|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №2.4  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А. В. Поляков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2024 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать алгоритм для работы с ориентированным взвешенным графом.

Входные данные: текстовый файл, в первой строке – целое число - количество вершин графа n (5 до 100). Каждая из последующих строк содержит данные внутреннего представления графа (двумерный массив).

Реализовать класс для работы с ориентированным взвешенным графом, представленным заданным видом: матрица смежности

Методы:

1. Конструктор

2. Деструктор

3. Добавление вершины

4. Удаление вершины

5. Добавление ребра/дуги

6. Удаление ребра/дуги

7. Печать (внутреннего представления)

8. Топологическая сортировка

9. Алгоритм Дейкстры

**2 Описание класса**

**Поля**:

int\*\* Matrix - матрица смежности

int V - количество вершин в графе

**Методы**:

Конструкторы:

Graph(const string& filename) - конструктор графа, загружающий данные из файла

Деструктор:

~Graph() - деструктор графа, освобождающий память

Методы для работы с графом:

void addVertex() - добавление вершины в граф

void removeVertex(int v) - удаление вершины из графа

void addEdge(int u, int v, int weight) - добавление дуги с весом

void removeEdge(int u, int v) - удаление дуги

void print() - вывод графа

bool detectCycle() - проверка графа на наличие цикла

void topologicalSort() - топологическая сортировка графа

void deikstra(int element) - алгоритм Дейкстры для поиска кратчайшего пути

Вспомогательные методы:

bool detectCycleUtil(int v, vector<bool>& visited, vector<bool>& recStack) - метод для поиска цикла

void topologicalSortUtil(int v, vector<bool>& visited, vector<int>& Stack) - метод для топологической сортировки

int minDistance(const vector<int>& dist, const vector<bool>& sptSet) - нахождение вершины с минимальным расстоянием

void printSolution(const vector<int>& dist, int element) - вывод результата алгоритма Дейкстры

Список формальных параметров:

int V - количество вершин в графе

int\*\* Matrix - матрица смежности графа

const string& filename - имя файла для загрузки графа

int v - индекс вершины

int u - начальная вершина дуги

int v - конечная вершина дуги

int weight - вес дуги

vector<bool>& visited - вектор посещенных вершин

vector<bool>& recStack - стек рекурсивных вызовов

vector<int>& Stack - стек для топологической сортировки

const vector<int>& dist - вектор расстояний

const vector<bool>& sptSet - множество вершин, включенных в кратчайший путь

Входные данные:

const string& filename - имя файла для загрузки графа

int v - индекс вершины

int u - начальная вершина дуги

int v - конечная вершина дуги

int weight - вес дуги

int element - элемент для алгоритма Дейкстры

Выходные данные:

void addVertex() - добавление вершины в граф

void removeVertex(int v) - удаление вершины из графа

void addEdge(int u, int v, int weight) - добавление дуги с весом

void removeEdge(int u, int v) - удаление дуги

void print() - вывод графа

bool detectCycle() - проверка графа на наличие цикла

void topologicalSort() - топологическая сортировка графа

void deikstra(int element) - алгоритм Дейкстры для поиска кратчайшего пути

**3 Текст программы**

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

#include <limits>

using namespace std;

class Graph {

private:

    int\*\* Matrix;

public:

    int V;

    Graph(const string& filename) {

        ifstream file(filename);

        if (file.is\_open()) {

            file >> V;

            Matrix = new int\*[V];

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

                Matrix[i] = new int[V];

                for (int j = 0; j < V; ++j) {

                    file >> Matrix[i][j];

                }

            }

            file.close();

        } else {

            cout << "Файл не найден\n";

            V = 0;

            Matrix = nullptr;

        }

    }

    ~Graph() {

        for (int i = 0; i < V; ++i) {

            delete[] Matrix[i];

        }

        delete[] Matrix;

    }

    void addVertex() {

        int\*\* newMatrix = new int\*[V + 1];

        for (int i = 0; i < V + 1; ++i) {

            newMatrix[i] = new int[V + 1];

            for (int j = 0; j < V + 1; ++j) {

                if (i < V && j < V) {

                    newMatrix[i][j] = Matrix[i][j];

                } else {

                    newMatrix[i][j] = 0;

                }

            }

        }

        for (int i = 0; i < V; ++i) {

            delete[] Matrix[i];

        }

        delete[] Matrix;

        Matrix = newMatrix;

        ++V;

    }

    void removeVertex(int v) {

        v--;

        if (v >= V) return;

        int\*\* newMatrix = new int\*[V - 1];

        for (int i = 0, ni = 0; i < V; ++i) {

            if (i == v) continue;

            newMatrix[ni] = new int[V - 1];

            for (int j = 0, nj = 0; j < V; ++j) {

                if (j == v) continue;

                newMatrix[ni][nj] = Matrix[i][j];

                nj++;

            }

            ni++;

        }

        for (int i = 0; i < V; ++i) {

            delete[] Matrix[i];

        }

        delete[] Matrix;

        Matrix = newMatrix;

        --V;

    }

    void addEdge(int u, int v, int weight) {

        u--;

        v--;

        if (u < V && v < V) {

            Matrix[u][v] = weight;

        }

    }

    void removeEdge(int u, int v) {

        u--;

        v--;

        if (u < V && v < V) {

            Matrix[u][v] = 0;

        }

    }

    void print() {

        for (int i = 0; i < V; ++i) {

            for (int j = 0; j < V; ++j) {

                cout << Matrix[i][j] << " ";

            }

            cout << endl;

        }

    }

    bool detectCycleUtil(int v, vector<bool>& visited, vector<bool>& recStack) {

        if (!visited[v]) {

            visited[v] = true;

            recStack[v] = true;

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

                if (Matrix[v][i]) {

                    if (!visited[i] && detectCycleUtil(i, visited, recStack)) {

                        return true;

                    } else if (recStack[i]) {

                        return true;

                    }

                }

            }

        }

        recStack[v] = false;

        return false;

    }

    // Обход в глубину

    bool detectCycle() {

        vector<bool> visited(V, false);

        vector<bool> recStack(V, false);

        for (int i = 0; i < V; ++i) {

            if (detectCycleUtil(i, visited, recStack)) {

                return true;

            }

        }

        return false;

    }

    void topologicalSortUtil(int v, vector<bool>& visited, vector<int>& Stack) {

        visited[v] = true;

        for (int i = 0; i < V; ++i) {

            if (Matrix[v][i] && !visited[i]) {

                topologicalSortUtil(i, visited, Stack);

            }

        }

        Stack.push\_back(v);

    }

    void topologicalSort() {

        if (detectCycle()) {

            cout << "Граф содержит цикл.\n";

            return;

        }

        vector<int> Stack;

        vector<bool> visited(V, false);

        for (int i = 0; i < V; ++i) {

            if (!visited[i]) {

                topologicalSortUtil(i, visited, Stack);

            }

        }

        reverse(Stack.begin(), Stack.end());

        for (const auto& v : Stack) {

            cout << v + 1 << " ";

        }

        cout << endl;

    }

    void deikstra(int element) {

        element -= 1;

        vector<int> D(V, numeric\_limits<int>::max());

        vector<bool> visited(V, false);

        D[element] = 0;

        for (int i = 0; i < V; ++i) {

            for (int j = 0; j < V; ++j) {

                if (Matrix[i][j] < 0) {

                    cout << "Граф содержит отрицательный вес дуги: " << i + 1 << " -> " << j + 1 << " = " << Matrix[i][j] << endl;

                    return;

                }

            }

        }

        for (int count = 0; count < V - 1; ++count) {

            int u = minDistance(D, visited);

            visited[u] = true;

            for (int v = 0; v < V; ++v) {

                if (!visited[v] && Matrix[u][v] && D[u] != numeric\_limits<int>::max() && D[u] + Matrix[u][v] < D[v]) {

                    D[v] = D[u] + Matrix[u][v];

                }

            }

        }

        printSolution(D, element);

    }

    int minDistance(const vector<int>& dist, const vector<bool>& sptSet) {

        int min = numeric\_limits<int>::max(), min\_index = -1;

        for (int v = 0; v < V; ++v) {

            if (!sptSet[v] && dist[v] <= min) {

                min = dist[v], min\_index = v;

            }

        }

        return min\_index;

    }

    void printSolution(const vector<int>& dist, int element) {

        for (int i = 0; i < V; ++i) {

            if (dist[i] != numeric\_limits<int>::max())

                cout << element + 1 << " -> " << i + 1 << " = " << dist[i] << endl;

            else

                cout << element + 1 << " -> " << i + 1 << " = маршрута нет\n";

        }

    }

};

int main() {

    Graph g("graph.txt");

    if (g.V == 0) {

        cout << "В графе нет вершин\n";

        return 1;

    }

    cout << "Граф:\n";

    g.print();

    cout << "\nТопологическая сортировка:\n";

    g.topologicalSort();

    cout << "\nАлгоритм Дейкстры:\n";

    g.deikstra(1);

    g.addVertex();

    cout << "\nГраф после добавления точки:\n";

    g.print();

    g.addEdge(6, 2, 10);

    cout << "\nГраф после добавления дуги (6 -> 2 с весом 10):\n";

    g.print();

    g.removeEdge(1, 4);

    cout << "\nГраф после удаления дуги (1 -> 4):\n";

    g.print();

    g.removeVertex(1);

    cout << "\nГраф после удаления точки 1:\n";

    g.print();

    return 0;

}

**4. Тесты**

**Тесты корректности алгоритма:**

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление элемента в пустой граф | 6 2 10  Граф: 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0 | 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 10 0 0 0 0 |
| Добавление дуги отрицательного веса в граф | 6 2 -1  Граф: 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0 | 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 -1 0 0 0 0 |
| Удаление вершины из графа | 1  Граф: 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 11 0 0 0 0 | 0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  11 0 0 0 0 |
| Удаление дуги | 6 2  Граф: 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 11 0 0 0 0 | 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0 |
| Топологическая сортировка | Граф: 0 2 0 8 0  0 0 0 0 0  0 2 0 0 0  0 0 7 0 0  4 0 0 0 0 | 5 1 4 3 2 |
| Поиск кратчайшего пути Дейкстры | 1  Граф:  0 2 0 8 0  0 0 0 0 0  0 2 0 0 0  0 0 7 0 0  4 0 0 0 0 | Алгоритм Дейкстры:  1 -> 1 = 0  1 -> 2 = 2  1 -> 3 = 15  1 -> 4 = 8  1 -> 5 = маршрута нет |