Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ

УДК 004.65

Отчёт по творческой работе

по дисциплине “Информатика” на тему:

“Решение задачи коммивояжёра”

Выполнили

Студенты группы РИС-23-1б

Жуланов Н.А.

Валов И.И.

Проверил доц. кафедры ИТАС

Петренко А.А.

Пермь 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ3

1 АНАЛИЗ ЗАДАЧИ И ВЫБОР МЕТОДА РЕШЕНИЯ5

1.1 Формулировка задачи коммивояжёра5

1.2 Обзор методов решения задачи коммивояжёра5

1.3 Обоснование выбора метода ветвей и границ7

2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ8

2.1 Выбор фреймворка и инструментов разработки8

2.2 Разработка структуры данных для представления графа9

2.3 Разработка алгоритма решения задачи коммивояжёра методом ветвей и границ10

2.4 Разработка графического интерфейса пользователя12

2.5 Реализация визуализации графа13

ЗАКЛЮЧЕНИЕ15

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ17

ПРИЛОЖЕНИЕ А18

ПРИЛОЖЕНИЕ Б19

ПРИЛОЖЕНИЕ В20

ВВЕДЕНИЕ

*Задача коммивояжёра* (Travelling Salesman Problem, TSP) — одна из классических задач комбинаторной оптимизации, имеющая широкое практическое применение в различных областях. Она возникает в задачах логистики и маршрутизации (оптимизация доставки товаров, планирование маршрутов транспорта), в микроэлектронике (проектирование печатных плат), в биоинформатике (секвенирование генома) и многих других.

Существует множество методов решения задачи коммивояжёра, которые можно разделить на точные и приближённые. Точные методы (полный перебор, метод ветвей и границ) гарантируют нахождение оптимального решения, однако их вычислительная сложность экспоненциально возрастает с увеличением количества городов, что делает их неприменимыми для больших графов. Приближённые методы (жадные алгоритмы, генетические алгоритмы, муравьиный алгоритм) позволяют найти достаточно хорошее решение за приемлемое время, но не гарантируют оптимальности.

В данной работе рассматривается разработка приложения для визуализации и решения задачи коммивояжёра методом ветвей и границ. Этот метод представляет собой компромисс между точностью и временем выполнения, позволяя находить оптимальное решение для небольших графов и демонстрируя наглядно процесс поиска решения, что делает его подходящим для учебных целей.

Целью работы является *анализ* оптимальной реализации и *проектирование* приложения, которое позволит пользователю:

1. Вводить данные о графе (количество вершин, матрицу весов).
2. Визуализировать граф.
3. Рассчитывать кратчайший маршрут методом ветвей и границ.
4. Отображать найденный маршрут и его длину.

Разработанное приложение позволит наглядно продемонстрировать работу алгоритма ветвей и границ, а также позволит пользователю экспериментировать с различными графами и оценивать эффективность метода.

1. АНАЛИЗ ЗАДАЧИ И ВЫБОР МЕТОДА РЕШЕНИЯ

1.1 Формулировка задачи коммивояжёра

Задача коммивояжёра формулируется следующим образом [2]:

Дано: Полный взвешенный граф G = (V, E), где V — множество вершин (городов), E — множество рёбер (путей между городами), каждому ребру (i, j) ∈ E присвоен вес , представляющий собой расстояние (или стоимость проезда) между городами i и j.

Требуется: найти гамильтонов цикл минимальной длины (стоимости), то есть цикл, проходящий через каждую вершину графа ровно один раз и возвращающийся в начальную вершину.

Ограничения:

* Граф должен быть полным, то есть между каждой парой вершин должно существовать ребро.
* Веса рёбер должны быть неотрицательными числами.

*Вывод*: Задача коммивояжёра является NP-трудной задачей, что означает, что для неё не существует известного алгоритма, который бы находил оптимальное решение за полиномиальное время.

1.2 Обзор методов решения задачи коммивояжёра

Существует множество методов решения задачи коммивояжёра, которые можно классифицировать следующим образом [1]:

1. Точные методы:

* *Полный перебор*: проверяются все возможные гамильтоновы циклы и выбирается цикл с минимальной длиной. Этот метод гарантирует нахождение оптимального решения, но его вычислительная сложность равна (n-1)!, где n — количество вершин графа, что делает его неприменимым для графов с большим количеством вершин.
* *Метод ветвей и границ*: этот метод основан на построении дерева поиска, где каждый узел представляет собой подмножество возможных решений. Алгоритм последовательно рассматривает узлы дерева, отсекая ветви, которые заведомо не могут привести к оптимальному решению. Метод ветвей и границ может найти оптимальное решение, но его время выполнения может быть экспоненциальным в худшем случае.

1. Приближённые методы:

* *Жадные алгоритмы*: эти алгоритмы строят решение пошагово, на каждом шаге выбирая ребро, которое кажется наиболее выгодным в данный момент. Жадные алгоритмы работают быстро, но не гарантируют нахождение оптимального решения. Примерами жадных алгоритмов являются алгоритм ближайшего соседа, алгоритм вставки.
* *Метаэвристики*: это более сложные алгоритмы, использующие случайный поиск и другие эвристики для поиска приближённого решения. Метаэвристики, как правило, более эффективны, чем жадные алгоритмы, но также не гарантируют оптимальность. Примеры метаэвристик: генетические алгоритмы, муравьиный алгоритм, имитация отжига.

*Вывод*: Выбор метода решения задачи коммивояжёра зависит от размера графа и требований к точности решения. Для небольших графов можно использовать точные методы, а для больших графов — приближённые методы.

1.3 Обоснование выбора метода ветвей и границ

Для реализации в приложении был выбран метод ветвей и границ по следующим причинам [3]:

* *Компромисс между точностью и временем выполнения*: Метод ветвей и границ позволяет найти оптимальное решение для графов среднего размера (до 20-30 вершин) за приемлемое время.
* *Наглядная демонстрация процесса поиска решения*: Метод ветвей и границ основан на построении дерева поиска, которое можно визуализировать, что позволяет пользователю наглядно увидеть, как работает алгоритм.
* *Подходит для учебных целей*: Метод ветвей и границ является относительно простым для понимания и реализации, что делает его хорошим выбором для учебного приложения.

*Вывод*: Метод ветвей и границ является подходящим методом для решения задачи коммивояжёра в разрабатываемом приложении, учитывая его компромисс между точностью, временем выполнения и возможностью наглядной демонстрации процесса поиска решения.

1. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Выбор фреймворка и инструментов разработки

Для разработки приложения был выбран фреймворк *Qt* (версия 6.7.0) и язык программирования *C++.* Обоснование выбора [4]:

* *Кроссплатформенность***:** Qt позволяет создавать приложения, работающие на различных операционных системах (Windows, macOS, Linux), что расширяет аудиторию пользователей.
* *Удобство создания графического интерфейса***:** Qt предоставляет богатый набор виджетов и инструментов для создания удобных и современных пользовательских интерфейсов (GUI).
* *Наличие инструментов для работы с графами***:** Qt содержит классы, позволяющие рисовать и манипулировать графами, что упрощает реализацию визуализации.

*Основные библиотеки и классы Qt, используемые в приложении***:**

* *QWidget***:** Базовый класс для всех пользовательских интерфейсных элементов.
* *QMainWindow***:** Класс для создания главного окна приложения.
* *QPainter***:** Класс для рисования на виджетах.
* *QTableWidget***:** Класс для создания таблиц.
* *QApplication***:** Класс, управляющий основным циклом событий приложения.
* *QStyleFactory***:** Класс для изменения стиля оформления приложения.
* *QPalette***:** Класс для управления цветовой палитрой приложения.
* *QPushButton***:** Класс для создания кнопок.
* *QLineEdit***:** Класс для создания полей ввода текста.
* *QTextEdit***:** Класс для создания многострочных текстовых полей.

*Вывод***:** Фреймворк Qt и язык C++ предоставляют все необходимые инструменты для разработки кроссплатформенного приложения с графическим интерфейсом для решения задачи коммивояжёра.

2.2 Разработка структуры данных для представления графа

Для представления графа в приложении был разработан класс *Graph*.

Описание класса*Graph***:**

* Поля**:**
  1. *vertexCount*: Целое число, хранящее количество вершин в графе.
  2. *adjacencyMatrix*: Вектор векторов целых чисел, представляющий матрицу смежности графа. Элемент *adjacencyMatrix[i][j]* хранит вес ребра между вершинами *i* и *j*.
* Методы**:**
  1. *getVertexCount()*: Возвращает количество вершин в графе.
  2. *getEdgeWeight(int i, int j)*: Возвращает вес ребра между вершинами *i* и *j*.
  3. *setEdgeWeight(int i, int j, int weight)*: Устанавливает вес ребра между вершинами *i* и *j* равным *weight*.

*Пример использования представлен в приложении А.*

*Вывод***:** Класс *Graph* предоставляет удобный способ хранения и манипулирования данными о графе, необходимыми для решения задачи коммивояжёра.

2.3 Разработка алгоритма решения задачи коммивояжёра методом ветвей и границ

Для решения задачи коммивояжёра в приложении реализован алгоритм *метода ветвей и границ*.

*Описание алгоритма***:**

1. *Инициализация***:**
   * Создаётся очередь с приоритетом *pq*, в которую помещается начальный узел, представляющий собой вершину 0 и пустой путь.
   * Устанавливается минимальная стоимость маршрута *minCost* равной бесконечности.
2. *Цикл***:**
   * Пока очередь *pq* не пуста:
     + Извлекается узел с наименьшей стоимостью из *pq*.
     + Если уровень узла равен количеству вершин в графе:
       - Вычисляется полная стоимость маршрута, добавляя стоимость возврата в начальную вершину.
       - Если полная стоимость меньше *minCost*, обновляется *minCost* и сохраняется найденный маршрут.
   * Иначе:

Для каждой непосещённой вершины, смежной с текущей:

* + - * Создаётся новый узел, добавляя вершину в путь и обновляя стоимость.
      * Если стоимость нового узла меньше *minCost*, он добавляется в очередь *pq*.

1. *Результат***:**

Возвращается найденный маршрут с минимальной стоимостью *minCost*.

*Структуры данных:*

* *Node*: Структура, представляющая узел в дереве поиска. Содержит поля: *vertex* (текущая вершина), *level* (уровень в дереве), *cost* (стоимость пути до текущей вершины), *path* (вектор вершин, представляющий путь), *visited* (вектор булевых значений, указывающих, посещена ли вершина).
* *std::priority\_queue<Node, std::vector<Node>, Comp>*: Очередь с приоритетом, хранящая узлы дерева поиска. Узлы сортируются по возрастанию стоимости.

*Вывод***:** Реализованный алгоритм метода ветвей и границ позволяет эффективно находить оптимальное решение задачи коммивояжёра для графов среднего размера.

2.4 Разработка графического интерфейса пользователя

Графический интерфейс приложения разработан с использованием Qt Designer и состоит из следующих элементов:

* Поле ввода количества вершин **(***vertexCountLineEdit***):** Позволяет пользователю ввести количество вершин в графе.
* Таблица для ввода матрицы смежности **(***adjacencyMatrixTableWidget***):** Таблица, в которой пользователь может ввести веса рёбер графа.
* Кнопка создания графа **(***createGraphButton***):** Создает таблицу для ввода матрицы смежности с заданным количеством вершин.
* Кнопка построения графа **(***buildGraphButton***):** Строит граф на основе введенной матрицы смежности и отображает его в отдельном окне.
* Кнопка расчёта маршрута **(***calculateButton***):** Запускает алгоритм решения задачи коммивояжёра и отображает результат.
* Поле вывода результата **(***resultTextEdit***):** Отображает найденный маршрут и его длину.

*Вид интерфейса главного окна представлен в приложении Б, рисунок Б.1.*

*Вывод***:** Графический интерфейс приложения предоставляет пользователю удобный способ ввода данных, визуализации графа и просмотра результатов решения задачи коммивояжёра.

2.5 Реализация визуализации графа

Для визуализации графа используется класс *GraphWidget*, наследуемый от *QWidget*.

Описаниекласса*GraphWidget***:**

Метод*paintEvent()***:**

* + Вызывается при необходимости перерисовки виджета.
  + Рисует вершины графа в виде окружностей, равномерно распределённых по окружности.
  + Рисует рёбра графа в виде линий, соединяющих соответствующие вершины.
  + Отображает веса рёбер рядом с линиями.

*Алгоритм размещения вершин***:**

1. Вычисляется центр виджета.
2. Вычисляется радиус окружности, на которой будут размещены вершины.
3. Для каждой вершины вычисляются её координаты на окружности с использованием тригонометрических функций:
   * *x = centerX + radius \* cos(2 \* M\_PI \* i / vertexCount)*
   * *y = centerY + radius \* sin(2 \* M\_PI \* i / vertexCount)*  
     где *i* - номер вершины, *vertexCount* - количество вершин.

*Алгоритм рисования рёбер и весов***:**

Для каждой пары вершин, между которыми есть ребро:

* + Рисуется линия между соответствующими точками на окружности.
  + Вычисляются координаты середины ребра.
  + Отображается вес ребра рядом с серединой ребра.

*Вид интерфейса окна визуализации представлен в приложении Б, рисунок Б.2.*

*Общая структура программы в виде UML диаграммы класса представлена в приложении В.*

*Вывод***:** Класс *GraphWidget* обеспечивает наглядную визуализацию графа, позволяя пользователю легко воспринимать информацию о структуре графа и весах его рёбер.

ЗАКЛЮЧЕНИЕ

В ходе работы была достигнута поставленная цель: разработано приложение для решения задачи коммивояжёра методом ветвей и границ, позволяющее пользователю вводить данные о графе, визуализировать его, рассчитывать оптимальный маршрут и отображать результат.

Разработанное приложение обладает следующими преимуществами:

* *Наглядность***:** Приложение визуализирует граф и найденный маршрут, что делает результаты решения задачи более понятными и интерпретируемыми.
* *Простота**использования***:** Интуитивно понятный графический интерфейс позволяет легко освоить функциональность приложения even пользователям, не знакомым с программированием.
* *Кроссплатформенность***:** Благодаря использованию фреймворка Qt, приложение может запускаться на различных операционных системах (Windows, macOS, Linux), что делает его доступным для широкой аудитории.

Однако, приложение имеет и некоторые недостатки:

* *Ограниченная эффективность для больших графов***:** Метод ветвей и границ, используемый для решения задачи, имеет экспоненциальную сложность, что ограничивает его применимость для графов с большим количеством вершин.
* *Реализован только один метод решения***:** В приложении реализован только метод ветвей и границ, что не позволяет сравнить его эффективность с другими методами решения задачи коммивояжёра.

*Перспективы дальнейшего развития проекта***:**

* *Реализация других методов решения задачи коммивояжёра***:** Добавление жадных алгоритмов, генетических алгоритмов и других методов позволит сравнить их эффективность и выбрать наиболее подходящий для конкретной задачи.
* *Работа с разными типами графов***:** Добавление функциональности для работы с ориентированными и взвешенными графами расширит область применения приложения.
* *Оптимизация производительности***:** Оптимизация алгоритма решения задачи и визуализации графа позволит работать с большими графами.
* *Сохранение и загрузка графов из файла***:** Добавление возможности сохранять и загружать графы из файла упростит работу с приложением и позволит хранить и использовать различные тестовые данные.

В целом, разработанное приложение является полезным инструментом для изучения и решения задачи коммивояжёра, а также может быть использовано для решения практических задач оптимизации маршрутов. Планируемые улучшения позволят сделать приложение более функциональным и эффективным, расширив его возможности и область применения.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ. — М.: Вильямс, 2019. — 1328 с.
2. *Шрайбер Т.* Линейное и целочисленное программирование. — М.: Мир, 1981. — 392 с.
3. *Штовба С.Д.* Методы решения задачи коммивояжёра. — СПб.: Питер, 2002. — 256 с.
4. *Шлее М.* Qt 5.10. Профессиональное программирование. — СПб.: БХВ-Петербург, 2018. — 912 с.

ПРИЛОЖЕНИЯ

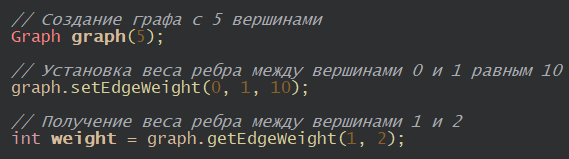
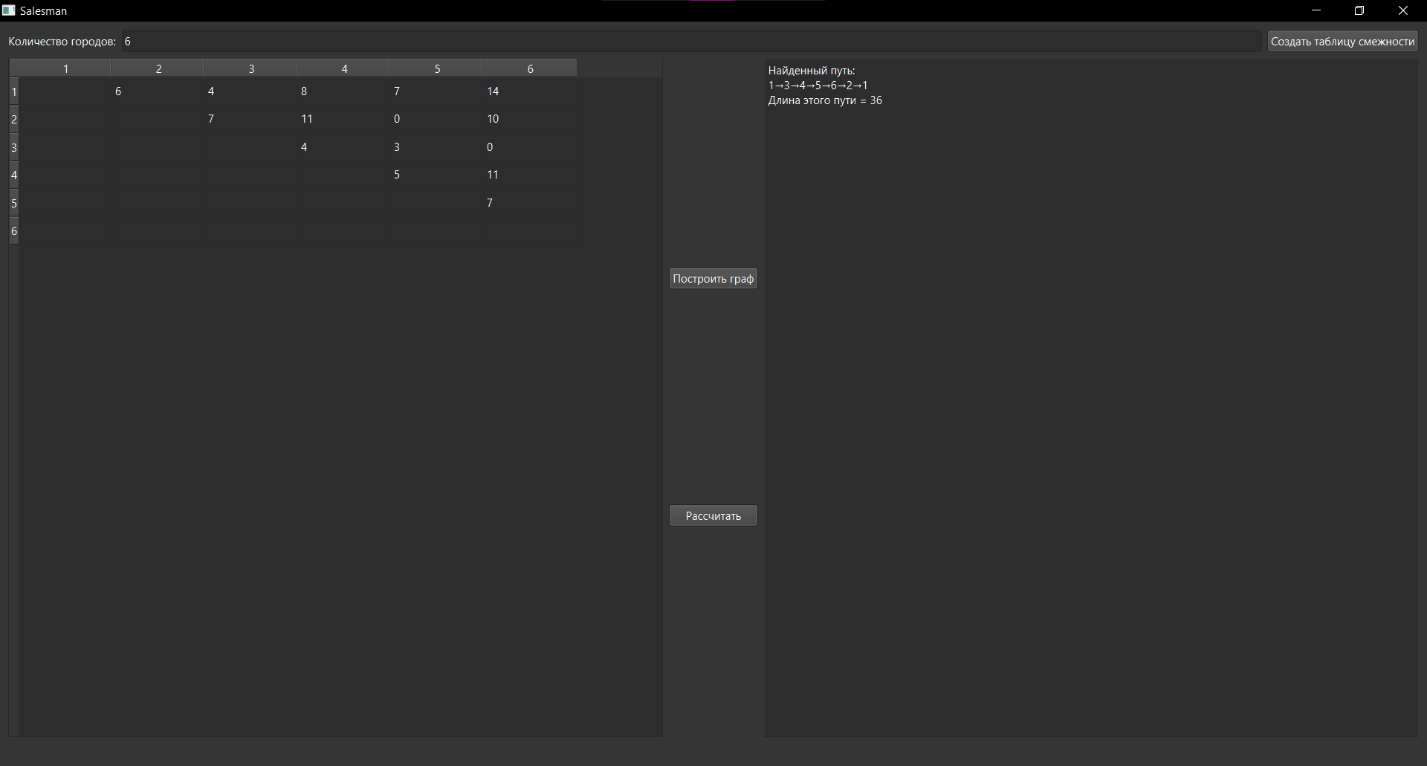
ПРИЛОЖЕНИЕ А

Рисунок А – Пример использования класса *Graph*

ПРИЛОЖЕНИЕ Б

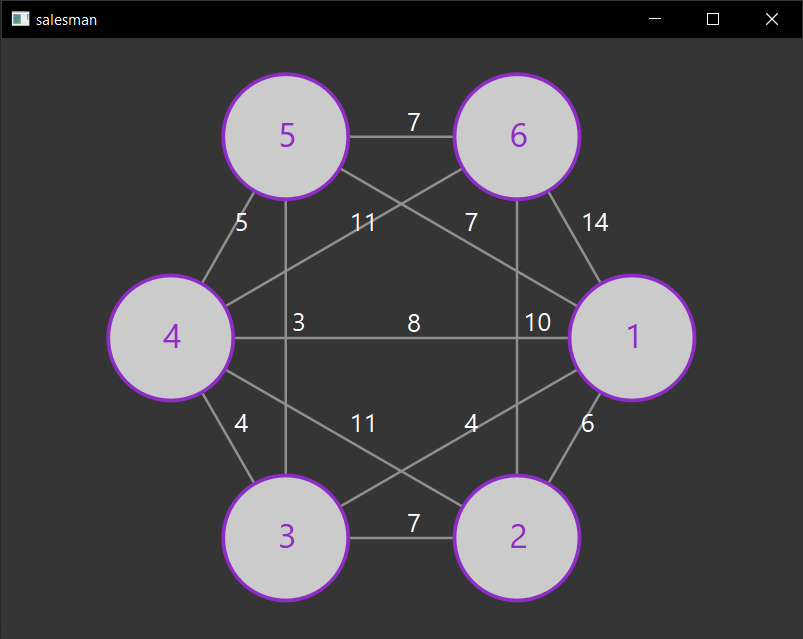
Рисунок Б.1 – вид интерфейса главного окна программы

Рисунок Б.2 – вид интерфейса окна визуализации

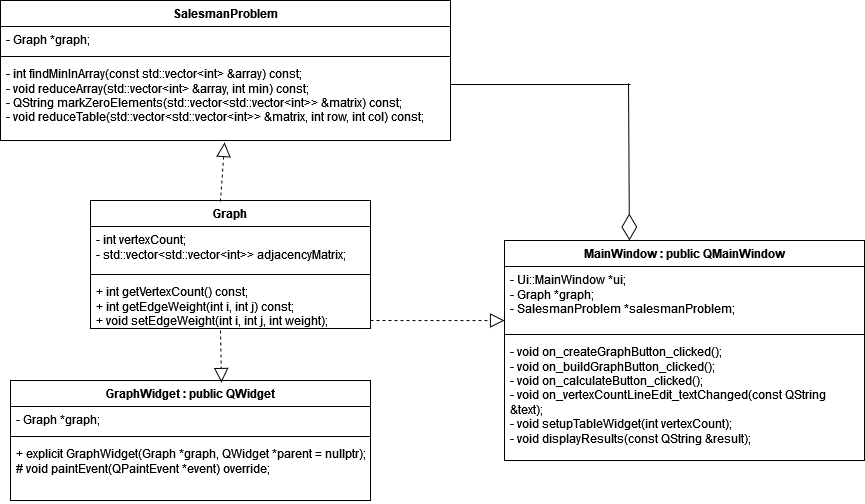
ПРИЛОЖЕНИЕ В

Рисунок В – UML диаграмма классов приложения