Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Лабораторная работа №3

“Задача Коммивояжера”

Руководитель ТР,

доц. кафедры ИТАС \_\_\_\_\_\_\_\_\_\_ О.А. Полякова

Пермь, 2025 г.

ВВЕДЕНИЕ

Задача коммивояжёра (Travelling Salesman Problem, TSP) является классической задачей комбинаторной оптимизации, целью которой является нахождение кратчайшего маршрута, проходящего через все заданные города ровно один раз с возвращением в начальную точку.

Программа, описанная ниже, позволяет пользователю вводить города и дороги, решать задачу коммивояжёра и визуализировать граф с возможностью интерактивного добавления дорог. Отчет описывает дизайнерские и конструкторские решения, UML-диаграмму, ключевые классы и функции, используемые инструменты, а также достижения, которыми гордится автор.

Входные данные: ненаправленный взвешенный граф, где вершины представляют города, а рёбра — дороги с весами (расстояниями).

Цель: найти гамильтонов цикл (маршрут, проходящий через все вершины ровно один раз) с минимальной суммарной длиной, начиная и заканчивая в первом городе (индекс 0).

**Дизайнерские и конструкторские решения**

Программа спроектирована с акцентом на модульность, интерактивность и удобство использования. Основные дизайнерские и конструкторские решения включают:

1. **Модульная структура**:

1) Программа разделена на два основных класса: Graph для управления данными графа и алгоритмической логики, и GraphEditor для обработки пользовательского ввода и визуализации.

2) Класс Graph инкапсулирует всю логику работы с графом, включая добавление/удаление узлов и рёбер, решение задачи коммивояжёра и отрисовку.

3) Класс GraphEditor отвечает за взаимодействие с пользователем через SFML, предоставляя интерактивный интерфейс.

1. **Алгоритмическая основа**:

1) Для решения задачи коммивояжёра используется рекурсивный алгоритм с возвратом (backtracking), который перебирает все возможные маршруты, начиная с первого города, и выбирает маршрут с минимальной стоимостью.

2) Граф реализован с использованием списка смежности (std::vector<std::vector<Edge>>), что обеспечивает эффективное хранение и доступ к рёбрам.

1. **Визуализация**:

1) Города отображаются как зелёные круги с подписями (названиями городов), а дороги — как чёрные линии с красными метками весов.

2) Города автоматически располагаются по окружности при вводе с консоли, что упрощает визуальное восприятие и предотвращает наложение узлов.

3) Интерактивный режим добавления дорог позволяет пользователю выбирать города кликом мыши и вводить веса через консоль.

1. **Интерактивность**:

1) Пользователь может добавлять дороги в реальном времени, переключая режим нажатием клавиши A.

2) Решение задачи коммивояжёра запускается нажатием клавиши T, с выводом результата в консоль.

3) Инструкции по управлению отображаются в окне SFML для удобства.

1. **Обработка данных**:

1) Ввод данных осуществляется через консоль, где пользователь задаёт количество городов, их названия и координаты, а также дороги и их веса.

2) Автоматическое обновление индексов при удалении узлов обеспечивает корректность структуры графа.

**UML-диаграмма и её разъяснение**

UML-диаграмма, представленная ниже, описывает структуру классов и их взаимосвязи:

**Класс Edge**:

-Атрибуты: to (индекс вершины назначения), weight (вес ребра).

-Конструктор: инициализирует ребро с заданным пунктом назначения и весом.

-Назначение: представляет направленное ребро в графе.

**Класс Graph**:

Атрибуты:

-adjacencyList: список смежности для хранения рёбер.

-nodePositions: координаты узлов для визуализации.

-edges: список всех рёбер для отрисовки.

-edgeWeights: веса рёбер для отображения.

-cityNames: названия городов.

Методы:

-Приватный метод tspRecursive реализует рекурсивное решение задачи коммивояжёра.

-Публичные методы для добавления/удаления узлов и рёбер, решения TSP, отрисовки и доступа к данным.

-Назначение: управляет структурой графа и алгоритмической логикой.

**Класс GraphEditor**:

Атрибуты:

-graph: объект графа для работы с данными.

-window: окно SFML для визуализации.

-selectedNode, addingEdge, edgeStart: переменные для управления интерактивным добавлением рёбер.

Методы:

-run: основной цикл программы.

-handleEvents: обработка событий мыши и клавиатуры.

-handleLeftClick: обработка кликов мыши для выбора городов.

-render: отрисовка графа и интерфейса.

-Назначение: обеспечивает интерактивный интерфейс и визуализацию.

**Связи**:

-Graph содержит множество объектов Edge через adjacencyList (композиция).

-GraphEditor содержит один объект Graph и один объект sf::RenderWindow (композиция).

UML диаграмма (рис 1):

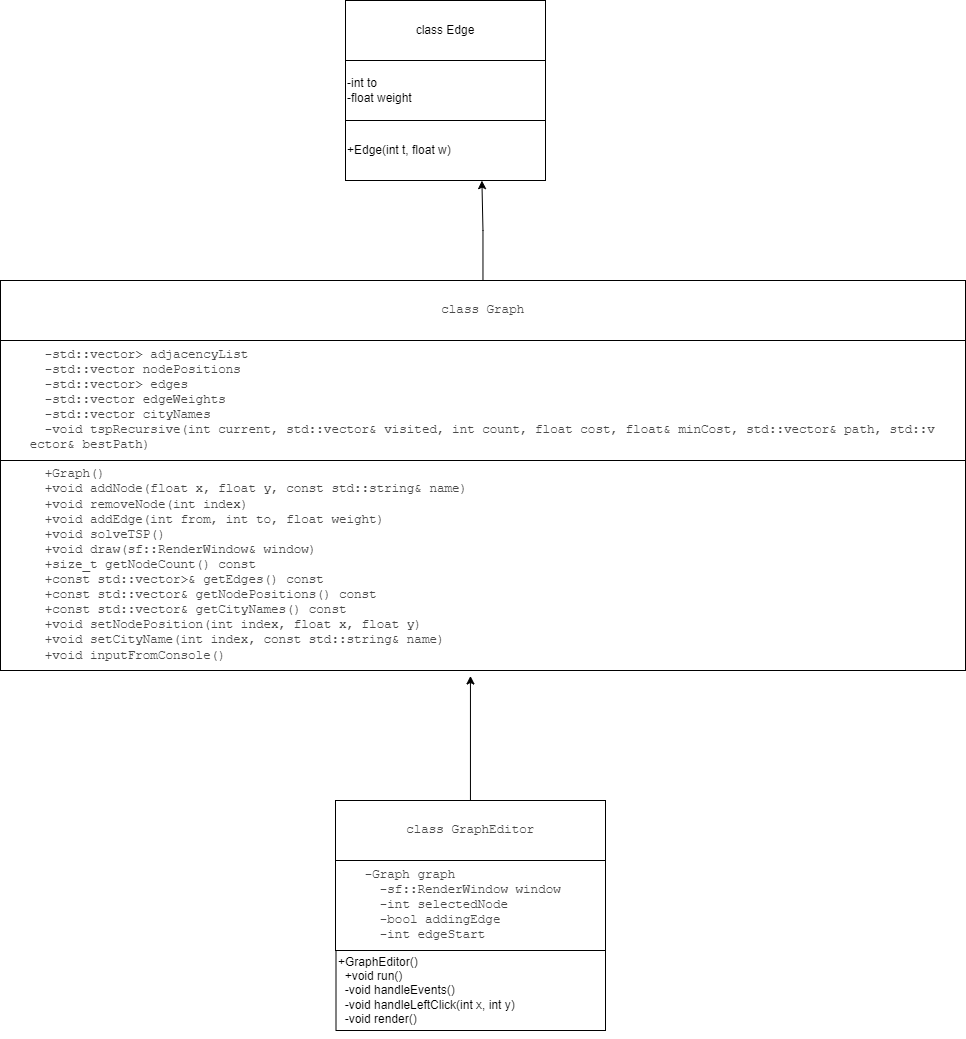


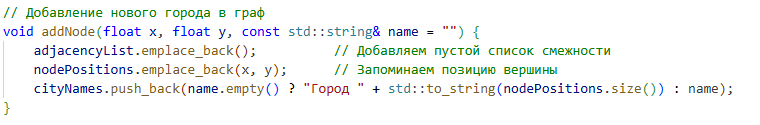
Рисунок 1 – UML диаграмма

Реализация ключевых классов и функций

**Класс Graph:**

Конструктор: инициализирует пустой граф с использованием стандартных контейнеров STL (std::vector).

*Метод addNode:*

 Добавляет новый узел с координатами и названием, автоматически присваивая имя, если оно не указано.

*Метод addEdge:*

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки. Добавляет двустороннее ребро между городами с заданным весом.

*Метод solveTSP:*

** Решает задачу коммивояжёра, используя рекурсивный алгоритм с возвратом, и выводит оптимальный маршрут и его стоимость.

*Метод draw:*



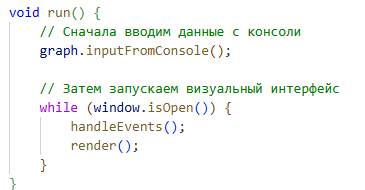
Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки. Рисует граф, включая линии дорог, круги городов и подписи с названиями и весами.

**Класс GraphEditor:**

Конструктор: создаёт окно SFML размером 800x600 пикселей.

*Метод run:*

******

Запускает основной цикл программы, сначала запрашивая данные с консоли, затем обрабатывая события и отрисовывая интерфейс.

*Метод handleLeftClick:*

***Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.*** Обрабатывает клики мыши для выбора городов и добавления рёбер.

**Используемые инструменты, технологии и программы**

1. Язык программирования: C++ (стандарт C++11 и выше) для реализации логики и работы с графикой.
2. Библиотека SFML (Simple and Fast Multimedia Library):

Использована для создания графического интерфейса, отрисовки узлов, рёбер и текста.

Требует наличия шрифта arial.ttf для отображения подписей.

1. STL (Standard Template Library):

std::vector для хранения списков смежности, позиций узлов, рёбер и названий городов.

std::pair для представления рёбер.

std::numeric\_limits для инициализации минимальной стоимости в задаче TSP.

1. Среда разработки: Код может быть скомпилирован и запущен в любой IDE, поддерживающей C++ и SFML (например, Visual Studio, CLion, Code::Blocks).
2. Операционная система: Программа кроссплатформенная благодаря SFML, но тестировалась с учётом наличия шрифта arial.ttf, что предполагает наличие стандартных шрифтов Windows или их аналогов.

Достижения

1. **Полная функциональность задачи коммивояжёра**:

Реализован рекурсивный алгоритм с возвратом для поиска оптимального маршрута, который корректно обрабатывает графы с двусторонними рёбрами.

1. **Интерактивный интерфейс**:

Возможность добавления дорог в реальном времени через графический интерфейс делает программу удобной и интуитивной.

1. **Визуализация**:

Граф отображается с чёткими визуальными элементами (круги для городов, линии для дорог, подписи с весами и названиями), что улучшает восприятие структуры графа.

1. **Модульность**:

Разделение логики на классы Graph и GraphEditor упрощает поддержку и расширение программы.

1. **Автоматическое размещение узлов**:

Города располагаются по окружности при вводе с консоли, что предотвращает наложение и улучшает читаемость графа.

1. **Гибкость ввода данных**:

Пользователь может вводить данные через консоль, а затем взаимодействовать с графом через интерфейс, что делает программу универсальной.

**Заключение**

Программа представляет собой полноценное решение задачи коммивояжёра с интерактивным визуальным интерфейсом. Использование C++, SFML и STL позволило создать эффективное и удобное приложение. UML-диаграмма демонстрирует чёткую структуру классов, а модульный дизайн упрощает дальнейшую разработку.

**Список использованных источников**

1 Страуструп Б. Язык программирования С++. – 1997–№1 – С.76.

2 Страуструп Б. Программирование. Принципы и практика использования C++. – 2011–№1 – С. 279.

3 Кнут Д.Э. Искусство программирования. – 1968–№1 – С. 98.

4 Керниган Б. Язык программирования С. – 1978–№1 – С. 116.

5 Сложность алгоритмов и программ. –URL: https://cyberleninka.ru /article /n/ slozhnost-algoritmov-i-programm (дата обращения 15.05.2024).