МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №7**

**з дисципліни «Алгоритми та структури даних»**

**на тему: «Алгоритми на графах»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Куроп’ятник О. С.

Дніпро, 2020

**Лабораторна робота №7**

**Тема.** Алгоритми на графах.

**Мета**. Закріпити поняття про граф та отримати практичні навички обробки графів.

**Постановка задачі**

Написати програму мовою java для реалізації одного з алгоритмів обробки графів (за варіантом):

1.Уоршелла.

2.Дейкстри.

3.Прима.

4.Топологічного сортуванняна основі пошуку вглиб.

Програма повинна мати текстове меню і реалізовувати такий функціонал:

-додавання вершин та ребер у граф;

-перегляд представлення графа: матриці або списку суміжності;

-введення вхідних даних відповідно до особливостей алгоритму;

-обробка графа відповідно до алгоритму;

-виведення результатів роботи алгоритму графа.

Номер варіанту обирається за списком групи (підгрупи).

Для переставлення графа передбачити окремий клас, що має приватний набір полів для опису структури графа. Для методу, який реалізує алгоритм обробки графа відповідно до варіанта, дати повну специфікацію та представити опис алгоритму за методом покрокової деталізації.

**Текст програми**

Vertex.java

package com.company;  
  
public class Vertex {  
 private String Char;  
  
 public Vertex(String Char) {  
 this.Char = Char;  
 }  
  
 public String getChar() {  
 return Char;  
 }  
}

Graph.java

package com.company;  
  
import static java.util.Arrays.*fill*;  
  
public class Graph {  
 private Vertex[] vertices;  
 public int currAmountOfVer;  
 private int amount\_verteces;  
 private int[][] adjacency\_matrix;  
 private int INF = Integer.*MAX\_VALUE* / 2;  
  
 public Graph(int vertex) {  
 amount\_verteces = vertex;  
 adjacency\_matrix = new int[amount\_verteces + 1][amount\_verteces + 1];  
 for (int i = 0; i < amount\_verteces + 1; i++)  
 for (int j = 0; j < amount\_verteces + 1; j++)  
 adjacency\_matrix[i][j] = INF;  
  
 vertices = new Vertex[amount\_verteces + 1];  
 currAmountOfVer = -1;  
 }  
  
 public void addVertex(Vertex vertex) {  
 if(currAmountOfVer > amount\_verteces)  
 System.*out*.println("Немає місця для додавання Vertex");  
 else {  
 ++currAmountOfVer;  
 vertices[currAmountOfVer] = vertex;  
 }  
 }  
  
 public void addEdge(String start, String end, int Dist) {  
 try {  
 int Start = -1, End = -1;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(start.equals(vertices[i].getChar()))  
 Start = i;  
 if(end.equals(vertices[i].getChar()))  
 End = i;  
 }  
 adjacency\_matrix[Start][End] = Dist;  
 adjacency\_matrix[End][Start] = Dist;  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
  
 public int MST() {  
 int ret = 0; // вес MST  
 boolean[] used = new boolean [amount\_verteces]; // массив пометок  
 int[] dist = new int [amount\_verteces]; // массив расстояния  
  
  
 *fill*(dist, INF); // устанаавливаем расстояние до всех вершин INF  
 dist[0] = 0; // для начальной вершины положим 0  
  
 for (;;) {  
 int v = -1;  
  
 for (int nv = 0; nv < amount\_verteces; nv++) // перебираем вершины  
 if (!used[nv] && dist[nv] < INF && (v == -1 || dist[v] > dist[nv])) { // выбираем самую близкую непомеченную вершину  
 System.*out*.println("Самая близкая вершына " + nv);  
 v = nv;  
 }  
  
 if (v == -1)  
 break; // ближайшая вершина не найдена  
  
 used[v] = true; // помечаем ее  
  
 for (int nv = 0; nv < amount\_verteces; nv++)  
 if (!used[nv] && adjacency\_matrix[v][nv] < INF) // для всех непомеченных смежных  
 dist[nv] = Math.*min*(dist[nv], adjacency\_matrix[v][nv]); // улучшаем оценку расстояния  
  
 /\*for (int i = 0; i < amount\_verteces; i++)  
 System.out.println(dist[i]);;\*/  
 }  
  
 for (int v = 0; v < amount\_verteces; v++)  
 ret += dist[v];  
  
 return ret;  
 }  
  
 public void removeEdge(String start, String end) {  
 try {  
 int Start = -1, End = -1;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(start.equals(vertices[i].getChar()))  
 Start = i;  
 if(end.equals(vertices[i].getChar()))  
 End = i;  
 }  
 adjacency\_matrix[Start][End] = 0;  
 adjacency\_matrix[End][Start] = 0;  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
  
 public void deleteVertex(String Char) {  
 int indxOfVertexFound = -1;  
 boolean found = false;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(Char.equals(vertices[i].getChar())) {  
 indxOfVertexFound = i;  
 currAmountOfVer--;  
 found = true;  
 }  
 }  
 if(!found) {  
 System.*out*.println("Немає такої вершини!");  
 return;  
 }  
 for(int i = indxOfVertexFound; i < currAmountOfVer + 1; i++) {  
 vertices[i] = vertices[i + 1];  
 for (int j = 0; j < currAmountOfVer + 1; j++) {  
 adjacency\_matrix[i][j] = adjacency\_matrix[i + 1][j + 1];  
 }  
 }  
 }  
  
 public void print() {  
 for (int i = 0; i < currAmountOfVer + 1; i++)  
 System.*out*.print("\t" + vertices[i].getChar());  
  
 System.*out*.println();  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 System.*out*.print(vertices[i].getChar());  
 for (int j = 0; j < currAmountOfVer + 1; j++) {  
 if (adjacency\_matrix[i][j] == INF)  
 System.*out*.print("\t" + 0);  
 else  
 System.*out*.print("\t" + adjacency\_matrix[i][j]);  
 }  
 System.*out*.println();  
 }  
 }  
}

Main.java

package com.company;  
  
public class Main {  
  
 public static void main(String[] args) {  
 // write your code here  
 Graph graph = new Graph(6);  
 graph.addVertex(new Vertex("A"));  
 graph.addVertex(new Vertex("B"));  
 graph.addVertex(new Vertex("C"));  
 graph.addVertex(new Vertex("D"));  
 graph.addVertex(new Vertex("E"));  
 graph.addVertex(new Vertex("F"));  
  
 graph.addEdge("A", "B", 6);  
 graph.addEdge("A", "C", 15);  
 graph.addEdge("C", "F", 10);  
 graph.addEdge("B", "E", 25);  
 graph.addEdge("E", "F", 43);  
 graph.addEdge("D", "B", 3);  
 graph.addEdge("F", "B", 23);  
 graph.addEdge("E", "D", 9);  
  
 graph.print();  
  
 System.*out*.println("MST = " + graph.MST());  
  
 }  
}

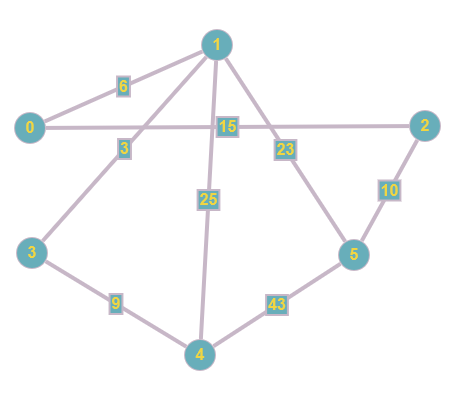


Рисунок 1. Представлення графа графічно.

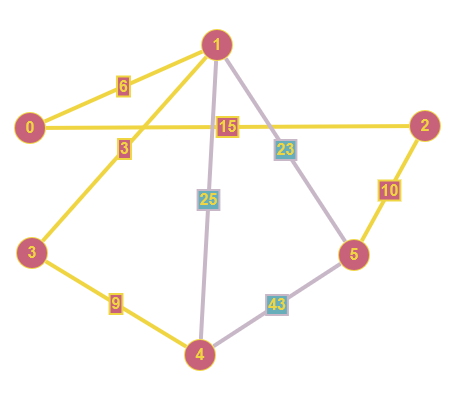


Рисунок 2. Пошук мінімального остовного дерева (результат MST = 43).

Матриця суміжності

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| A | 0 | 6 | 15 | 0 | 0 | 0 |
| B | 6 | 0 | 0 | 3 | 25 | 23 |
| C | 15 | 0 | 0 | 0 | 0 | 10 |
| D | 0 | 3 | 0 | 0 | 9 | 0 |
| E | 0 | 25 | 0 | 9 | 0 | 43 |
| F | 0 | 23 | 10 | 0 | 43 | 0 |

Висновок: на вхід алгоритму подається зв'язний неорієнтований граф. Для кожного ребра задається його вартість.

Спочатку береться довільна вершина і знаходиться ребро, інцидентне даної вершині і володіє найменшою вартістю. Знайдене ребро і з'єднуються їм дві вершини утворюють дерево. Потім, розглядаються ребра графа, один кінець яких - вже належить дереву вершина, а інший - ні; з цих ребер вибирається ребро найменшої вартості. Обиране на кожному кроці ребро приєднується до дереву. Зростання дерева відбувається до тих пір, поки не будуть вичерпані всі вершини вихідного графа.

Результатом роботи алгоритму є кістяк мінімальної вартості.