МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Кафедра програмної інженерії та інформаційних технологій управління

Звіт з лабораторної роботи № 6
з дисципліни «Основи теорії алгоритмів»

Виконав:
ст. гр. КН-221в
Шулюпов Є.Р.
Перевірила:
доцент каф. ПІТУ
Солонська С.В

Харків
2022

ТЕМА: ФУНДАМЕНТАЛЬНІ АЛГОРИТМИ НА ГРАФАХ І ДЕРЕВАХ

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Розробити програму, яка читає з клавіатури числа N, M (1 < N, M < 256) — кількість вершин та ребер графу; послідовність M пар цілих чисел - ребра графу. Програма зберігає граф та виконує над ним алгоритм згідно варіанту.

Варіанти представлення графів.

1 Матриця суміжності.

2 Список суміжності.

Варіанти алгоритмів.

1 Пошук у ширину. На екран потрібно вивести вершини у порядку обходу. Для кожної вказати час прибуття та предка у дереві обходу.

2 Пошук у глибину. На екран потрібно вивести вершини у порядку обходу. Для кожної вказати час початку розгляду, кінця розгляду та предка у дереві обходу.

3 Топологічне сортування. На екран потрібно вивести ті ж дані, що і для пошуку в глибину, а також результат сортування.

4 Визначити, чи є заданий граф деревом або лісом.

5 Побудувати остовне дерево алгоритмом Прима.

6 Побудувати остовне дерево алгоритмом Крускала.

МЕТА РОБОТИ

Ознайомлення з фундаментальними алгоритмами пошуку на графах і деревах.

1 ОСНОВНІ ТЕОРЕТИЧНІ ПОЛОЖЕННЯ

Граф – це сукупність непустої множини вершин і множини зв’язків між вершинами(ребер). Графи можуть бути орієнтованими та неорієнтованими [1].

Орієнтований граф визначається як пара скінчених множин вершин та множину впорядкованих пар різних вершин, що називаються дугами чи орієнтованими ребрами.

В неорієнтованому графі множина ребер складається із невпорядкованих пар вершин. Позначають неорієнтоване ребро як , при цьому для неорієнтованого графа і позначають одне і те саме ребро [2].

Ребра називаються суміжними, якщо вони відрізняються та мають спільну граничну вершину [2].

Вершина називається ізольованою, якщо немає ребер, для яких вона є граничною. На рисунку 1 вершина – ізольована [1].

Якщо на графі є ребро, говорять, що вершина суміжна з іншою вершиною. Для неорієнтованих графів відношення суміжності являється симетричним, а для орієнтованих графів це не обов’язково [1].

Дерево – це зв’язаний ациклічний граф. Зв’язність означає наявність шляхів між будь-якою парою вершин, ациклічність – відсутність циклів і те, що між парами вершин є лише один шлях [1].

Графи застосовуються для моделювання та описання різних систем і процесів, як наприклад, для транспортних маршрутів, відношення об’єктів, потоків вантажів і повідомлень. Для розв’язання різноманітних задач на графах існує велика кількість алгоритмів, це і пошук вершин на графі, і знаходження найкоротших шляхів, побудова мінімального кістякового дерева і т.д [1].

Для подання графів в лабораторній роботі необхідно використовувати матрицю суміжності чи список суміжності в залежності від завдання [2].

Основна ідея алгоритму пошуку в ширину на графі – перелічити всі досяжні вершини із вказаної початкової в порядку зростання відстані [1].

Відстанню між вершинами вважається довжина мінімального шляху (кількість ребер). В процесі виконання пошуку в ширину вершини графа розфарбовуються в різні кольори та отримують мітку відстані. Спочатку всі вершини білі. Коли вершина вперше виявлена, вона зафарбовується в сірий колір. Вершина стає чорною, коли виявлені всі суміжні до неї вершини [1].

Алгоритм пошуку в глибину дозволяє побудувати обхід графа, в якому відвідуються всі вершини, доступні з початкової. В процесі виконання пошуку в глибину, вершини графа також зафарбовуються в різні кольори, що свідчать про їх стан. Після попадання в тупик повертаємося назад вздовж пройденого шляху, поки не виявимо вершину, у якої ще є невиявлені суміжні вершини, а потім рухаємось в цьому новому напрямку. Якщо вершина повністю оброблена (тобто всі суміжні з нею вершини сірі), вона зафарбовується в чорний колір і ставиться її друга мітка часу. Процес завершується, коли ми повернулись в початкову точку, а всі суміжні з нею вершини вже виявлені [2].

2 ОПИСАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Програма реалізована в одному файлі source.cpp

```
#include <iostream>
#include <list>
using namespace std;


class DFSGraph
{
    int V;
    list<int>* adjList;
    void DFS_util(int v, bool visited[]);
    int time = 1;
    bool visited2[256];

public:
    // class Constructor
    DFSGraph(int V)
    {
        this->V = V;
        adjList = new list<int>[V];
    }


    void addEdge(int v, int w) {
        adjList[v].push_back(w);
    }
    void DFS();
    void printGraph();
};
void DFSGraph::DFS_util(int v, bool visited[])
{

    visited[v] = true;
    cout << "Вершина " << v << "\t Мітка часу 1 -  ";
    cout << time << endl;
```

```
    time++;
    list<int>::iterator i;
    for (i = adjList[v].begin(); i != adjList[v].end(); i++)
    {

        if (!visited[*i])
            DFS_util(*i, visited);
        if (visited[*i])
        {
            cout << "Вершина " << *i << "\t Мітка часу 2 - " << time << endl;
            time++;
            visited2[*i] = true;
        }

    }
}


void DFSGraph::DFS()
{

    bool* visited = new bool[V];
    for (int i = 0; i < V; i++)
    {
        visited[i] = false;
        visited2[i] = false;
    }


    for (int i = 0; i < V; i++)
    {
        if (visited[i] == false)
            DFS_util(i, visited);
        if (visited[i] == true && visited2[i] != true)
        {
            cout << "Вершина " << i << "\t Мітка часу 2 - " << time << endl;
            time++;

        }
    }
}

void DFSGraph::printGraph() {
    for (int d = 0; d < V; ++d) {
        cout << "\n Вершина " << d << ":";
        for (auto x : adjList[d])
            cout << "-> " << x;
        printf("\n");
    }
}

int main()
{
    setlocale(LC_ALL, "Ukr");
    int n, m, v, u;
    cout << "Введіть значення n - кількість вершин, та значення m - кількість ребер відповідно:" <<
endl;
    cin >> n >> m;
    DFSGraph g(n);
    for (int i = 0; i < m; i++)
    {
        cout << "Введіть дві вершини графа. Перша вказує на другу:" << endl;
        cin >> v >> u;
        g.addEdge(v, u);
```

```
    }
    cout << "Список суміжності:" << endl;
    g.printGraph();
    cout << "Пошук в глибину з першої точки" << endl;

    g.DFS();
    return 0;
}
```

Результат роботи представлений на рисунку 2.1.

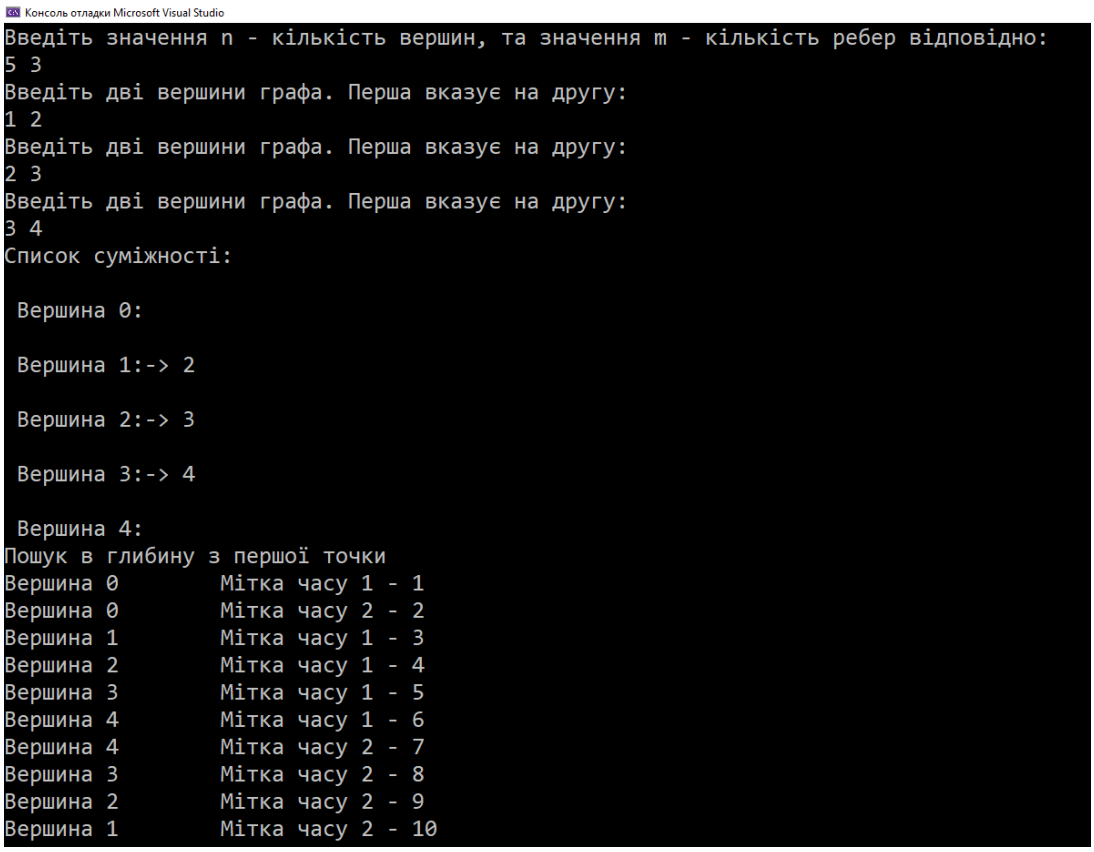

Рисунок 2.1 – Результат

ВИСНОВКИ

Під час цієї лабораторної роботи була розроблена програма, яка виконує всі функції відповідно мого варіанту завдання, а саме записує всі дані до списку суміжності а потім виконує пошук в глибину. Дана програма повністю розкриває всю суть роботи

із графами та правилами пошуку та встановлювання міток часу для тих чи інших елементів даної структури даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Методичні вказівки до виконання лабораторних робіт з курсу "Алгоритми і структури даних": для студентів, які навчаються за спец. 121 "Інженерія програмного забезпечення" [Електронний ресурс] / уклад. Н. К. Стратієнко, І. О. Бородіна ; Харківський політехнічний інститут, національний технічний університет університет – Електрон. текстові дані. – Харків, 2017. – 36 с. – Режим доступу: http://repository.kpi.kharkov.ua/handle/KhPI-Press/26426.

2. Алгоритми і структури даних: практикум: навч. посіб./ Н.К. Стратієнко, М.Д. Годлевський, І.О. Бородіна.- Харьков: НТУ"ХПИ", 2017. - 224 с.