

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 Савенко Я. О.

Прийняла:

Куроп’ятник О. С.

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*

Топологічне сортування на основі пошуку вглиб.

**Текст програми**

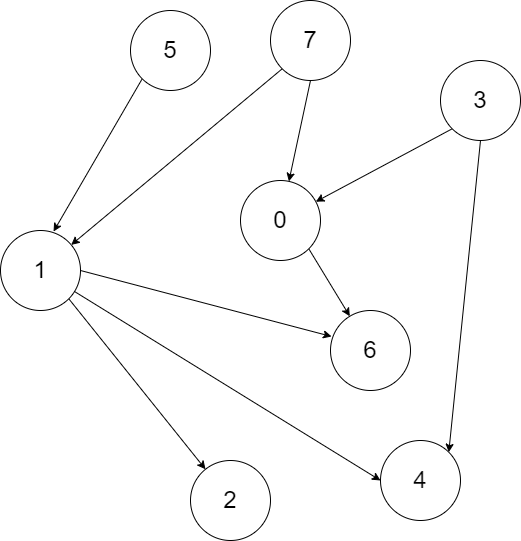
*Main.java*

**package** com.company;  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
  
 **int** adjMatrix[][] = **new int**[][]{  
 {0, 0, 0, 0, 0, 0, 1, 0},  
 {0, 0, 1, 0, 1, 0, 1, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0},  
 {1, 0, 0, 0, 1, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 1, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0},  
 {1, 1, 0, 0, 0, 0, 0, 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 матриці суміжності"**);  
 System.***out***.println(**"6. Топологічне сортування"**);  
 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());  
 **if**(!graph.addEdge(vertex, vertex2)){  
 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.topologicalSort();  
 **break**;  
 **case** 7:  
 System.*exit*(0);  
 **break**;  
 **default**:  
 System.***out***.println(**"Невірний ввід"**);  
 }  
 }  
}  
}

*Graph.java*

**package** com.company;  
**import** java.util.Queue;  
**import** java.util.Stack;  
  
**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)  
 {  
 **if** (start < 0 || end < 0 || start >= **ajdetencyMatrix**.**length** || end >= **ajdetencyMatrix**.**length**)  
 **return false**;  
 **ajdetencyMatrix**[start][end] = 1;  
 **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**;  
 }  
  
 *//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) + **" "**);  
 **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) + **" "**);  
 **for** (**int** i = 0; i < **ajdetencyMatrix**.**length**; i++) {  
 **if** (!visited[i] && **ajdetencyMatrix**[start][i] > 0) {  
 dfs(i, visited);  
 }  
 }  
 }  
  
 *//топологічне сортування* **public void** topologicalSort()  
 {  
 **int** [] indegree = **new int**[**ajdetencyMatrix**.**length**];  
 **for** (**int** i = 0; i < **ajdetencyMatrix**.**length**; i++) {  
 **for** (**int** j = 0; j < **ajdetencyMatrix**.**length**; j++) {  
 **if** (**ajdetencyMatrix**[j][i] == 1) {  
 indegree[i]++;  
 }  
 }  
 }  
 Stack<Integer> stack = **new** Stack<Integer>();  
 **for** (**int** i = 0; i < indegree.**length**; i++) {  
 **if** (indegree[i] == 0) {  
 stack.push(i);  
 }  
 }  
 **while** (!stack.isEmpty()) {  
 **int** vertex = stack.pop();  
 System.***out***.print(vertex + **" "**);  
 **for** (**int** i = 0; i < indegree.**length**; i++) {  
 **if** (**ajdetencyMatrix**[vertex][i] == 1) {  
 indegree[i]--;  
 **if** (indegree[i] == 0) {  
 stack.push(i);  
 }  
 }  
 }  
 }  
 System.***out***.println();  
 }  
  
}

**Опис тестового прикладу**



*Рис.1 – Графічне представлення графа*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

*Табл. 1 – Матриця суміжності*

|  |  |
| --- | --- |
| Вузол | Суміжний з |
| 0 | 6 |
| 1 | 2, 4, 6 |
| 2 | – |
| 3 | 0, 4 |
| 4 | – |
| 5 | 1 |
| 6 | – |
| 7 | 0, 1 |

*Табл. 2 – список суміжності*

Обхід графа в ширину та глибину справдився, все працює корректно. Вершини до яких немає шляху не відображається, це можна виправити написавши код, де буде робитися перевірка на всі пройдені вершини. Алгоритм топологічного сортування працює також правильно, але результат може розбігатися з очікуваним, так як може буде декілька варіантів топологічного сортування одного графу.

**Висновок**

Під час виконання лабораторної роботи реалізовував граф та алгоритм топологічного сортування.

Графи використовуються в математикі, фізикі, хімії, програмуванні, комп’ютерних іграх.

Для реалізації графа в лабораторній роботі викостовувалась матриця суміжності. На мою думку вона має більш просту реалізацію, ніж список суміжності, проте займає більше місця в оперативній пам’яті.