МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра МОиСИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7.2
«НЕЛИНЕЙНЫЕ СТРУКТУРЫ»

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-10-24 Бикташев И. И.

Практическую работу выполнил «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

Цель работы: Составить программу создания графа и реализовать процедуру для работы с графом, определенную индивидуальным вариантом задания.

Реализуемый алгоритм: Нахождение кратчайшего пути методом Дейкстра.

Описание решения: Алгоритм Дейкстры позволяет найти кратчайший путь от одной верны взвешенного ориентированного графа до другой. Каждой вершине из множества вершин V сопоставим метку — минимальное известное расстояние от этой вершины до стартовой вершины a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Решение: Для программной реализации графа будем использовать структуру Edge и класс Graph. Класс Edge представляет собой ребро графа с определённым весом и направлением (конечной вершиной) (листинг 1)

Листинг 1 - Структура Edge

```
struct Edge {
    int destination;
    int weight;

    Edge(int dest, int w) : destination(dest), weight(w) {}
};
```

Класс Graph содержит количество вершин, и их соединения в виде вектора векторов рёбер (листинг 2)

Листинг 2 - Класс Graph

```
class Graph {
    int vertices;
    vector<vector<Edge>> edges;
public:
    // Конструктор
    Graph(int v) : vertices(v), edges(v) {}

    // Добавление ребра в граф
    void addEdge(int source, int destination, int weight) {
        edges[source].push_back(Edge(destination, weight));
    }

    // Вывод графа
    void printGraph() {
        cout << "Структура графа:" << endl;
        for (int i = 0; i < vertices; i++) {
            cout << "Вершина " << i << " соединена с: ";
            for (auto& edge : edges[i]) {
```

```
                cout << "(" << edge.destination << ", вес: " << edge.weight <<
") ";
            }
            cout << endl;
        }
    }

    // Получение количества вершин
    int getVerticesCount() {
        return vertices;
    }

    // Алгоритм Дейкстры для нахождения кратчайшего пути
    vector<int> Graph::dijkstra(int startVertex, int endVertex);
};
```

Сам алгоритм Дейкстры реализуются с помощью очереди приоритетов в которую мы подаем пары из значений «вес ребра» и «конечная вершина». Приоритет будет выше у той пары, вес чьего ребра наименьший. Начинаем проходится по вершинам графа начиная с начальной вершины пока не встретим конечную вершину. Каждую итерацию записываем в отдельный массив с какой вершины до текущей идёт ребро, являющееся частью кратчайшего маршрута, что позволяет нам после выполнения самого алгоритма восстановить путь от конечной вершины до начальной. (листинг 3)

Листинг 3 - Алгоритм Дейкстры

```
vector<int> Graph::dijkstra(int startVertex, int endVertex) {
    vector<int> weights(vertices, INT_MAX), previous(vertices, -1);
    vector<bool> visited(vertices, false);
    // Приоритетная очередь для выбора вершины с минимальным расстоянием
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> pq;
    weights[startVertex] = 0;
    pq.push({0, startVertex});
    while (!pq.empty()) {
        // Извлекаем вершину с минимальным расстоянием
        int currentVertex = pq.top().second;
        pq.pop();
        // Если вершина уже посещена, пропускаем
        if (visited[currentVertex])
            continue;
        visited[currentVertex] = true;
        // Если достигли конечной вершины, выходим
        if (currentVertex == endVertex)
            break;
        // Обновляем расстояния до соседних вершин
        for (auto& edge : edges[currentVertex]) {
            int neighbor = edge.destination;
            int newDistance = weights[currentVertex] + edge.weight;

            if (newDistance < weights[neighbor]) {
                weights[neighbor] = newDistance;
                previous[neighbor] = currentVertex;
```

```
                pq.push({newDistance, neighbor});
            }
        }
    }
    // Восстанавливаем путь от конечной вершины к начальной
    vector<int> path;
    if (weights[endVertex] == INT_MAX) {
        // Путь не существует
        return path;
    }
    for (int vertex = endVertex; vertex != -1; vertex = previous[vertex]) {
        path.push_back(vertex);
    }
    return path;
}
```

Для демонстрации работы алгоритма были реализованы два режима работы с программой: демонстрационный и интерактивный. Демонстрационный режим демонстрирует работу алгоритма на графе из варианта задания. Интерактивный позволяет создать собственный граф и найти кратчайший путь от одной вершины данного графа к другой. На следующий рисунках предоставлены результаты работы программы.

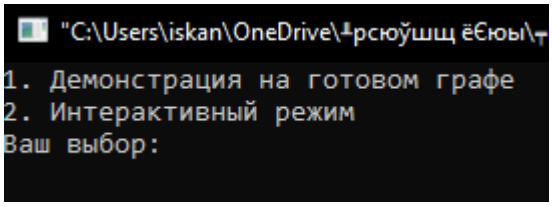

Рисунок 1– Экран выбора режима

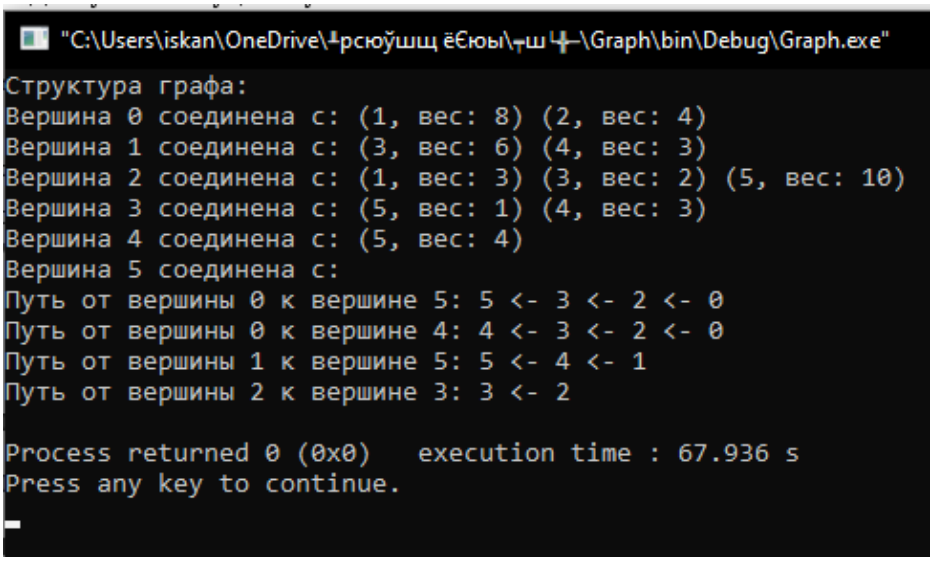

Рисунок 2 – Демонстрационный режим

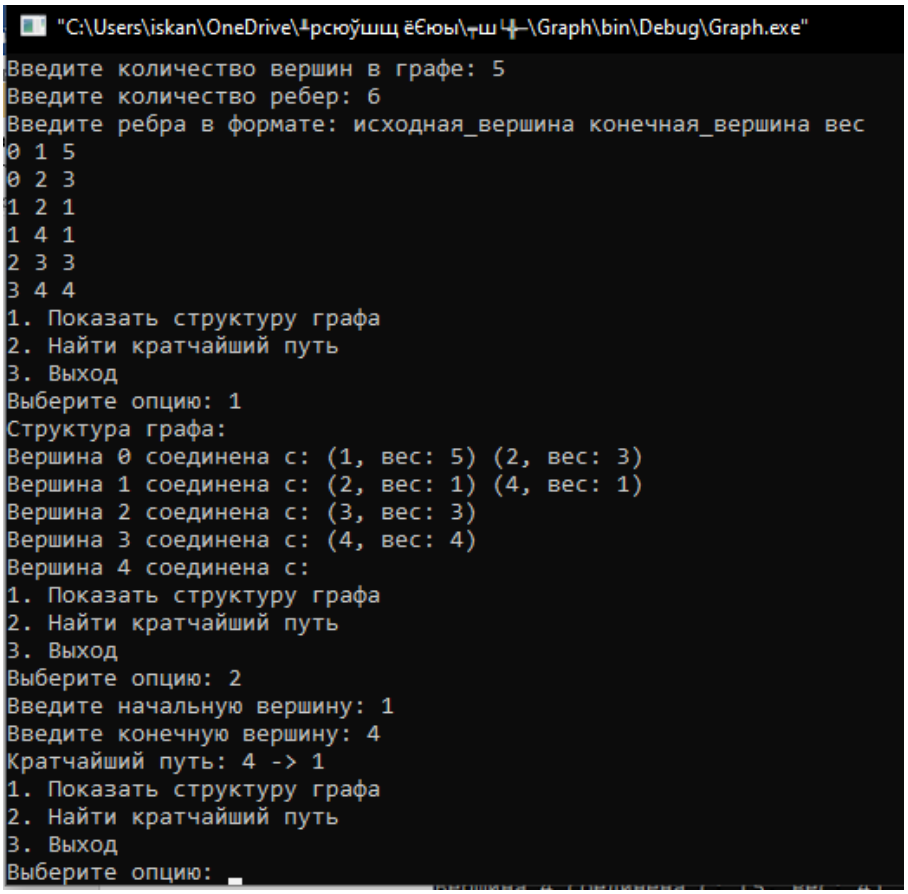

Рисунок 3 - Интерактивный режим

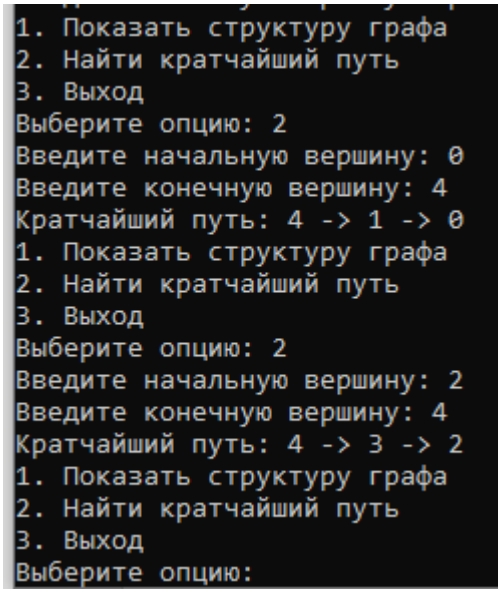

Рисунок 4 - Кратчайшие пути между вершинами

Вывод

Поставленные задачи выполнены: изучены и реализованы алгоритмы по работе с взвешенным ориентированным графом. Реализован алгоритм поиска кратчайшего пути в графе методом Дейксра.

Литература

Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 31.09.2025)

Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 31.09.2025)