**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 6 на тему:   
**«Разработка программ моделирования работы стека и очереди.»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Кандиков М.В.

«11» Ноября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/М.В. Кандиков /

Принял: к.ф.-м.н., доцент, И.А. Моисеев

«12» Ноября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание**

**1. Титульный лист 1**

**2. Содержание 2**

**3. Основная часть 3**

3.1. Цель работы 3

3.2. Теоретическая часть 3

3.2. Практическая часть 4

3.3. Выводы 7

**4. Приложение 8**

**Основная часть**

**Цель работы:**

Целью данной работы является разработка и реализация алгоритма для нахождения кратчайшего маршрута машины, развозящей почту, от почтамта до нескольких отделений связи и обратно. Задача заключается в минимизации общего пути, который проходит машина, при условии, что она должна посетить каждое отделение связи один раз и вернуться в исходную точку (почтамт). Решение этой задачи позволяет изучить и применить алгоритмы поиска кратчайшего пути (алгоритм Дейкстры) и оптимального маршрута (задача коммивояжёра).

**Теоретическая часть:**  
Для решения задачи был использован граф, где:

* Вершины представляют почтамт и отделения связи.
* Рёбра (ребро между двумя вершинами) определяют расстояния между почтамтом и каждым отделением, а также расстояния между отделениями связи.

Задача состоит из двух частей:

1. Поиск кратчайших расстояний между всеми вершинами. Для этого используется алгоритм Дейкстры, который находит кратчайший путь от одной вершины графа до всех остальных. Он работает за время O(V2), что является эффективным подходом для относительно небольшого числа вершин.
2. Решение задачи коммивояжёра. Здесь требуется найти маршрут с минимальной общей длиной, проходящий через все вершины ровно один раз и возвращающийся в начальную точку. Для этого используется метод динамического программирования, который запоминает минимальные пути для всех комбинаций посещённых вершин.

**Практическая часть:**

Ниже приведены ключевые фрагменты кода, реализующие основные этапы решения.

**1. Построение графа и добавление рёбер**

Этот фрагмент кода показывает создание графа с заданными расстояниями между вершинами, где “addEdge” добавляет неориентированное ребро с определённым весом между вершинами.

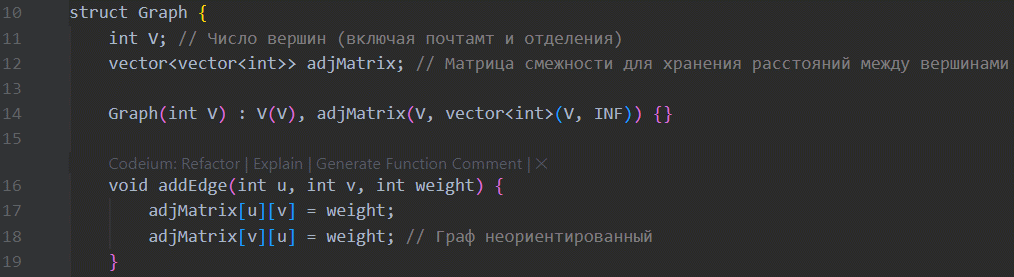


Рисунок 1. Структура Graph

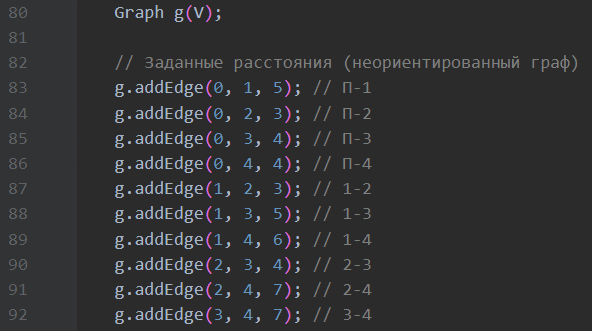


Рисунок 2. Построение графа и добавление рёбер.

**2. Алгоритм Дейкстры для нахождения кратчайших путей**

Функция dijkstra находит кратчайшие расстояния от одной вершины до всех остальных с помощью приоритетного поиска минимального расстояния. Этот этап подготавливает данные для задачи коммивояжёра.

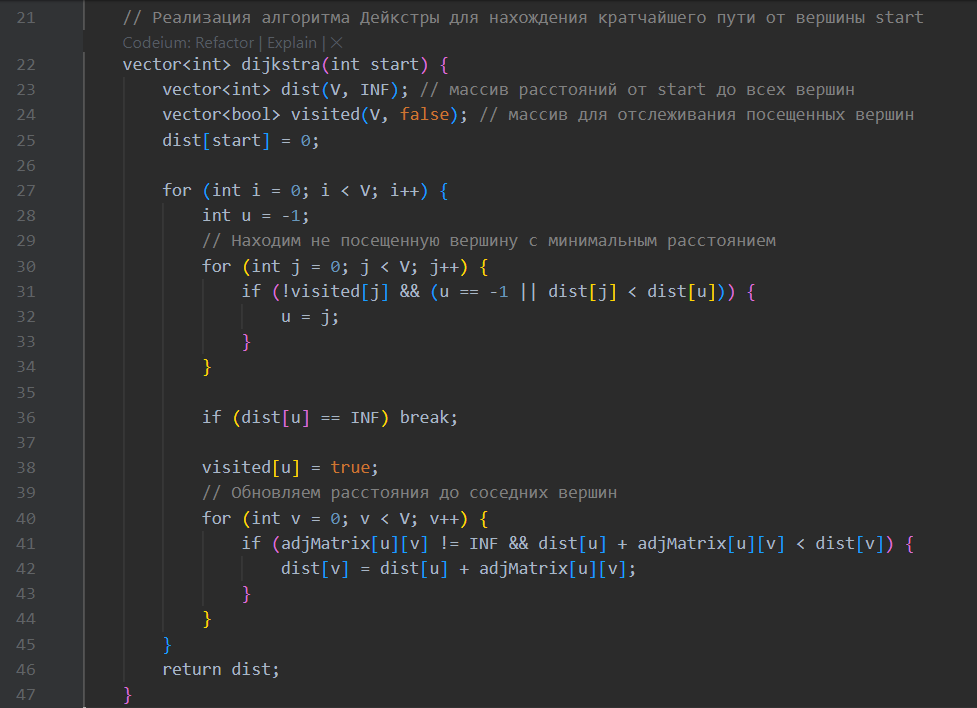


Рисунок 3. Функция алгоритма Дийкстры.

**3. Решение задачи коммивояжёра**

Этот рекурсивный метод travelingSalesman находит минимальный путь, проходящий через все вершины. Он использует динамическое программирование для хранения промежуточных значений, чтобы избежать избыточных вычислений.

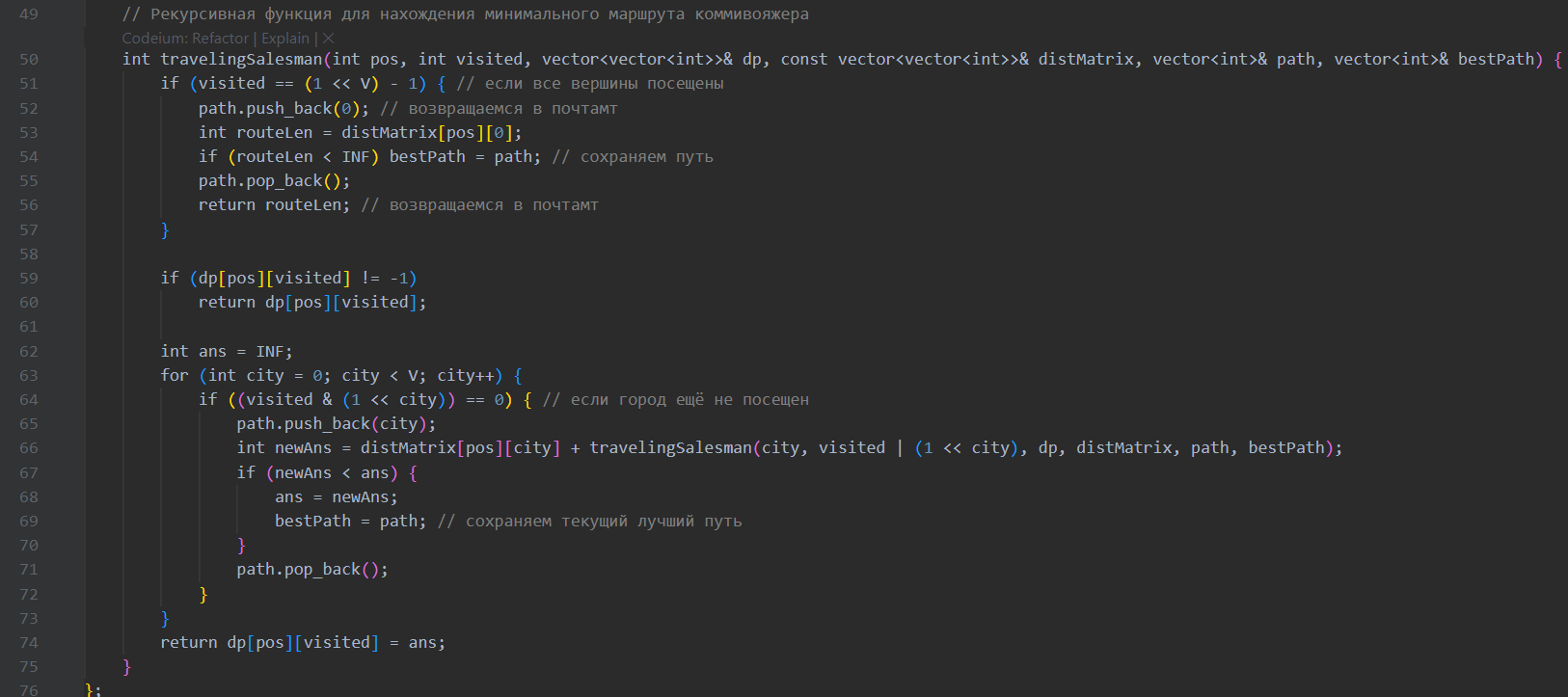


Рисунок 4. Рекурсивный метод travelingSalesman.

**4. Вывод результатов**

Этот блок кода выводит длину кратчайшего маршрута и его последовательность, начиная и заканчивая на почтамте.



Рисунок 5. Вывод результатов.

**5. Полученные результаты**

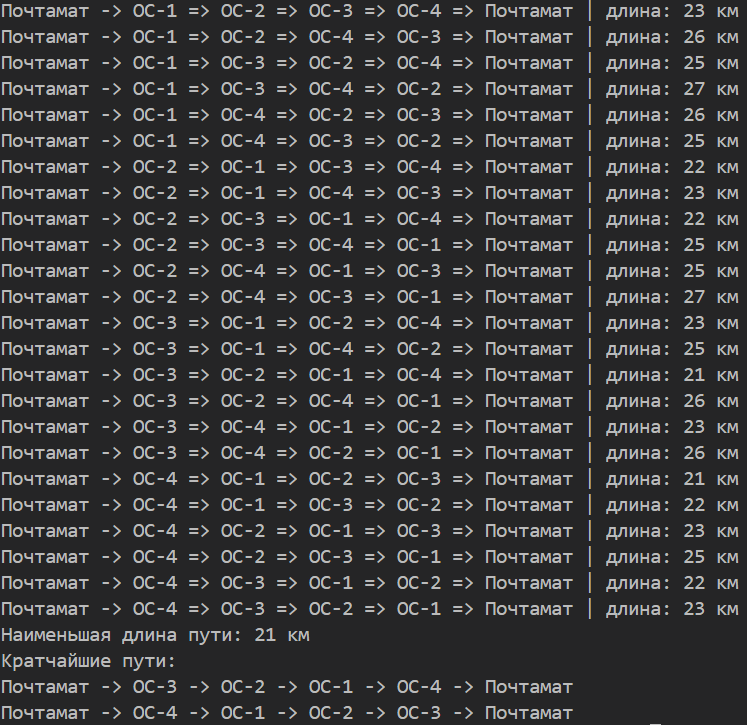


Рисунок 6. Результат выполненной программы.

**6. Полученные результаты**

Программа вычислила оптимальный маршрут развоза почты, минимизирующий общее пройденное расстояние. Кратчайший маршрут составил 21 км, и машина будет следовать по следующей последовательности точек:

1. Почтамат
2. ОС-3 (отделение связи №3)
3. ОС-2 (отделение связи №2)
4. ОС-1 (отделение связи №1)
5. ОС-4 (отделение связи №4)
6. Возвращение на Почтамт

Таким образом, маршрут позволяет машине посетить каждое отделение связи ровно один раз и вернуться в начальную точку, при этом общая длина пути составляет 21 км. Этот результат достигается благодаря использованию алгоритма Дейкстры для расчёта кратчайших путей между отделениями и последующей оптимизации маршрута с использованием подхода динамического программирования для задачи коммивояжёра.

**Выводы:**

В ходе выполнения данной работы была успешно решена задача по нахождению оптимального маршрута для развоза почты между почтамтом и отделениями связи, при котором общее пройденное расстояние минимально. Программа эффективно использует алгоритмы для расчёта кратчайших путей и оптимизации маршрута, что позволило получить точный результат с минимальными вычислительными затратами.

Основные выводы:

1. **Эффективность алгоритма Дейкстры для поиска кратчайших путей**: Алгоритм Дейкстры успешно рассчитал кратчайшие расстояния между каждой парой вершин в графе, представляемом как матрица смежности. Это позволило преобразовать исходную задачу, связанную с реальными расстояниями между отделениями, в более упрощённую форму для дальнейшей оптимизации маршрута.
2. **Оптимизация маршрута с использованием задачи коммивояжёра**: После расчёта кратчайших расстояний между всеми вершинами, был применён подход к решению задачи коммивояжёра с помощью рекурсивного алгоритма с динамическим программированием. Этот подход позволяет учитывать все возможные маршруты и выбирать оптимальный путь, при котором каждое отделение посещается только один раз, и машина возвращается в почтамт. Данный метод доказал свою эффективность для решения задач такого типа.
3. **Результат в 21 км как минимально возможная длина маршрута**: Рассчитанный программой минимальный маршрут в 21 км подтверждает правильность работы алгоритма и его применения к исходной задаче. Этот результат представляет собой минимальное расстояние, которое может быть пройдено машиной для доставки почты по каждому отделению и возвращения в начальную точку.
4. **Применимость методов графов и оптимизации в реальных задачах логистики**: Работа показала, что алгоритмы теории графов, такие как алгоритм Дейкстры и динамическое программирование для задачи коммивояжёра, применимы к реальным логистическим задачам. Эти методы позволяют автоматизировать и оптимизировать задачи распределения ресурсов и построения маршрутов, что важно для минимизации времени и затрат в сферах транспорта и логистики.

В заключение, проделанная работа продемонстрировала высокую практическую ценность методов поиска кратчайших путей и оптимизации маршрутов для задач, связанных с логистикой и планированием маршрутов. Разработанный алгоритм может быть использован в похожих задачах для оптимизации транспортных и ресурсных затрат, что является актуальной задачей в различных сферах деятельности.

**Приложение**

#include <iostream>

#include <vector>

#include <limits>

#include <algorithm>

using namespace std;

const int INF = numeric\_limits<int>::max();

struct Graph {

    int V; // Число вершин (включая почтамт и отделения)

    vector<vector<int>> adjMatrix; // Матрица смежности для хранения расстояний между вершинами

    Graph(int V) : V(V), adjMatrix(V, vector<int>(V, INF)) {}

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

        adjMatrix[u][v] = weight;

        adjMatrix[v][u] = weight; // Граф неориентированный

    }

    // Реализация алгоритма Дейкстры для нахождения кратчайшего пути от вершины start

    vector<int> dijkstra(int start) {

        vector<int> dist(V, INF); // массив расстояний от start до всех вершин

        vector<bool> visited(V, false); // массив для отслеживания посещенных вершин

        dist[start] = 0;

        for (int i = 0; i < V; i++) {

            int u = -1;

            // Находим не посещенную вершину с минимальным расстоянием

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

                if (!visited[j] && (u == -1 || dist[j] < dist[u])) {

                    u = j;

                }

            }

            if (dist[u] == INF) break;

            visited[u] = true;

            // Обновляем расстояния до соседних вершин

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

                if (adjMatrix[u][v] != INF && dist[u] + adjMatrix[u][v] < dist[v]) {

                    dist[v] = dist[u] + adjMatrix[u][v];

                }

            }

        }

        return dist;

    }

    // Рекурсивная функция для нахождения минимального маршрута коммивояжера

    int travelingSalesman(int pos, int visited, vector<vector<int>>& dp, const vector<vector<int>>& distMatrix, vector<int>& path, vector<int>& bestPath) {

        if (visited == (1 << V) - 1) { // если все вершины посещены

            path.push\_back(0); // возвращаемся в почтамт

            int routeLen = distMatrix[pos][0];

            if (routeLen < INF) bestPath = path; // сохраняем путь

            path.pop\_back();

            return routeLen; // возвращаемся в почтамт

        }

        if (dp[pos][visited] != -1)

            return dp[pos][visited];

        int ans = INF;

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

            if ((visited & (1 << city)) == 0) { // если город ещё не посещен

                path.push\_back(city);

                int newAns = distMatrix[pos][city] + travelingSalesman(city, visited | (1 << city), dp, distMatrix, path, bestPath);

                if (newAns < ans) {

                    ans = newAns;

                    bestPath = path; // сохраняем текущий лучший путь

                }

                path.pop\_back();

            }

        }

        return dp[pos][visited] = ans;

    }

};

int main() {

    int V = 5; // 1 почтамт и 4 отделения связи

    Graph g(V);

    // Заданные расстояния (неориентированный граф)

    g.addEdge(0, 1, 5); // П-1

    g.addEdge(0, 2, 3); // П-2

    g.addEdge(0, 3, 4); // П-3

    g.addEdge(0, 4, 4); // П-4

    g.addEdge(1, 2, 3); // 1-2

    g.addEdge(1, 3, 5); // 1-3

    g.addEdge(1, 4, 6); // 1-4

    g.addEdge(2, 3, 4); // 2-3

    g.addEdge(2, 4, 7); // 2-4

    g.addEdge(3, 4, 7); // 3-4

    // Находим кратчайшие расстояния между всеми парами вершин с помощью алгоритма Дейкстры

    vector<vector<int>> distMatrix(V, vector<int>(V));

    for (int i = 0; i < V; i++) {

        distMatrix[i] = g.dijkstra(i);

    }

    // Инициализация для поиска минимального маршрута

    vector<vector<int>> dp(V, vector<int>(1 << V, -1));

    vector<int> path = {0}; // путь начинается с почтамта (вершина 0)

    vector<int> bestPath; // для хранения наилучшего пути

    int shortestPath = g.travelingSalesman(0, 1, dp, distMatrix, path, bestPath);

    // Выводим результаты

    cout << "Кратчайший маршрут для машины: " << shortestPath << " км" << endl;

    cout << "Последовательность движения: ";

    for (int i = 0; i < bestPath.size(); ++i) {

        if (bestPath[i] == 0 && i != 0 && i != bestPath.size() - 1) {

            cout << "Почтамт -> "; // выводим "Почтамт" для начальной и конечной позиции

        } else if (bestPath[i] == 0) {

            cout << "Почтамт";

        } else {

            cout << "ОС-" << bestPath[i];

        }

        if (i != bestPath.size() - 1) cout << " -> ";

    }

    cout << " -> Почтамт" << endl;

    return 0;

}