidență, adiacență sau listă de adiacență), determinarea grafului de acoperire pornind de la orice vârf ales și extragerea informațiilor relevante în formatul dorit, fie că este vorba despre afișare pe ecran sau imprimare.

Această implementare nu numai că ne permite să analizăm grafuri complexe cu ușurință, dar și să explorăm și să înțelegem mai bine structura și proprietățile acestora. În concluzie, dezvoltarea acestui program reprezintă un pas semnificativ în direcția facilitării analizei și manipulării grafurilor în contexte diverse.