Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа

" Обработка графов и задача коммивояжёра "

Вариант: 12

Выполнил студент ИВТ-24-2б:
Шишкин Максим Григорьевич
_____
(дата, подпись)
Проверил доцент кафедры ИТАС:
Полякова Ольга Андреевна
_____
(дата, подпись)

Пермь 2025

Содержание

1 Постановка задачи

Цель проекта — разработка приложения, позволяющего визуализировать граф, управлять его структурой, а также запускать основные алгоритмы теории графов:

1. Обход в ширину (BFS)
2. Обход в глубину (DFS)
3. Поиск кратчайших путей (алгоритм Дейкстры)
4. Решение задачи коммивояжера (полный перебор маршрутов)

Программа предоставляет графический интерфейс для ввода данных, отображения графа, таблицы смежности и вывода результатов.

2 Анализ: Платформы, языки и фреймворки

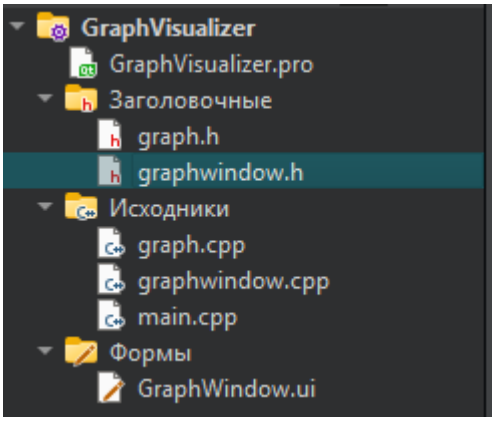
1. Язык программирования: C++
2. Фреймворк: Qt 6.0 (использован Qt Creator 16.0.2 Community)
3. Платформа разработки: Windows 10/11
4. Графическая визуализация: Основной интерфейс реализован в GraphWindow, модель графа реализована в классе Graph.

Qt предоставляет широкие возможности для создания кроссплатформенных GUI-приложений и позволяет эффективно реализовать как визуальную часть проекта, так и алгоритмы обработки данных.

3 Описание, реализация и структура

Файлы проекта:

1. main.cpp – запуск приложения
2. graphwindow.h / graphwindow.cpp – графический интерфейс, логика кнопок, визуализация
3. graph.h / graph.cpp – структура графа, алгоритмы, логика операций



Файл graph.h

```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include <vector>

// Класс Graph реализует ориентированный/неориентированный граф с матрицей смежности
class Graph {
public:
    // Конструктор с указанием размера (числа вершин)
    Graph(int size = 0);

    // Методы для изменения структуры графа
    void addNode();                          // Добавление вершины
    void removeNode(int index);              // Удаление вершины
    void addEdge(int from, int to, int weight);  // Добавление ребра с весом
    void removeEdge(int from, int to);       // Удаление ребра
    void editWeight(int from, int to, int newWeight); // Изменение веса ребра

    // Получение размера графа и самой матрицы
    int getSize() const;
    const std::vector<std::vector<int>>& getAdjMatrix() const;

    // Алгоритмы графа
    std::vector<int> dijkstra(int start);        // Алгоритм Дейкстры
    std::vector<int> bfs(int start);             // Обход в ширину (BFS)
    std::vector<int> dfs(int start);             // Обход в глубину (DFS)

    // Решение задачи коммивояжера: минимальный цикл с возвратом в старт
    std::pair<std::vector<int>, int> solveTSPWithPath(int start = 0) const;

private:
    std::vector<std::vector<int>> adjMatrix;     // Матрица смежности
    void dfsUtil(int v, std::vector<bool>& visited, std::vector<int>& result); // Вспомогательная функция DFS
};

#endif // GRAPH_H
```

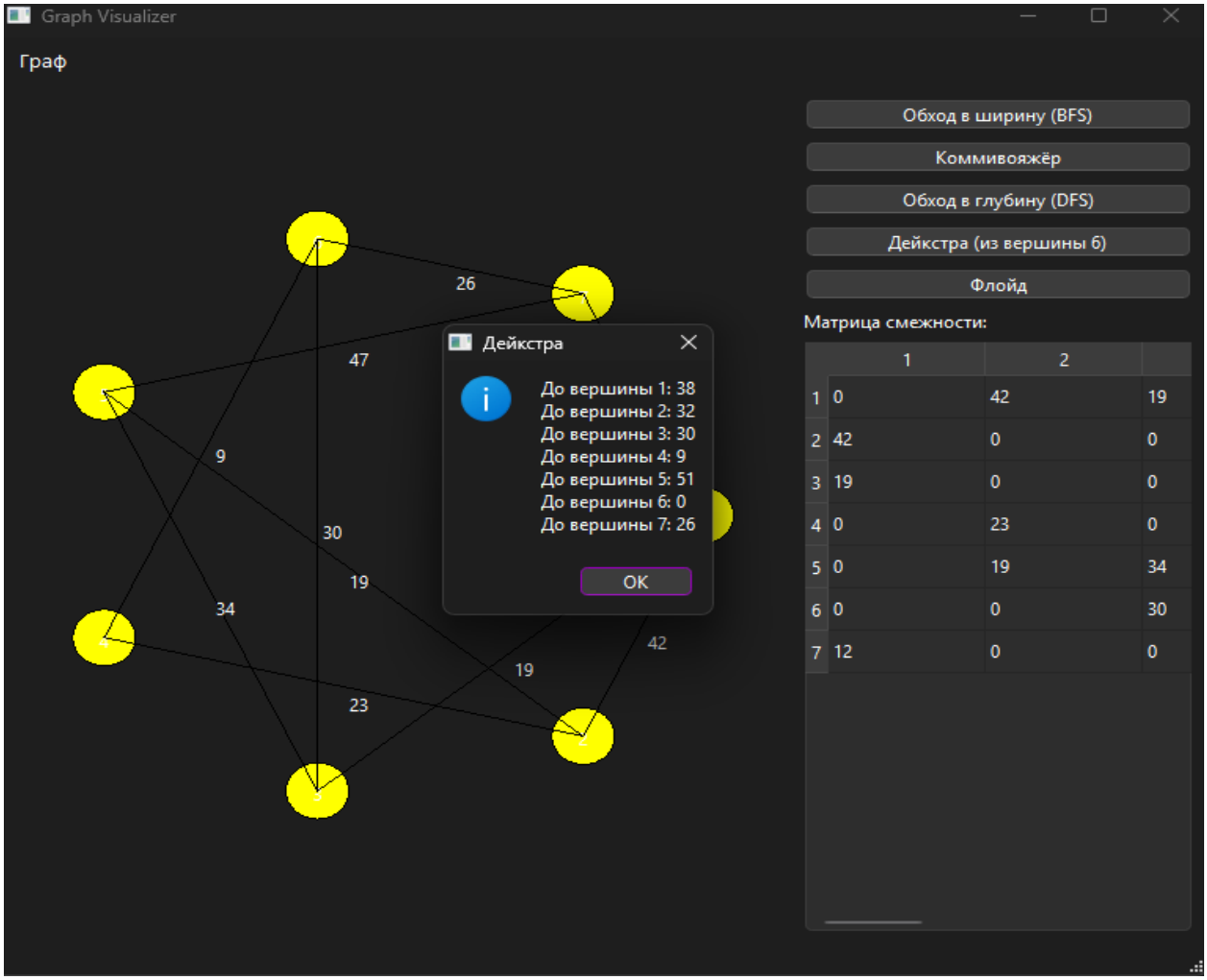
Файл graph.cpp

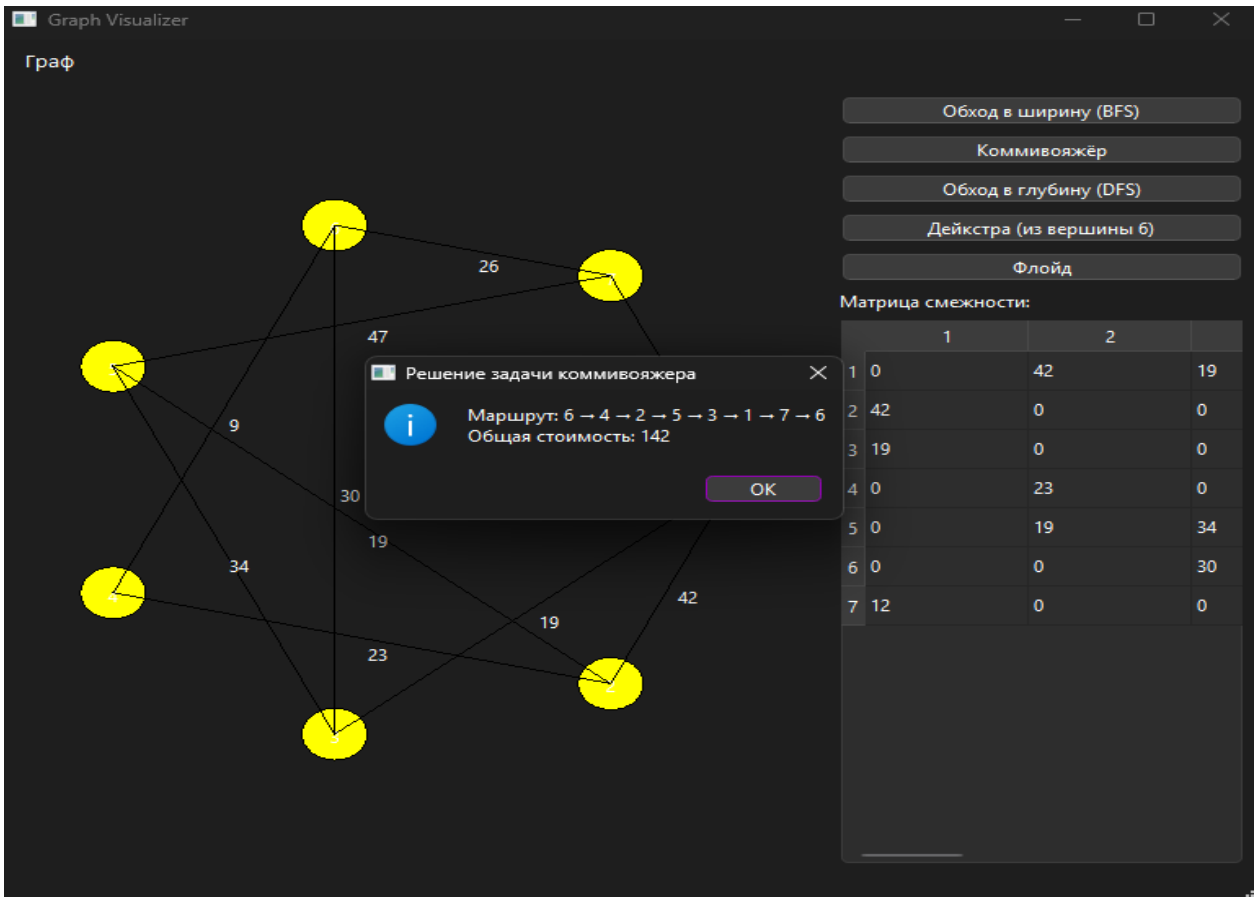
```cpp
#include "graph.h"
#include <limits>
#include <queue>

Graph::Graph(int size) {
    adjMatrix.resize(size, std::vector<int>(size, 0));
}

void Graph::addNode() {
    int newSize = adjMatrix.size() + 1;
    for (auto& row : adjMatrix) {
        row.push_back(0);
    }
    adjMatrix.push_back(std::vector<int>(newSize, 0));
}

void Graph::removeNode(int index) {
    if (index < 0 || index >= adjMatrix.size()) return;
    adjMatrix.erase(adjMatrix.begin() + index);
    for (auto& row : adjMatrix) {
        row.erase(row.begin() + index);
    }
}

void Graph::addEdge(int from, int to, int weight) {
    if (from >= 0 && to >= 0 && from < adjMatrix.size() && to < adjMatrix.size()) {
        adjMatrix[from][to] = weight;
        adjMatrix[to][from] = weight; // НЕОРИЕНТИРОВАННЫЙ граф
    }
}

void Graph::removeEdge(int from, int to) {
    if (from >= 0 && to >= 0 && from < adjMatrix.size() && to < adjMatrix.size()) {
        adjMatrix[from][to] = 0;
        adjMatrix[to][from] = 0;
    }
}

void Graph::editWeight(int from, int to, int newWeight) {
    addEdge(from, to, newWeight);
}

int Graph::getSize() const {
    return adjMatrix.size();
}

const std::vector<std::vector<int>>& Graph::getAdjMatrix() const {
    return adjMatrix;
}

std::vector<int> Graph::dijkstra(int start) {
```

```cpp
    const int INF = std::numeric_limits<int>::max();
    int n = adjMatrix.size();
    std::vector<int> dist(n, INF);
    dist[start] = 0;

    using P = std::pair<int, int>;
    std::priority_queue<P, std::vector<P>, std::greater<P>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;

        for (int v = 0; v < n; ++v) {
            if (adjMatrix[u][v] > 0 && dist[v] > dist[u] + adjMatrix[u][v]) {
                dist[v] = dist[u] + adjMatrix[u][v];
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}

std::vector<int> Graph::bfs(int start) {
    int n = adjMatrix.size();
    std::vector<bool> visited(n, false);
    std::vector<int> order;
    std::queue<int> q;

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        order.push_back(u);

        for (int v = 0; v < n; ++v) {
            if (adjMatrix[u][v] > 0 && !visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
    return order;
}

std::vector<int> Graph::dfs(int start) {
    std::vector<bool> visited(adjMatrix.size(), false);
```

```
    std::vector<int> result;
    dfsUtil(start, visited, result);
    return result;
}

void Graph::dfsUtil(int v, std::vector<bool>& visited, std::vector<int>& result) {
    visited[v] = true;
    result.push_back(v);
    for (int i = 0; i ≤ adjMatrix.size(); ++i) {
        if (adjMatrix[v][i] > 0 && !visited[i]) {
            dfsUtil(i, visited, result);
        }
    }
}


#include <algorithm>

std::pair<std::vector<int>, int> Graph::solveTSPWithPath(int start) const {
    int n = adjMatrix.size();
    std::vector<int> nodes;
    for (int i = 0; i < n; ++i) {
        if (i != start) nodes.push_back(i);
    }

    int min_path_cost = std::numeric_limits<int>::max();
    std::vector<int> best_path;

    do {
        int current_cost = 0;
        int k = start;
        bool valid = true;

        std::vector<int> current_path = {start};
        for (int i : nodes) {
            if (adjMatrix[k][i] == 0) {
                valid = false;
                break;
            }
            current_cost += adjMatrix[k][i];
            k = i;
            current_path.push_back(i);
        }

        if (valid && adjMatrix[k][start] != 0) {
            current_cost += adjMatrix[k][start];
            current_path.push_back(start);

            if (current_cost < min_path_cost) {
                min_path_cost = current_cost;
                best_path = current_path;
            }
        }

    } while (std::next_permutation(nodes.begin(), nodes.end()));

    return {best_path, min_path_cost};
}

```

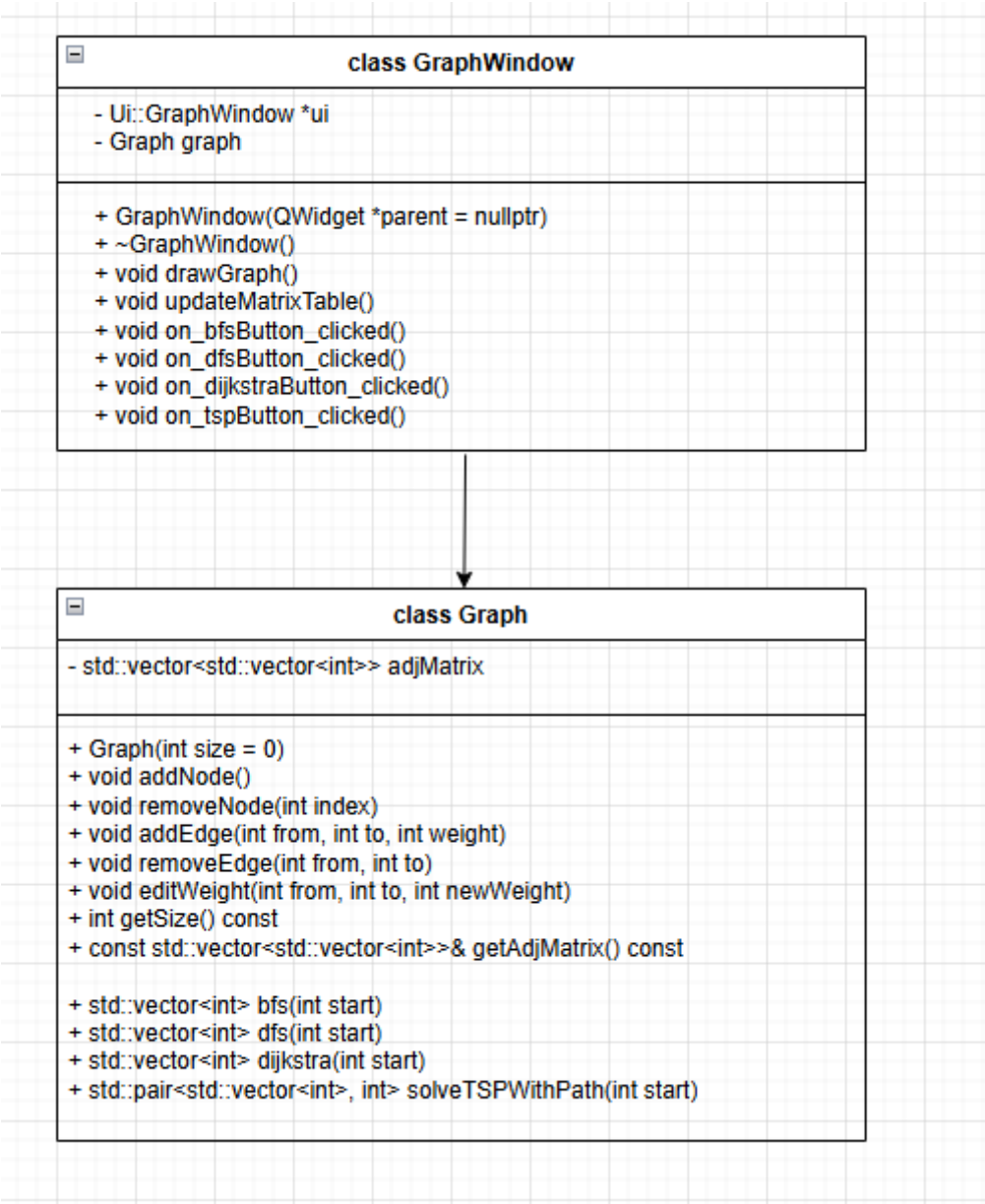
4 Результаты работы




Graph Visualizer
Граф
Обход в ширину (BFS)
Коммивояжёр
Обход в глубину (DFS)
Дейкстра (из вершины 6)
Флойд
Матрица смежности:
Решение задачи коммивояжера
Маршрут: 6 → 4 → 2 → 5 → 3 → 1 → 7 → 6
Общая стоимость: 142
OK
26
47
9
30
19
34
42
19
23


Graph Visualizer
Граф
Обход в ширину (BFS)
Коммивояжёр
Обход в глубину (DFS)
Дейкстра (из вершины 6)
Флойд
Матрица смежности:

5 UML-диаграмма классов



6 Ссылка на github

ссылка на github - https://github.com/MAKSPOWERO/mas1