

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 Кулик С. В.

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 дейкстри

**Текст програми**

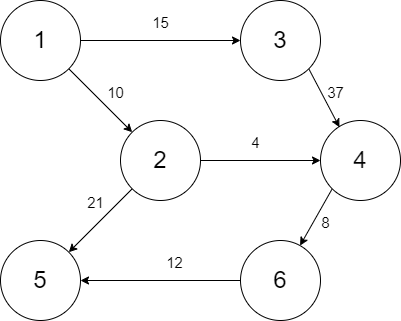
*Main.java*

package com.company;  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 int adjMatrix[][] = new int[][]{  
 {0, 10, 15, 0, 0, 0},  
 {0, 0, 0, 4, 21, 0},  
 {0, 0, 0, 37, 0, 0},  
 {0, 0, 0, 0, 0, 8},  
 {0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 12, 0}  
 };  
 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 зв'язків вершин та ваги");  
 System.out.println("6. Вивід матриці суміжності");  
 System.out.println("7. Пошук найкоротшого шляху");  
 System.out.println("8. 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 - 1);  
 break;  
 case 4:  
 System.out.println("Введіть початкову вершину:");  
 vertex = Integer.parseInt(new Scanner(System.in).nextLine());  
 graph.bfs(vertex - 1);  
 break;  
 case 5:  
 graph.printEdgeWithWeight();  
 break;  
 case 6:  
 graph.printAjdetencyMatrix();  
 break;  
 case 7:  
 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());  
 System.out.println("Найкоротший шлях між вершинами " + vertex + " та " + vertex2 + ": " + graph.findShortestWay(vertex - 1, vertex2 - 1));  
 break;  
 case 8:  
 System.exit(0);  
 break;  
 default:  
 System.out.println("Невірний ввід");  
 }  
 }  
}  
}

*Graph.java*

package com.company;  
import java.util.Queue;  
  
public class Graph   
{  
 private int [][] ajdetencyMatrix;  
  
 public Graph(int size)  
 {  
 ajdetencyMatrix = new int[size][size];  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 ajdetencyMatrix[i][j] = 0;  
 }  
 }  
 }  
  
 public Graph(int [][] matrix)  
 {  
 ajdetencyMatrix = matrix;  
 }  
  
 //додати ребро  
 public boolean addEdge(int start, int end, int weight)  
 {  
 if (start < 0 || end < 0 || start >= ajdetencyMatrix.length || end >= ajdetencyMatrix.length)  
 return false;  
 ajdetencyMatrix[start][end] = weight;  
 return true;  
 }  
  
 //видалити ребро  
 public boolean removeEdge(int start, int end)  
 {  
 if (start < 0 || end < 0 || start >= ajdetencyMatrix.length || end >= ajdetencyMatrix.length)  
 return false;  
 ajdetencyMatrix[start][end] = 0;  
 return true;  
 }  
  
 //взяти вагу ребра  
 public int getWeight(int start, int end)  
 {  
 if (start < 0 || end < 0 || start >= ajdetencyMatrix.length || end >= ajdetencyMatrix.length)  
 return 0;  
 return ajdetencyMatrix[start][end];  
 }  
  
 //пошук найкоротшого шляху  
 public int findShortestWay(int start, int end)  
 {  
 if (start < 0 || end < 0 || start >= ajdetencyMatrix.length || end >= ajdetencyMatrix.length)  
 return 0;  
 int [] distances = new int[ajdetencyMatrix.length];  
 boolean [] visited = new boolean[ajdetencyMatrix.length];  
  
 //заповнення вершин початковими значеннями  
 for (int i = 0; i < distances.length; i++) {  
 distances[i] = Integer.MAX\_VALUE;  
 visited[i] = false;  
 }  
 distances[start] = 0; //відстань до початкової вершини рівна 0  
  
 for (int i = 0; i < distances.length; i++) {  
 int minDistance = Integer.MAX\_VALUE;  
 int minIndex = -1;  
 for (int j = 0; j < distances.length; j++) {  
 //якщо вершина не відвідана і відстань до неї менша за найкоротшу  
 //то оновлюємо найкоротшу відстань і вершину  
 if (!visited[j] && distances[j] < minDistance) {  
 minDistance = distances[j];  
 minIndex = j;  
 }  
 }  
 if (minIndex == -1)  
 break;  
 visited[minIndex] = true;  
 for (int j = 0; j < distances.length; j++) {  
 //додавання найкоротшої відстані до вершини  
 if (!visited[j] && ajdetencyMatrix[minIndex][j] > 0 && distances[j] > distances[minIndex] + ajdetencyMatrix[minIndex][j]) {  
 distances[j] = distances[minIndex] + ajdetencyMatrix[minIndex][j];  
 }  
 }  
 }  
 return distances[end];  
 }  
  
 //друк матриці суміжності  
 public void printAjdetencyMatrix()  
 {  
 for (int i = 0; i < ajdetencyMatrix.length; i++) {  
 for (int j = 0; j < ajdetencyMatrix.length; j++) {  
 System.out.print(ajdetencyMatrix[i][j] + " ");  
 }  
 System.out.println();  
 }  
 }  
  
 //обхід графа в ширину  
 public void bfs(int start)  
 {  
 if (start < 0 || start >= ajdetencyMatrix.length) {  
 System.out.println("Неправильний початковий вузол");  
 return;  
 }  
 Queue<Integer> queue = new java.util.LinkedList<>();  
 queue.add(start);  
 boolean [] visited = new boolean[ajdetencyMatrix.length];  
 visited[start] = true;  
  
 while (!queue.isEmpty()) {  
 int current = queue.poll();  
 System.out.print((current + 1) + " ");  
 for (int i = 0; i < ajdetencyMatrix.length; i++) {  
 if (!visited[i] && ajdetencyMatrix[current][i] > 0) {  
 queue.add(i);  
 visited[i] = true;  
 }  
 }  
 }  
 }  
  
 //обхід графа в глибину  
 public void dfs(int start)  
 {  
 if (start < 0 || start >= ajdetencyMatrix.length) {  
 System.out.println("Неправильний початковий вузол");  
 return;  
 }  
 boolean [] visited = new boolean[ajdetencyMatrix.length];  
 dfs(start, visited);  
 }  
 private void dfs(int start, boolean [] visited)  
 {  
 visited[start] = true;  
 System.out.print((start + 1) + " ");  
 for (int i = 0; i < ajdetencyMatrix.length; i++) {  
 if (!visited[i] && ajdetencyMatrix[start][i] > 0) {  
 dfs(i, visited);  
 }  
 }  
 }  
  
 //вивід зв'язків в з вагою  
 public void printEdgeWithWeight()  
 {  
 for (int i = 0; i < ajdetencyMatrix.length; i++) {  
 for (int j = 0; j < ajdetencyMatrix.length; j++) {  
 if (ajdetencyMatrix[i][j] != 0)  
 System.out.println("Ребро від " + (i + 1) + " до " + (j + 1) + " вага " + ajdetencyMatrix[i][j]);  
 }  
 }  
 }  
}

**Опис тестового прикладу**

****

*Рис.1 – Графічне представлення графа*

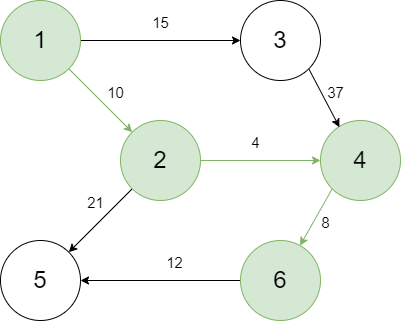
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *0* | *10* | *15* | *0* | *0* | *0* |
| *0* | *0* | *0* | *4* | *21* | *0* |
| *0* | *0* | *0* | *37* | *0* | *0* |
| *0* | *0* | *0* | *0* | *0* | *8* |
| *0* | *0* | *0* | *0* | *0* | *0* |
| *0* | *0* | *0* | *0* | *12* | *0* |

*Табл. 1 – Матриця суміжності*

|  |  |
| --- | --- |
| Вузол | Суміжний з |
| 1 | 3, 2 |
| 2 | 5, 4 |
| 3 | 4 |
| 4 | 6 |
| 5 | - |
| 6 | 5 |

*Табл. 2 – список суміжності*

Мої очікування, щодо обходу графа вширину та глибину вцілому справдилися, при обході в ширину алгоритм спочатку виводить всих сусідів початкового обраного вузла, а поті сусідів судів і так до кінця. Так як граф орієнтований, то елементи у яких немає доступу до сусідів не виводяться. Обхід в глибину реалізований за допомогою рекурсії, тому алгоритм спочатку виводить елементи на більшій глибині від початкового вузла. Алгоритм дейкстри працює правильно, і корректно знаходить найкоротший шлях, у випадку якщо шляху до вузла немає то він повертає нескінченність, у випадку з мовою джава – це константа, яка вказує на найбільше ціле число типу Integer.



*Рис. 2 – Найкоротший шлях від вузла 1 до вузла 6*

**Висновок**

Під час виконання лабораторної роботи реалізовував орієнтований граф та алгоритм дейкстри для пошуку найкоротшого шляху.

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. Також графи можна використовувати для написання штучного інтелекту для гри в шахи або рух песонажа в грі по клітинкам, можна досліджувати лабіринт, використовувати графи в фізиці, в електросхемах, авіалініях або залізничних маршрутах, фізиці хімії, також сузір’я на небі теж являє собою граф.

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

У підсумку можна сказати, що графи – невід’ємна частина життя, вони використовуються у багатьох областях науки.