

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №6**

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

**на тему: «Алгоритми на графах»**

Виконав:

студент гр. ПЗ2011 Грива Я. А.

Прийняла:

Демидович І. М.

Дніпро, 2022

**Тема.** Алгоритми на графах.

**Мета.** Ознайомитися з поняттям графа. Отримати практичні навички реалізації різних представлень та обходу графів. Дослідити можливості застосування графів для моделювання реальних об’єктів та процесів.

**Завдання**

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

1. Уоршелла (для пошуку транзитивних шляхів).
2. Дейкстри.
3. Прима.
4. Топологічного сортування на основі пошуку вглиб.

Програма повинна мати текстове меню і реалізовувати такий функціонал: − додавання вершин та ребер у граф;

* перегляд внутрішнього представлення графа: матриці або списку суміжності;
* обхід графа в ширину та глибину;
* введення вхідних даних відповідно до особливостей алгоритму;
* обробка графа відповідно до алгоритму − виведення результатів роботи алгоритму обробки графа.

Для кожного пункту меню передбачити зворотній зв'язок у вигляді виведення результатів виконання дії та/або текстового повідомлення.

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.

Вибір виду графу: орієнтований чи неорієнтований зробити врахувавши обмеження, які накладає алгоритм обробки згідно варіанту.

*Індивідуальне завдання*

Алгоритм прима

**Текст програми**

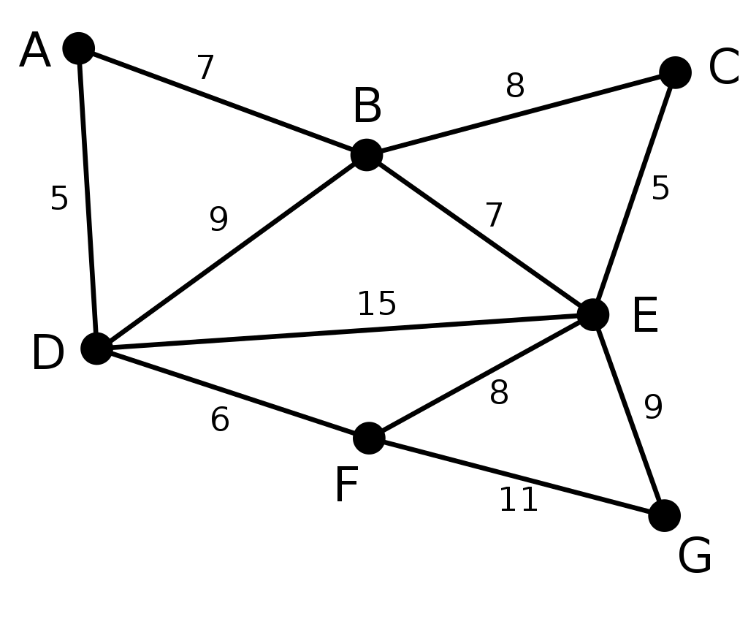
*Main.java*

package com.company;  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 int adjMatrix[][] = new int[][]{  
 *// A B C D E F G* {0, 7, 0, 5, 0, 0, 0}, *//A* {7, 0, 8, 9, 7, 0, 0}, *//B* {0, 8, 0, 0, 5, 0, 0}, *//C* {5, 9, 0, 0, 15, 6, 0}, *//D* {0, 7, 5, 15, 0, 8, 9}, *//E* {0, 0, 0, 6, 8, 0, 11}, *//F* {0, 0, 0, 0, 9, 11, 0}, *//G* };  
  
 Graph graph = new Graph(adjMatrix);  
  
 int choice = 0;  
 while (true) {  
 System.*out*.println("Виберіть дію:");  
 System.*out*.println("1. Додати вершину");  
 System.*out*.println("2. Видалити вершину");  
 System.*out*.println("3. Обхід графа в глибину");  
 System.*out*.println("4. Обхід графа в ширину");  
 System.*out*.println("5. Вивід матриці суміжності");  
 System.*out*.println("6. Алгоритм Прима");  
 System.*out*.println("7. Вихід");  
 choice = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 switch (choice) {  
 case 1:  
 System.*out*.println("Введіть початкову вершину:");  
 int vertex = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 System.*out*.println("Введіть кінцеву вершину:");  
 int vertex2 = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 System.*out*.println("Введіть вагу:");  
 int weight = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 if(graph.addEdge(vertex, vertex2, weight)){  
 System.*out*.println("Неможливо додати вершину");  
 }  
 else {  
 System.*out*.println("Вершина додана");  
 }  
 break;  
 case 2:  
 System.*out*.println("Введіть початкову вершину:");  
 vertex = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 System.*out*.println("Введіть кінцеву вершину:");  
 vertex2 = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 if(!graph.removeEdge(vertex, vertex2))  
 System.*out*.println("Неможливо видалити вершину");  
 else  
 System.*out*.println("Вершина видалена");  
 break;  
 case 3:  
 System.*out*.println("Введіть початкову вершину:");  
 vertex = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 graph.dfs(vertex);  
 break;  
 case 4:  
 System.*out*.println("Введіть початкову вершину:");  
 vertex = Integer.*parseInt*(new Scanner(System.*in*).nextLine());  
 graph.bfs(vertex);  
 break;  
 case 5:  
 graph.printAjdetencyMatrix();  
 break;  
 case 6:  
 graph.primMST();  
 break;  
 case 7:  
 System.*exit*(0);  
 break;  
 default:  
 System.*out*.println("Невірний ввід");  
 }  
 }  
}  
}

*Graph.java*

package com.company;  
  
import java.util.Queue;  
  
  
class Graph {  
 int vertices;  
 int matrix[][];  
  
 public Graph(int vertex) {  
 this.vertices = vertex;  
 matrix = new int[vertex][vertex];  
 }  
  
 public Graph(int[][] matrix) {  
 this.vertices = matrix.length;  
 this.matrix = matrix;  
 }  
  
 public boolean addEdge(int source, int destination, int weight) {  
 if (source < 0 || destination < 0 || source >= vertices || destination >= vertices)  
 return false;  
 matrix[source][destination] = weight;  
 matrix[destination][source] = weight;  
 return true;  
 }  
  
 public boolean removeEdge(int start, int end)  
 {  
 if (start < 0 || end < 0 || start >= vertices || end >= vertices)  
 return false;  
 matrix[start][end] = 0;  
 matrix[end][start] = 0;  
 return true;  
 }  
  
 *//get the vertex with minimum key which is not included in MST* int getMinimumVertex(boolean [] mst, int [] key){  
 int minKey = Integer.*MAX\_VALUE*;  
 int vertex = -1;  
 for (int i = 0; i <vertices ; i++) {  
 if(mst[i]==false && minKey>key[i]){  
 minKey = key[i];  
 vertex = i;  
 }  
 }  
 return vertex;  
 }  
  
 public void printAjdetencyMatrix()  
 {  
 for (int i = 0; i < vertices; i++) {  
 for (int j = 0; j < vertices; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 public void bfs(int start)  
 {  
 if (start < 0 || start >= vertices) {  
 System.*out*.println("Неправильний початковий вузол");  
 return;  
 }  
 Queue<Integer> queue = new java.util.LinkedList<>();  
 queue.add(start);  
 boolean [] visited = new boolean[vertices];  
 visited[start] = true;  
  
 while (!queue.isEmpty()) {  
 int current = queue.poll();  
 System.*out*.print((current) + " ");  
 for (int i = 0; i < vertices; i++) {  
 if (!visited[i] && matrix[current][i] > 0) {  
 queue.add(i);  
 visited[i] = true;  
 }  
 }  
 }  
 }  
  
 *//обхід графа в глибину* public void dfs(int start)  
 {  
 if (start < 0 || start >= vertices) {  
 System.*out*.println("Неправильний початковий вузол");  
 return;  
 }  
 boolean [] visited = new boolean[vertices];  
 dfs(start, visited);  
 }  
 private void dfs(int start, boolean [] visited)  
 {  
 visited[start] = true;  
 System.*out*.print((start) + " ");  
 for (int i = 0; i < vertices; i++) {  
 if (!visited[i] && matrix[start][i] > 0) {  
 dfs(i, visited);  
 }  
 }  
 }  
  
 class ResultSet{  
 int parent;  
 int weight;  
 }  
  
 public void primMST(){  
 boolean[] mst = new boolean[vertices];  
 ResultSet[] resultSet = new ResultSet[vertices];  
 int [] key = new int[vertices];  
  
 for (int i = 0; i <vertices ; i++) {  
 key[i] = Integer.*MAX\_VALUE*;  
 resultSet[i] = new ResultSet();  
 }  
  
 key[0] = 0;  
 resultSet[0] = new ResultSet();  
 resultSet[0].parent = -1;  
  
 for (int i = 0; i <vertices ; i++) {  
  
 int vertex = getMinimumVertex(mst, key);  
  
 mst[vertex] = true;  
  
 for (int j = 0; j <vertices ; j++) {  
 if(matrix[vertex][j]>0){  
 if(mst[j]==false && matrix[vertex][j]<key[j]){  
 key[j] = matrix[vertex][j];  
 resultSet[j].parent = vertex;  
 resultSet[j].weight = key[j];  
 }  
 }  
 }  
 }  
 printMST(resultSet);  
 }  
  
 private void printMST(ResultSet[] resultSet){  
 int total\_min\_weight = 0;  
 System.*out*.println("Мінімальне остове дерево: ");  
 for (int i = 1; i <vertices ; i++) {  
 System.*out*.println("Ребро: " + i + " - " + resultSet[i].parent +  
 " вага: " + resultSet[i].weight);  
 total\_min\_weight += resultSet[i].weight;  
 }  
 System.*out*.println("Загальна вага: " + total\_min\_weight);  
 }  
}

**Опис тестового прикладу**



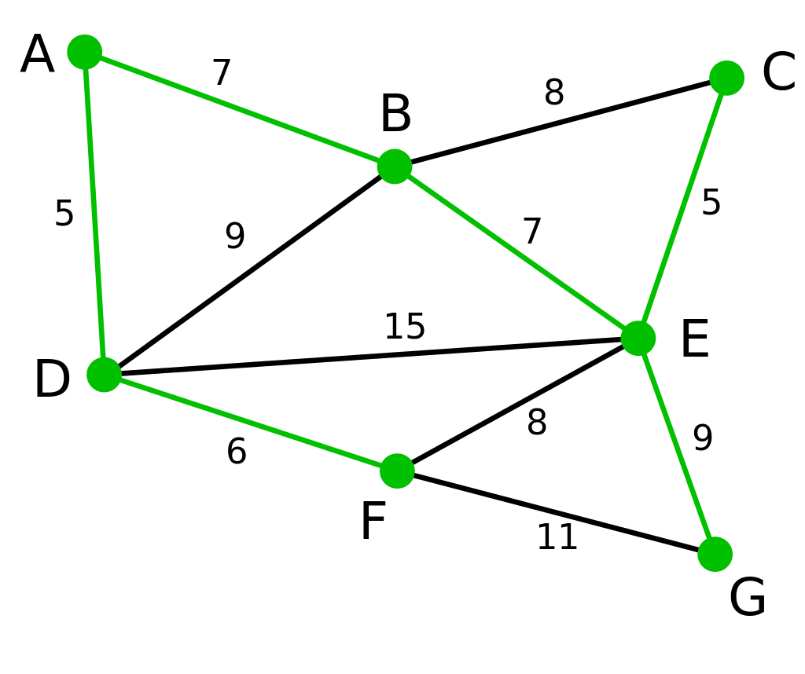
*Рис.1 – Графічне представлення графа*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *0* | *7* | *0* | *5* | *0* | *0* | *0* |
| *7* | *0* | *8* | *9* | *7* | *0* | *0* |
| *0* | *8* | *0* | *0* | *5* | *0* | *0* |
| *5* | *9* | *0* | *0* | *15* | *6* | *0* |
| *0* | *7* | *5* | *15* | *0* | *8* | *9* |
| *0* | *0* | *0* | *6* | *8* | *0* | *11* |
| *0* | *0* | *0* | *0* | *9* | *11* | *0* |

*Табл. 1 – Матриця суміжності*

|  |  |
| --- | --- |
| Вузол | Суміжний з |
| A | B, D |
| B | A, D, E, C |
| C | B, E |
| D | A, B, E, F |
| E | C, B, D, F |
| F | D, E, G |
| G | F, E |

*Табл. 2 – список суміжності*

**

*Рис.2 – Графічне представлення мінімального кістякового дерева*

**Висновок**

Під час виконання лабораторної роботи реалізовував орієнтований граф та алгоритм дейкстри для побудови мінімального кістякового дерева.

Графи застосовуються в різних наукових областях. Перше, що мені приходе на думку, це навігатор, який прокладує шлях у враховуючи пробки на дорогах або інші перешкоди. Графи застосовуються в різних областях науки, без них вирішення деяких задач було б складне або взагалі неможливе.

Для представлення графу я використовував матрицю суміжності, головною її перевагою є простота реалізації, проте вона потребує більше пам’яті на відміну від списка суміжності.