Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Звіт

з лабораторної роботи №6

«Алгоритми для роботи з графами»

з дисципліни «Алгоритми та структури даних»

Виконала:

ст.гр. ІП-22-4

Томин Є.В.

Перевірив:

Григорчук Л. І.

Івано-Франківськ

2023

**Тема.** Алгоритми для роботи з графами

**Мета:** Набуття практичних вмінь і навичок при представленні заданих графів різними способами та можливістю їх комп’ютерної реалізації

**Теоретичні відомості**

**Графи**

Нехай граф G= (V,E) містить вершини у, та ребра е;, і= 1,..,., j= 1,..,т. Визначимо відповідність Г, яка вказує, як звʼязані між собою вершини. Відповідність Г є багатозначним відображенням множини / в множину. Тоді граф можна позначати G = (V,Г).

Для простих графів відповідність Г визначає для кожної вершини v; суміжні з нею.

Для псевдографів відповідність Г визначає для кожної вершини у, суміжні з нею та додає У , якщо в цій вершині існує петля.

Для орієнтованих графів відповідність Г визначає для кожної вершини вершини у, якщо існує ребро п.ч; та додає вершину у, якщо в цій вершині існує петля.

**Матриці інцидентності**

Для простих графів:



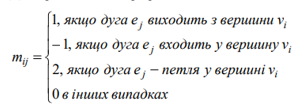
Для простого графа нема однакових стовпців і в кожному стовпці є точно по

дві 1.

Для мультографа - будуть однакові стовпці.

Для псевдографів для позначення петлі використовують 2.

Для орієнтованих графів:

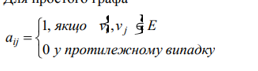


Для графа, зображеного на рис.10.1:



**Матриця суміжності 1= d , 1=1,.,1**

Для простого графа



Для неорієнтованого графа - симетрична і а; = 0, = j.

Для псевдографа елемент а дорівнює кількості ребер, що зʼєднують

відповідні вершини. Петля - а,i = 1,1= j.

Для орієнтованого графа - аналогічно (тільки матриця, взагалі кажучи, несиметрична).

Для орієнтованого мультографа елементи матриці рівні кількості ребер, що зʼєднують відповідні вершини.

**Список ребер** (найекономніший спосіб щодо памʼяті) - список пар, що відповідають ребрам графа.

**Списки суміжності** - задання графа відповідністю Г. використовується масив списків (по одному списку на кожну вершину), який для кожної вершини у містить у довільному порядку (вказівники на) вершини з множини Г(v,).

Списки суміжності доцільно використовувати у тих випадках, коли матриця суміжності є сильно розрідженою.

**Варіант 9**

**Завдання для самостійного виконання.**

Для орієнтованих та неорієнтованих графів зі своїх варіантів (варіант вибирається згідно номера по журналу, варіанти завдань наведено у додатку Б) виконати наступні завдання:

1. Створити програму, яка буде зберігати в компʼютері заданий граф наступними

способами:

- а) матрицею суміжності

- б) матрицею інцидентності

- в) списком ребер

- г) списком суміжності.

Передбачити вивід кожного представлення на екран.

2. Написати програму, яка виконуватиме наступні завдання:

- за заданою матрицею суміжності побудувати матрицю інцидентності;

- за заданою матрицею інцидентності побудувати список ребер;

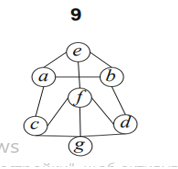
- за заданою матрицею суміжності побудувати список суміжності.

- за заданою матрицею інцидентності побудувати матрицю суміжності;

- за заданою матрицею суміжності побудувати список ребер;

-за заданою матрицею інцидентності побудувати список суміжності.

**Неорієнтований граф**

****

**Код програми**

**Завдання 1**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Функція для виведення матриці на екран

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

}

// Функція для виведення списку на екран

void printList(const vector<vector<int>>& list) {

for (size\_t i = 0; i < list.size(); ++i) {

cout << "Вершина " << i << ": ";

for (int neighbor : list[i]) {

cout << neighbor << " ";

}

cout << endl;

}

}

int main() {

int numVertices, numEdges;

cout << "Введіть кількість вершин: ";

cin >> numVertices;

cout << "Введіть кількість ребер: ";

cin >> numEdges;

// Ініціалізація графа

vector<vector<int>> adjacencyMatrix(numVertices, vector<int>(numVertices, 0));

vector<vector<int>> incidenceMatrix(numVertices, vector<int>(numEdges, 0));

vector<pair<string, string>> edgeList;

vector<vector<int>> adjacencyList(numVertices);

// Введення ребер

cout << "Введіть ребра (формат: початкова\_вершина кінцева\_вершина):" << endl;

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

string startVertex, endVertex;

cin >> startVertex >> endVertex;

edgeList.push\_back({startVertex, endVertex});

// Приведення рядків до числових значень (в даному випадку, можна використовувати і hash-функції)

int startVertexNum = hash<string>{}(startVertex) % numVertices;

int endVertexNum = hash<string>{}(endVertex) % numVertices;

adjacencyMatrix[startVertexNum][endVertexNum] = 1;

incidenceMatrix[startVertexNum][i] = 1;

incidenceMatrix[endVertexNum][i] = -1;

adjacencyList[startVertexNum].push\_back(endVertexNum);

adjacencyList[endVertexNum].push\_back(startVertexNum); // для неорієнтованого графа

}

// Виведення графа в різних представленнях

cout << "Матриця суміжності:" << endl;

printMatrix(adjacencyMatrix);

cout << "Матриця інцидентності:" << endl;

printMatrix(incidenceMatrix);

cout << "Список ребер:" << endl;

for (const auto& edge : edgeList) {

cout << edge.first << " " << edge.second << endl;

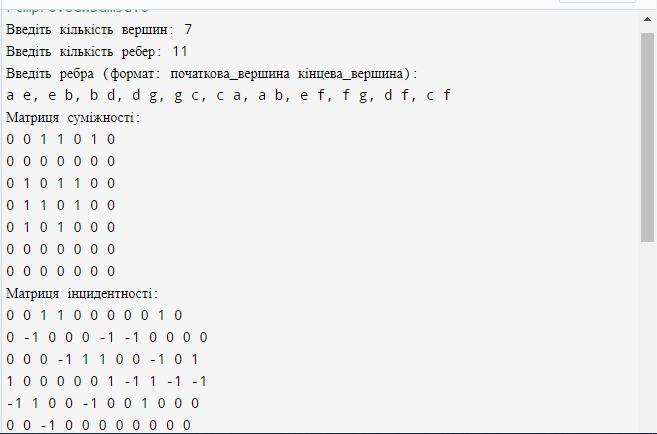
}

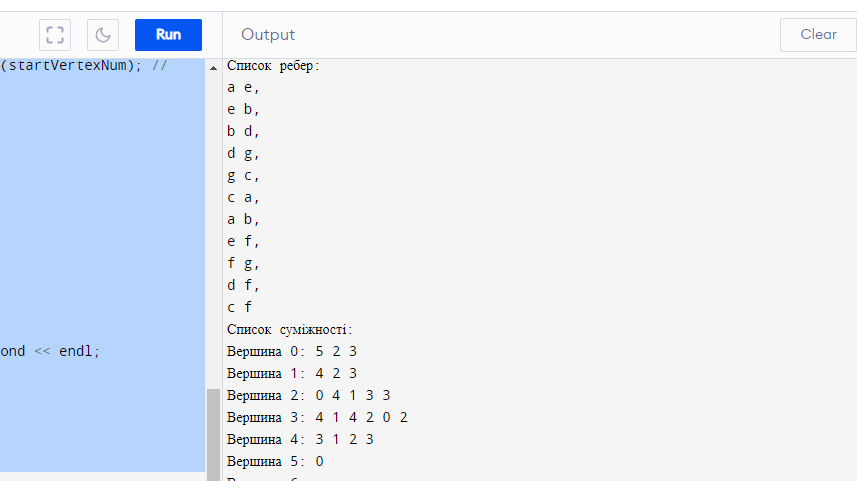
cout << "Список суміжності:" << endl;

printList(adjacencyList);

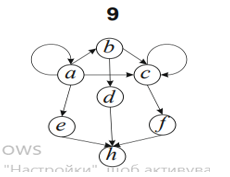
return 0;

}



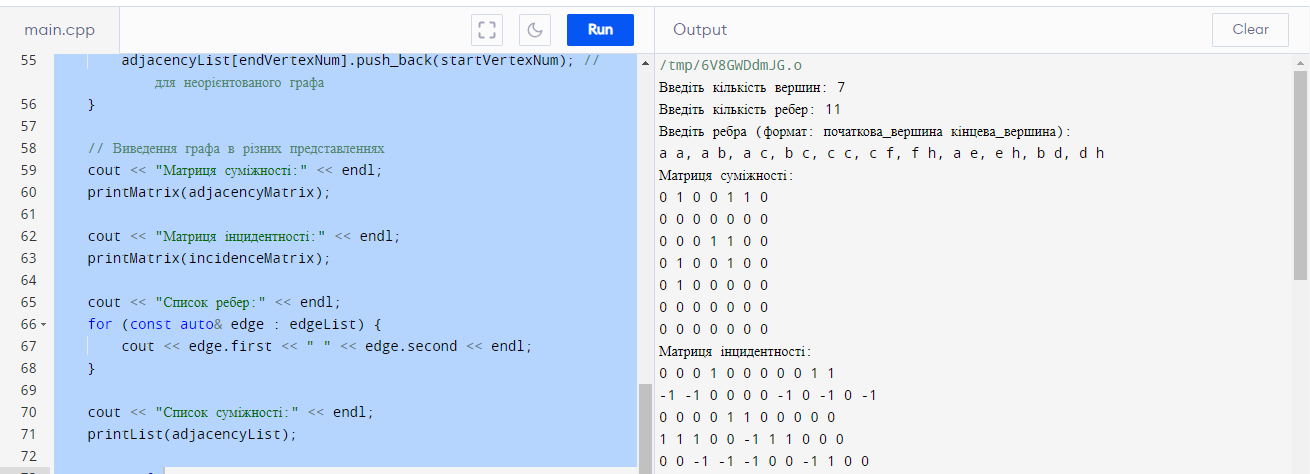


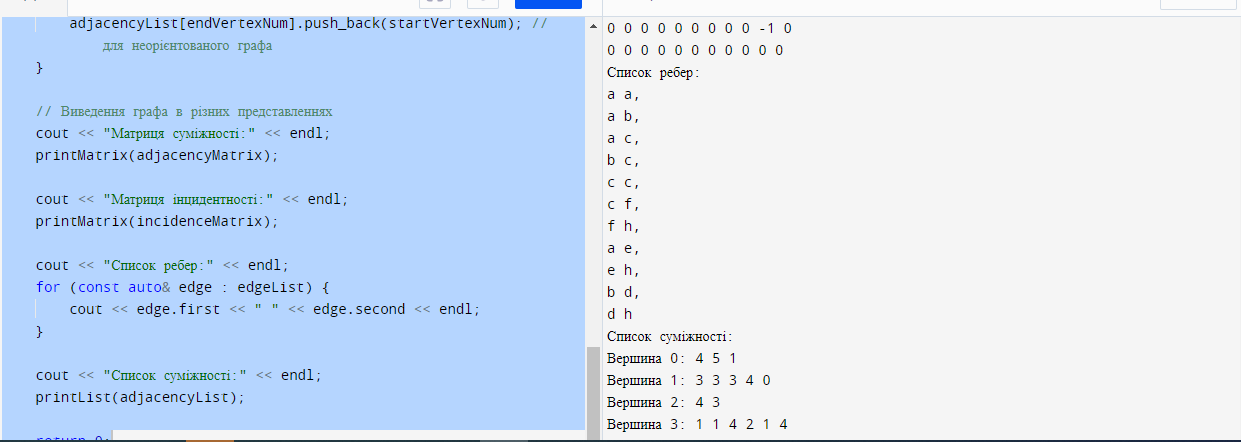
**Орієнтований граф**

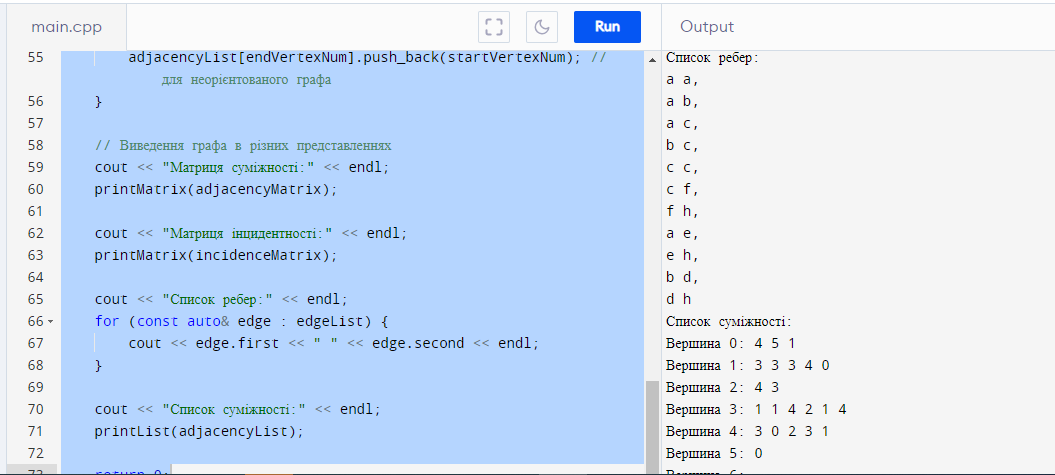
****

**Завдання 1**

**Код той самий**

****

****

****

**Виконання**

**Висновок**

Виконуючи лабораторну роботу, я набула практичних навичок при представленні заданих графів різними способами та можливістю їх комп’ютерної реалізації.

Розмісти ці ребра графа так, v1 v2, v1 v3, v1 v6, v2 v3, v2 v7, v3 v4, v4 v7, v4 v5, v5 v 7, v6 v5, v6 v7 щоб вийшла ось така матриця суміжності

0 1 1 0 0 1 0

1 0 1 0 0 0 1

0 1 0 1 0 0 0

0 0 1 0 1 0 1

0 0 0 1 0 1 1

1 0 0 0 1 0 1

0 1 0 1 1 1 0