МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка алгоритма оптимального планирования маршрутов между клиентами

Обучающийся А. В. Зайцева _____
(Инициалы Фамилия) (личная подпись)

Руководитель к. ф.-м. н. О. В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н., доцент С. А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

## Аннотация

Работа была выполнена студентом Тольяттинского Государственного Университета, группы ПМИб-2002а, Зайцевой Ариной Вячеславовной.

Выпускная квалификационная работа по теме «Разработка алгоритма оптимального планирования маршрутов между клиентами» посвящена реализации задачи поиска кратчайшего пути в графе.

Цель работы - разработка программного обеспечения для нахождения кротчайшего пути среди множества клиентов, которым оказывается услуга на дому.

Объект исследования - процесс построения оптимального маршрута между клиентами, находящимся в разных точках города.

Предмет исследования - алгоритм нахождения кротчайшего маршрута между клиентами, с целью минимизации затрат на логистику и увеличение числа обслуживаемых клиентов в единицу времени.

Задачи работы:

- выбрать оптимальный алгоритм поиска кратчайшего пути,
- выбрать язык программирования и средства для разработки программного обеспечения,
- реализовать выбранный алгоритм в программном обеспечении,
- протестировать полученный продукт и исправить ошибки, при их наличии.

Выпускная квалификационная работа содержит пояснительную записку объемом 42 страницы, состоит из введения, трех разделов, заключения, списка литературы, состоящего из 25 литературных источников и 32 рисунков, а также приложения «А» с исходным кодом.

## Abstract

The work was performed by a student of Tolyatti State University, Institute of Mathematics, Physics and Information Technology, PMIb-2002a group, Arina Vyacheslavovna Zaitseva.

The final qualification work on the topic "Development of an algorithm for optimal route planning between clients in a window installation company" is devoted to the implementation of the task of finding the shortest path in a graph.

The purpose of the work is to develop software to find the shortest path among the many customers who receive the service at home.

The object of the study is the process of building an optimal route between clients located in different parts of the city.

The subject of the study is an algorithm for finding the shortest route between customers, in order to minimize logistics costs and increase the number of customers served per unit of time.

Tasks of the work:

- choose the optimal algorithm for finding the shortest path,
- choose a programming language and tools for software development,
- implement the selected algorithm in the software,
- test the resulting product and correct errors, if any.

The final qualifying work contains an explanatory note of 42 pages, consists of an introduction, three sections, a conclusion, a list of references consisting of 25 literary sources and 32 drawings, as well as an appendix «A» with source code.

**Содержание**

# Введение

В наше информационное время, в условиях стремительного развития технологий и повышенной конкуренции, эффективное управление логистическими процессами становится важной частью успешной деятельности компаний и предприятий, предоставляющих услуги, которые оказываются непосредственно у клиентов на дому. Это обусловлено не только растущим спросом на такие услуги, но и высокой необходимостью оптимизации времени и ресурсов при обслуживании клиентов.

Исходя из этого, разработка программного обеспечения для поиска оптимального маршрута между клиентами становится стратегически важной задачей. Такая программа предоставляет компаниям возможность не только сократить затраты на транспортировку, а также увеличить производительность, но и повысить уровень обслуживания, что в конечном итоге способствует укреплению их позиций на рынке и росту числа клиентов.

В данной выпускной работе мы сосредоточим внимание на процессе разработки и реализации подобной программы, нацеленной на улучшение логистических функций компаний, специализирующихся на оказании услуг в домах клиентов. Мы рассмотрим ключевые принципы, лежащие в основе работы такой программы, ее функциональные возможности и потенциальные преимущества для бизнеса. Также будет рассмотрено влияние данного подхода на уровень сервиса и удовлетворенность клиентов.

В ходе выполнения данной выпускной работы объектом исследования будут являться методы оптимизации для решения логистических задач. В данном объекте исследования выделим предмет, который будем изучать в ходе написания данной ВКР – метод ветвей и границ для решения задачи коммивояжёра.

Цель данной выпускной квалификационной работы состоит в нахождении кратчайшего маршрута между клиентами для минимизации

затрат на логистику и увеличения числа обслуживаемых клиентов в единицу времени.

В процессе выполнения работы необходимо будет выполнить следующие задачи:

- анализ математических методов поиска кратчайшего пути,
- обоснование выбора метода ветвей и границ для аналитического решения задачи коммивояжёра, сформулированной в ВКР,
- аналитическое решение задачи коммивояжёра методом ветвей и границ,
- выбор языка программирования и средств для создания программного кода,
- реализация выбранного алгоритма в виде программного кода,
- тестирование программного кода.

В первом разделе мы рассмотрим поставленную задачу и проведем анализ существующих решений. Во втором разделе мы проанализируем решение задачи коммивояжёра один из выбранных методов. Третий раздел будет содержать в себе разработку программного обеспечения и его тестирования.

# 1. Теоретическое обоснование задачи

## 1.1 Описание исследуемой задачи

Задачей разработки программного обеспечения для поиска оптимального маршрута между клиентами в компаниях по установке пластиковых окон является создание инструмента, способного эффективно использовать имеющиеся логистические ресурсы (такие как автотранспорт, время и трудовые ресурсы) с целью максимизации количества обслуженных клиентов в единицу времени. Для решения этой задачи следует пройти ряд этапов:

- определить цели и критерии эффективности, выявить цели, которые должны быть достигнуты при оптимизации маршрутов обслуживания клиентов. Например, сокращение времени на доставку, уменьшение расходов на топливо, повышение удовлетворенности клиентов и т.д. Затем следует определить критерии для оценки эффективности программы, например, среднее время в пути, количество доставленных заказов за единицу времени и др;
- разработать математическую модель или алгоритм, который будет оптимально распределять ресурсы для достижения поставленных целей на основе данных анализа. Это включает в себя выбор оптимальных маршрутов, учет времени в пути, географических особенностей и других факторов;
- провести тестирования разработанной программы на реальных данных или в условиях симуляции. В случае необходимости внести корректировки и адаптировать программу для лучшего соответствия требованиям и условиям конкретной компании;
- внедрить разработанную программу на предприятии и ее регулярно

мониторить ее работу. Анализировать результаты работы программы с целью выявления возможных улучшений или корректировок для дальнейшего совершенствования логистических операций и услуг.

## 1.2 Анализ существующих решений

На текущий момент одним из наиболее распространенных инструментов для оптимизации маршрутов является программное обеспечение класса Route Optimization Software (ROS) [15]. Эти программы предоставляют широкий спектр функций для решения логистических задач, таких как оптимизация маршрутов доставки и управление транспортными ресурсами.

Одним из наиболее известным примером такого программного обеспечения является продукт под названием Route4Me, который предлагает широкий функционал для расчета оптимальных маршрутов доставки на основе различных критериев, таких как время в пути, расходы на топливо и т.д [17]. Эта программа также позволяет учитывать как ограничения, так и условия маршрута, например запретные зоны или требования к грузоподъемности перевозимого груза.

Кроме того, существуют специализированное программное обеспечение для оптимизации маршрутов в конкретных отраслях, такие как Transporeon TMS для транспортной логистики или WorkWave Route Manager для сервисных компаний [24].

Однако, несмотря на широкое распространение и функциональные возможности таких продуктов, они также имеют недостатки, которые могут быть критичными для определенных компаний или определенных задач:

- во-первых, большинство этих программ имеют высокую сложность настройки и использования, а также требуют значительных временных и финансовых затрат на настройку и

обучение персонала [21];

- во-вторых, эти программы имеют ограниченный функционал: Некоторые программы могут не учитывать специфические требования для конкретной компании, что может привести к необходимости дополнительной настройки или использованию различных инструментов [11];
- в-третьих, в процессе решении задач присутствует зависимость от качества данных: Точность маршрутов зависит от качества и достоверности входных данных об адресах клиентов, объектах доставки и условий дорожного движения [22].

Выводы по первому разделу:

Хотя программное обеспечение для оптимизации маршрутов предоставляет мощные инструменты для решения задач логистики, включая поиск оптимальных маршрутов между клиентами, оно также имеет свои недостатки, которые необходимо учитывать при выборе оптимального метода решения для конкретной компании.

## 2. Математическая формулировка модели

### 2.1 Анализ и выбор вычислительного метода

Рассмотрим несколько алгоритмов поиска кратчайшего пути в взвешенном графе.

- жадный алгоритм. При использовании данного подхода, на каждом шаге следующая вершина выбирается так, чтобы расстояние до неё было наименьшим из всех доступных вершин в данный момент. Этот метод будет эффективен для некоторых типов графов, но не гарантирует оптимального результата [2],[3];
- метод ветвей и границ (Branch and Bound). Этот метод используется для решения комбинаторных задач, а также может применяться для нахождения оптимального пути прохождения через вершины графа, минимизируя общую длину. Метод основан на построении дерева поиска, применении эвристических оценок и отсечении поддеревьев, которые не могут привести к наикратчайшему решению [5],[7];
- метод динамического программирования. Граф можно рассмотреть, как последовательность подграфов, где каждый подграф содержит вершины, которые ещё не были посещены. Затем можно использовать методы динамического программирования для нахождения кратчайшего пути через каждый из этих подграфов с учётом уже посещённых вершин [4];
- метод возврата к исходной вершине. В этом методе можно использовать обход графа в глубину или в ширину для поиска всех возможных путей, начиная с начальной вершины и возвращаясь в неё после прохождения всех остальных вершин. После чего выбирается самый короткий найденный путь [10].

В ходе выполнения данной выпускной работы будем использовать

метод ветвей и границ, так как данный метод гарантирует нахождение оптимального решения в задачах оптимизации. Это особенно важно в случаях, когда требуется именно точное решение, а не приближенное [6],[19]. Еще одной отличительной чертой метода ветвей и границ является возможность для настройки под конкретную задачу. Эвристики и стратегии отсечения ветвей могут быть адаптированы для повышения скорости работы алгоритма и улучшения его результатов.

Также стоит отметить, что задачи, решаемые методом ветвей и границ, поддаются параллельной и распределённой обработке, что позволяет эффективно использовать мощности многоядерных процессоров и кластеров вычислительных узлов для ускорения вычислений [10].

Реализация данного метода может потребовать дополнительных усилий, в сравнении с жадным методом или динамическим программированием, но она обеспечивает лучшие результаты для оптимизационных задач, таких как поиск оптимального пути через граф.

### 2.2 Общая структура алгоритма

Алгоритм метода ветвей и границ можно разделить на 4 шага.

А. Инициализация:

- задание начальных данных и параметров задачи,
- создание структуры данных для хранения текущего лучшего решения и текущей нижней оценки (lower bound),

Б. Составление начальной ветви:

- определение начального состояния,
- создание начальной ветви решения,

В. Цикл поиска:

- пока есть неисследованные ветви:
    а) выбор следующей ветви для исследования,

б) проверка критериев останова (например, достижение границы или улучшение текущего решения),

в) если ветвь может привести к оптимальному решению:

- вычисление нижней оценки для этой ветви,
- прореживание ветви, если её нижняя оценка хуже текущего лучшего решения,
- рекурсивное исследование дочерних ветвей (если они есть),

- возврат к предыдущей ветви и продолжение поиска,

Г. Заключение:

- возврат лучшего найденного решения.

### 2.3 Пример использования алгоритма.

Рассмотрим алгоритм метода ветвей и границ на собственном примере. Для этого построим взвешенный граф с пятью вершинами и произвольным весами ребер. Предположим, что вершины — это города, обозначенные латинскими буквами, а ребра – расстояния между ними. Полученный граф представлен на рисунке 1.

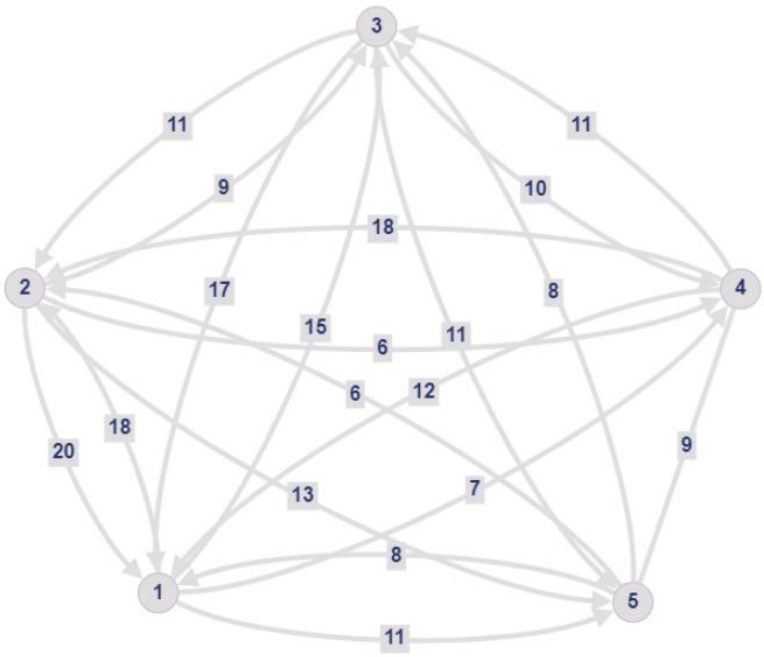


Рисунок 1 – Произвольный граф

Далее построим матрицу смежностей для данного графа. В клетках, расположенных на пересечениях строк и столбцов, указано расстояние между соответствующими городами. Полученная матрица представлена на рисунке 2.

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 18 | 15 | 7 | 11 |
| B | 20 | M | 9 | 6 | 13 |
| C | 17 | 11 | M | 10 | 11 |
| D | 12 | 18 | 11 | M | 9 |
| E | 8 | 6 | 8 | 9 | M |

Рисунок 2 – Матрица смежности

Расстояние от города к этому же городу обозначено буквой М. Названия

строк соответствуют городам отправления, столбцов — городам назначения. Стоит отметить, что в рассматриваем примере расстояние между городами отличается в зависимости от направления (предположим, что имеются дороги с односторонним движением или цена билета на прямой и обратный рейс могут отличаться).

Шаг 1.

Находя минимальный элемент в каждой строке, получим константы приведения по строкам. К примеру, в первой строе он будет равен 7, во второй 6 и т.д. После чего выпишем данные значения в отдельный столбец $d_i$ (рисунок 2).

| Город | A | B | C | D | E | di |
|---|---|---|---|---|---|---|
| A | M | 18 | 15 | 7 | 11 | 7 |
| B | 20 | M | 9 | 6 | 13 | 6 |
| C | 17 | 11 | M | 10 | 11 | 10 |
| D | 12 | 18 | 11 | M | 9 | 9 |
| E | 8 | 6 | 8 | 9 | M | 6 |

Рисунок 3 – Добавление столбца минимальных элементов в строке

Шаг 2.

Следующим этапом проведем редукцию строк. Данное действие предполагает, что из каждого элемента в строке вычитается его соответствующее ей значение минимума.

$$C_{ij} = C_{ij} - d_i, \quad (1)$$

где $C_{ij}$ – элемент матрицы.

К примеру, в ячейке AB мы имеем значение 18. После редукции получаем значение 11(18-7). Производим аналогичную процедуру для всех строчек. Получаем обновленную таблицу (рисунок 4).

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 11 | 8 | 0 | 4 |
| B | 14 | M | 3 | 0 | 7 |
| C | 7 | 1 | M | 0 | 1 |
| D | 3 | 9 | 2 | M | 0 |
| E | 2 | 0 | 2 | 3 | M |

Рисунок 4 – Матрица смежностей после редукции

Таким образом, мы получили матрицу, в каждой строке которой будет хотя бы одна нулевая клетка.

Шаг 3.

Затем находим минимальные значения в каждом столбце и записываем их в отдельную строчку $d_j$ внизу матрицы. Например, в первом столбце минимальное значение 2, во втором – 0 и т.д.

Шаг 4.

Следующим шагом проведем редукцию столбцов. Теперь уже их каждого элемента конкретного столбца вычитаем его минимальное значение. В обновленной матрице в каждом столбце будет присутствовать хотя бы одно нулевая клетка. Полученная таблица представлена на рисунке 5.

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 11 | 6 | 0 | 4 |
| B | 12 | M | 1 | 0 | 7 |
| C | 5 | 1 | M | 0 | 1 |
| D | 1 | 9 | 0 | M | 0 |
| E | 0 | 0 | 0 | 3 | M |
| dj | 2 | 0 | 2 | 0 | 0 |

Рисунок 5 – Матрица смежностей после редукции столбцов

Шаг 5.

Далее определим корневую локальную нижнюю границу ($H_k$) стоимости(длины) маршрута. Для этого суммируем константы приведения $d_i$ и $d_j$ найденные ранее.

$$H_0 = \sum d_i + \sum d_j. \quad (2)$$

В нашем случае получаем

$$H_0 = (7 + 6 + 10 + 9 + 6) + (2 + 2) = 42. \quad (3)$$

Ход решения данной задачи будем представлять в виде дерева, в котором каждая вершина будет представлять собой решение по включению или невключению в итоговый маршрут тех или иных отрезков пути. Ребра же будут показывать ход решения и альтернативные ветви маршрута.

На данном этапе нам известен корень этого дерева, куда мы записываем значение локальной нижней границы $H_0$=42. Стоит также отметить, что значения на вершине каждого узла дерева являются минимальной длиной маршрута.

Шаг 6.

Затем вычисляем оценку $p_{ij}$ для каждой нулевой клетки. Данное значение находится как сумма минимального элемента в столбце и строке, в которой находится конкретная клетка. Стоит отметить, что нулевая клетка, для которой ищется значение не учитывается. Оценку для каждой клетки будем записывать в ней же в скобках.

Возьмем для примера клетку A-D. Минимальное значение в строке A – 4, а минимальное значение в столбце D – 0. Следовательно оценка для данной клетке будет вычисляться как: 4 + 0 = 4. Аналогичным образом вычислим оценки для оставшихся клеток. Результат представлен на рисунке 6.

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 11 | 6 | 0(4) | 4 |
| B | 12 | M | 1 | 0(1) | 7 |
| C | 5 | 1 | M | 0(1) | 1 |
| D | 1 | 9 | 0(0) | M | 0(1) |
| E | 0(1) | 0(1) | 0(1) | 3 | M |

Рисунок 6 – Найденные оценки

Шаг 7.

Следующим шагом выбираем клетку с максимальной оценкой. Оценка также показывает, насколько удлинится путь, если выбрать другую ячейку. Поэтому целесообразно будет выбрать клетку с наибольшей оценкой, дабы избежать удлинения маршрута. Если в матрице несколько максимальных оценок с одинаковым значением, то можно выбрать любую. В нашем же случае это клетка A-D.

Шаг 8.

На следующем этапе наше решение ветвится: в первом случае мы включаем в маршрут отрезок A-D, во втором случае – не включаем. Рассмотрим случай включение данного отрезка. Возьмем клетку A-D и полностью вычеркнем строку и столбец, на пересечении которых она находится, так как при включении в маршрут отрезка A-D, из города A в другие города мы уже не будем выезжать, а в городе D мы уже побывали. Следовательно данная строка и столбец не будут использоваться при дальнейшем решении. Так же, в клетку D-A выпишем букву M.

Таким образом мы провели редукцию матрицы и получили обновленную таблицу (рисунок 7).

| Город | A | B | C | E |
|---|---|---|---|---|
| B | 12 | M | 1 | 7 |
| C | 5 | 1 | M | 1 |
| D | M | 9 | 0 | 0 |
| E | 0 | 0 | 0 | M |

Рисунок 7 – Таблица после редукции

Далее необходимо вычислить лекальную нижнюю границу для данной ветви. Алгоритм ее нахождения был описан выше. Полученный результат представлен на рисунках 8-10.

| Город | A | B | C | E | di |
|---|---|---|---|---|---|
| B | 12 | M | 1 | 7 | 1 |
| C | 5 | 1 | M | 1 | 1 |
| D | M | 9 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | M | 0 |

Рисунок 8 – Добавление столбца минимальных элементов в строке

| Город | A | B | C | E |
|---|---|---|---|---|
| B | 11 | M | 0 | 6 |
| C | 4 | 0 | M | 0 |
| D | M | 9 | 0 | 0 |
| E | 0 | 0 | 0 | M |
| dj | 0 | 0 | 0 | 0 |

Рисунок 9 – Добавление строки минимальных элементов в столбце

| Город | A | B | C | E |
|---|---|---|---|---|
| B | 11 | M | 0 | 6 |
| C | 4 | 0 | M | 0 |
| D | M | 9 | 0 | 0 |
| E | 0 | 0 | 0 | M |

Рисунок 10 – Таблица после редукции

Далее вычисляется локальная нижняя граница для данной ветви по формуле:

$$H_k = H_{k-1} + \sum d_i + \sum d_j. \quad (4)$$

В нашем случае получаем:

$$H_k = 42 + (1 + 1) = 44. \quad (5)$$

Шаг 9.

Далее вычислим локальную нижнюю границу для ветви, в которой мы не включили путь A-D. Она будет вычисляться по формуле:

$$H_k^* = H_{k-1} + p_{ij}, \quad (6)$$

где $p_{ij}$- максимальная оценка.

$$H_k^* = 42 + 4 = 46. \quad (7)$$

Шаг 10.

Затем добавляем рассмотренные ветви в наше дерево. Путь, который не входит в маршрут обозначим символом «*». Так же укажем на узлах нижние локальные границы. Полученное дерево представлено на рисунке 11.

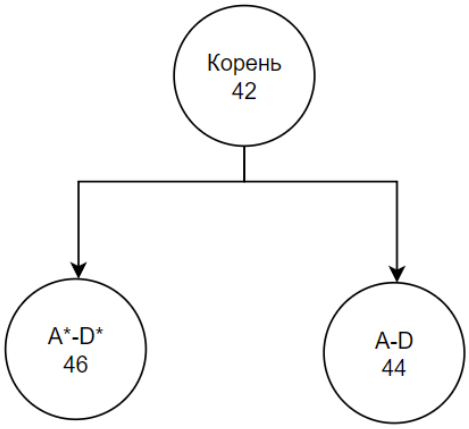


Рисунок 11 – Дерево решений

Далее среди всех неветвящихся вершин выбираем ту, которая имеет минимальную локальную границу. В нашем случае это ветвь с включением пути A-D. Следовательно дальнейшие решения начинаем с шага номер 6 – нахождения оценок для нулевых клеток. Также стоит отметить, что вновь находить минимумы по строкам и столбцам, а также проводить их редукцию не нужно — мы уже делали это ранее. Полученные оценки представлены на рисунке 12.

| Город | A | B | C | E |
|---|---|---|---|---|
| B | 11 | M | 0(6) | 6 |
| C | 4 | 0(0) | M | 0(0) |
| D | M | 9 | 0(0) | 0(0) |
| E | 0(4) | 0(0) | 0(0) | M |

Рисунок 12 - Найденные оценки

Максимально найденная оценка находится в клетке B-C. Далее решение опять расходится на две ветви: с включением отрезка B-C и его исключением из конечного маршрута. Рассмотрим ветвь, включающую данный отрезок. Сделаем редукцию матрицы и поставим M в ячейку C-B (рисунок 13).

| Город | A | B | E |
|---|---|---|---|
| C | 4 | M | 0 |
| D | M | 9 | 0 |
| E | 0 | 0 | M |

Рисунок 13 - Таблица после редукции

В каждой строке и в каждом столбце полученной таблицы минимальный элемент равен 0. Следовательно при редукции столбцов и строк из каждого элемента будет вычитаться 0, и таблица останется неизменной, поэтому опустим описание этого шага. Вычислим локальную нижнюю границу для данной ветви:

$$H_k = 44 + 0 = 44. \quad (8)$$

Также найдем локальную нижнюю границу для ветви, исключающей отрезок B-C:

$$H_k^* = 44 + 6 = 50. \quad (9)$$

Изобразим полученные ветви на дереве решений (рисунок 14).

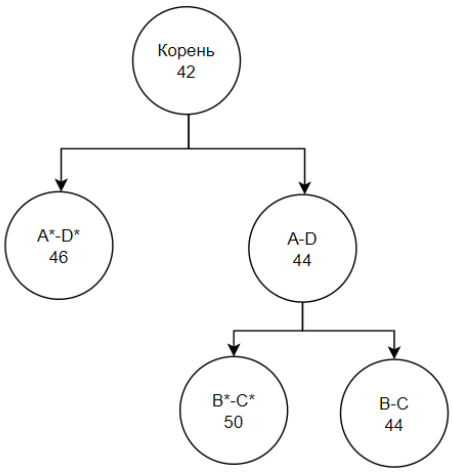


Рисунок 14 – Обновленное дерево решений

Как можно видеть, минимальный неветвящийся узел находится в ветке включения отрезка B-C. Поэтому повторяем данную процедуру еще раз. Ищем оценки каждой нулевой клетки (рисунок 15).

| Город | A | B | E |
|---|---|---|---|
| C | 4 | M | 0(4) |
| D | M | 9 | 0(9) |
| E | 0(4) | 0(9) | M |

Рисунок 15 – Найденные оценки

Наибольшее значение равное 9 находится в двух клетках: D-E и E-B. Выберем клетку D-E и проведем редукцию для ветви включающий этот отрезок (рисунок 16).

| Город | A | B |
|---|---|---|
| C | 4 | M |
| E | 0 | 0 |

Рисунок 16 – Таблица после редукции

После чего проведем редукцию строк и столбцов (рисунок 17).

| Город | A | B |
|---|---|---|
| C | 0 | M |
| E | 0 | 0 |

Рисунок 17 – Матрица смежностей после редукции строк

Вычислим локальную нижнюю границу для двух ветвей:

$$H_k = 44 + 4 = 48, \quad (10)$$

$$H_k^* = 44 + 9 = 53. \quad (11)$$

Добавим полученные ветви на дерево решений (рисунок 18).

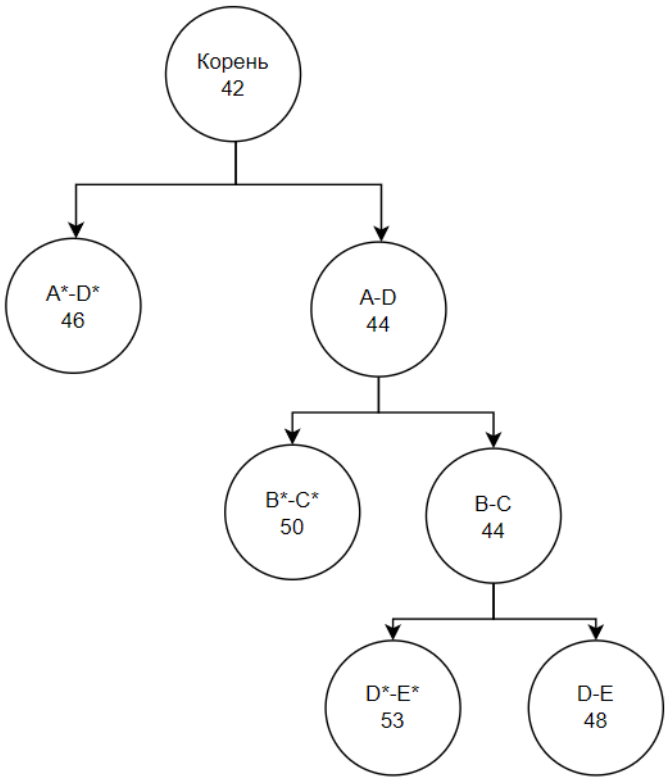


Рисунок 18 – Обновленное дерево решений

Теперь выбираем из всех еще не ветвившихся вершин ту, что имеет наименьшее значение локальной нижней границы. В данном случае, это вершина A*-D*. Для дальнейшего решения, нам необходимо вернуться к этапу, где мы рассматривали данную ветвь. В матрице смежности, в ячейке A-D поставим букву M. Так как этот путь мы исключили. Полученная матрица представлена на рисунке 19.

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 11 | 6 | M | 4 |
| B | 12 | M | 1 | 0 | 7 |
| C | 5 | 1 | M | 0 | 1 |
| D | 1 | 9 | 0 | M | 0 |
| E | 0 | 0 | 0 | 3 | M |

Рисунок 19 – Обновленная матрица смежностей

Далее проводим редукцию по строкам и столбцам, после чего вычисляем оценки для нулевых клеток (рисунок 20).

| Город | A | B | C | D | E |
|---|---|---|---|---|---|
| A | M | 7 | 2 | M | 0(2) |
| B | 12 | M | 1 | 0(1) | 7 |
| C | 5 | 1 | M | 0(1) | 1 |
| D | 1 | 9 | 0(0) | M | 0(0) |
| E | 0(1) | 0(1) | 0(0) | 3 | M |

Рисунок 20 – Найденные оценки

В данном случае максимальная оценка находится в клетке A-E. Проведем редукцию матрицы, вычеркивая соответствующие этой ячейке строку и столбец. Также ставим в клетке E-A букву M и сделаем редукцию столбцов и строк (рисунок 21).

| Город | A | B | C | D |
|---|---|---|---|---|
| B | 11 | M | 1 | 0 |
| C | 4 | 1 | M | 0 |
| D | 0 | 9 | 0 | M |
| E | M | 0 | 0 | 3 |

Рисунок 21 – Обновленная матрица смежностей

Вычислим локальные нижние границы для двух ветвей:

$$H_k = 46 + 1 = 47, \quad (12)$$

$$H_k^* = 46 + 2 = 48. \quad (13)$$

Таким методом обходим все вершины, пока не будет найдено оптимальное решение. В конечном итоге получаем дерево, изображенное на рисунке 22.

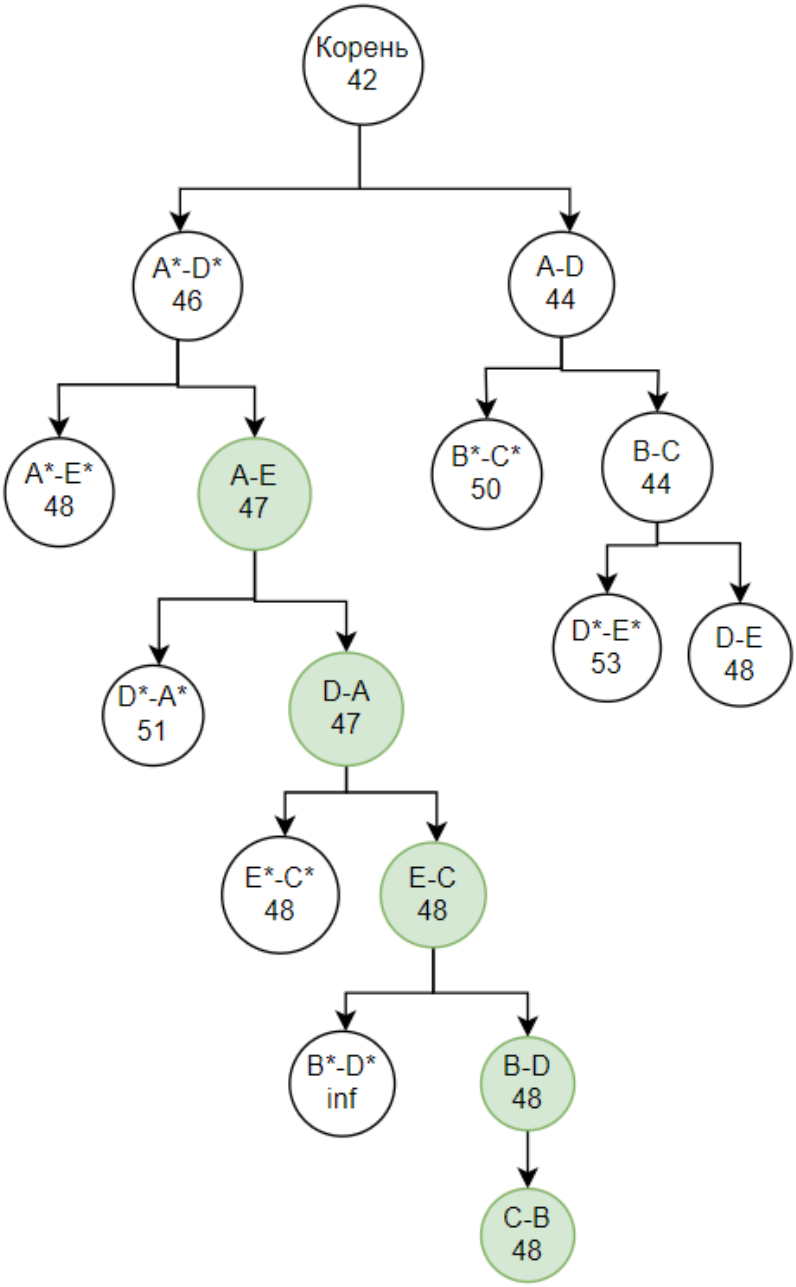


Рисунок 22 – Итоговое дерево решения

Стоит отметить, что помимо конечной вершины значение 48 также имеют три другие вершины: A*-E*, E*-C* И D-E, однако при дальнейшем их рассмотрении значение не может быть меньше 48, поэтому данные ветви мы не рассматриваем [8].

В ходе решения мы получили 5 отрезков маршрута: A-E, D-A, E-C, B-D, C-B. Далее соединим эти отрезки в один путь. Получаем: A->E->C->B->D->A. Таким образом общая длина пути равна 48.

Выводы по второму разделу:

В данном разделе были рассмотрены разные методы решения задачи коммивояжера и выбран метод ветвей и границ. После чего был решен пример задачи с использованием данного метода с описанием действий на каждом шаге.

# 3. Разработка программного обеспечения

## 3.1 Программная реализация алгоритма.

Реализовывать данный алгоритм будем на языке программирования Python, так как так как данный язык имеет простой синтаксис и большое количество библиотек, а также низкий порог вхождения [1],[12],[18]. Стоит отметить, что данный язык является кроссплатформенным и будет поддерживаться на большинстве устройств [13],[14]. Также для удобной работами с матрицами воспользуемся библиотекой NumPy. Для ее установки введем в командной строке следующий текст:

*pip install numpy*

Первым шагом подключим данную библиотеку в программе через функцию «import»

Затем реализуем функцию, которая будет проводить редукцию матрицы. Данная функция будет принимать на вход один параметр – исходную матрицу. Найдем минимальные элементы строка матрицы при помощи функции «min» из библиотеки NumPy [9]. В качестве параметров данной функции укажем матрицу, переданную в функцию, изначальное значение минимума (укажем бесконечность) и ось по который будем искать минимумы. Так как нам нужно найти минимальные элементы по строкам, то зададим этот параметр равным 1.

Далее с помощью цикла for пройдемся по строкам исходной матрицы и будем вычитать значения минимума. Длину пути из города в этот же самый город будем обозначать как бесконечность, поэтому чтобы избежать ошибки при вычитании минимального элемента, добавим условие, чтоб исходный элемент матрицы был не равен бесконечности. Аналогичным образом сделаем редукцию по столбцам, с тем лишь отличием, что в функции «min» значение оси зададим равным 0. После выполнения функция будет возвращать

обновленную матрицу, а также сумму минимумов по столбцам и строкам Полученная функция представлена на рисунке 23.

```
import numpy as np


def reduce_matrix(matrix):
    # Находим минимальные значения для строк
    row_min = np.min(matrix, where=~np.isinf(matrix), initial=np.inf, axis=1)
    for i in range(len(matrix)):
        if row_min[i] != np.inf:
            matrix[i] -= row_min[i]

    # Находим минимальные значения для столбцов
    col_min = np.min(matrix, where=~np.isinf(matrix), initial=np.inf, axis=0)
    for i in range(len(matrix)):
        if col_min[i] != np.inf:
            matrix[:, i] -= col_min[i]

    return matrix, np.sum(row_min[row_min != np.inf]) + np.sum(col_min[col_min != np.inf])
```

Рисунок 23 – Функция редукции матрицы

Далее опишем класс, реализующий узел дерева решений. Сперва реализуем конструктор для данного класса. Если при создании узла не был указан путь, то по умолчанию он будет равным нулю. Также будем инициализировать переменные, обозначающие обновленную матрицу и нижнюю границу с помощью ранее описанной функции «reduce_matrix».

Также в этом классе реализуем функцию «extend», которая будет создавать два дочерних узла. Каждый дочерний узел представляет собой возможное продолжение пути. В переменную «i» запишем индекс последнего города в текущем пути. Далее в цикле for перебираются все города из матрицы стоимости по переменной «j». Внутри цикла проверяем не равен ли путь из города i в город j бесконечности. Если условие выполнилось, то создаем копию матрицы с помощью функции «copy» и в столбце j и строке i ставим значение равное бесконечности, чтобы исключить город j из дальнейшего рассмотрения. Затем создадим новый узел, передавая скопированную матрицу, а также текущий путь с добавлением города j. Также увеличиваем

оценку «bound» на стоимость перехода от города i к городу j в редуцированной матрице. В конце функции возвращаем список созданных дочерних узлов. Реализованная класс представлен на рисунке 24.

```
36  class Node:
37      def __init__(self, matrix, path=None):
38          if path is None:
39              path = [0]
40
41          self.matrix, self.bound = reduce_matrix(matrix)
42          self.path = path
43
44      def expand(self):
45          children = []
46          i = self.path[-1]
47
48          for j in range(len(self.matrix)):
49              if self.matrix[i][j] != np.inf:
50                  matrix = np.copy(self.matrix)
51                  matrix[:, j] = np.inf
52                  matrix[i] = np.inf
53                  matrix[j][i] = np.inf
54
55                  child = Node(matrix, self.path + [j])
56                  child.bound += self.matrix[i][j] + self.bound
57
58                  children.append(child)
59
60          return children
```

Рисунок 24 – Класс описывающий узел дерева

Следующим этапом реализуем сам алгоритм метода ветвей и границ. Создадим функцию «tsp» (Travelling Salesman Problem), принимающую на вход матрицу стоимости и родительские узлы. Если его нет, то создается корневой узел. Далее создадим список «nodes», в которые присвоим родительские узлы. Затем в цикле while извлекаем первый узел из списка nodes с помощью функции «nodes.pop(0)» и присваиваем его переменной «node». После чего проверяем, равна ли длина пути в узле «node» количеству городов

в матрице. Если условие выполнилось, то добавляем начальный город в конец пути «path» и возвращается его, а также и оценку границы «bound».

Если уже условие не выполнилось, то в цикле for создаем дочерние узлы и добавляем их в список «nodes». После завершения цикла for, список «nodes» сортируется по оценке границы каждого узла с помощью функции «sort». Это делается для того, чтобы узлы с более низкой оценкой границы были обработаны раньше. Функция, реализующая метод ветвей и границ представлена на рисунке 25.

```
def tsp(matrix, parent=None):
    nodes = []
    if parent is None:
        parent = Node(matrix)

    nodes.append(parent)

    while nodes:
        node = nodes.pop(0)
        if len(node.path) == len(matrix):
            node.path.append(node.path[0])
            return node.path, node.bound

        for i in node.expand():
            nodes.append(i)

        nodes.sort(key=lambda x: x.bound)
```

Рисунок 25 – Функция метода ветвей и границ.

Пример использования данной функции изображен на рисунке 26.

```
path, cost = tsp(graph)
print(f"Путь: {path}")
print(f"Стоимость: {cost}")
```

Рисунок 26 – Запуск метода ветвей и границ

### 3.2 Проведение вычислительных экспериментов

Следующим шагом протестируем разработанное программное обеспечение. Для первого теста возьмем произвольный граф из 5 вершин (рисунок 27).

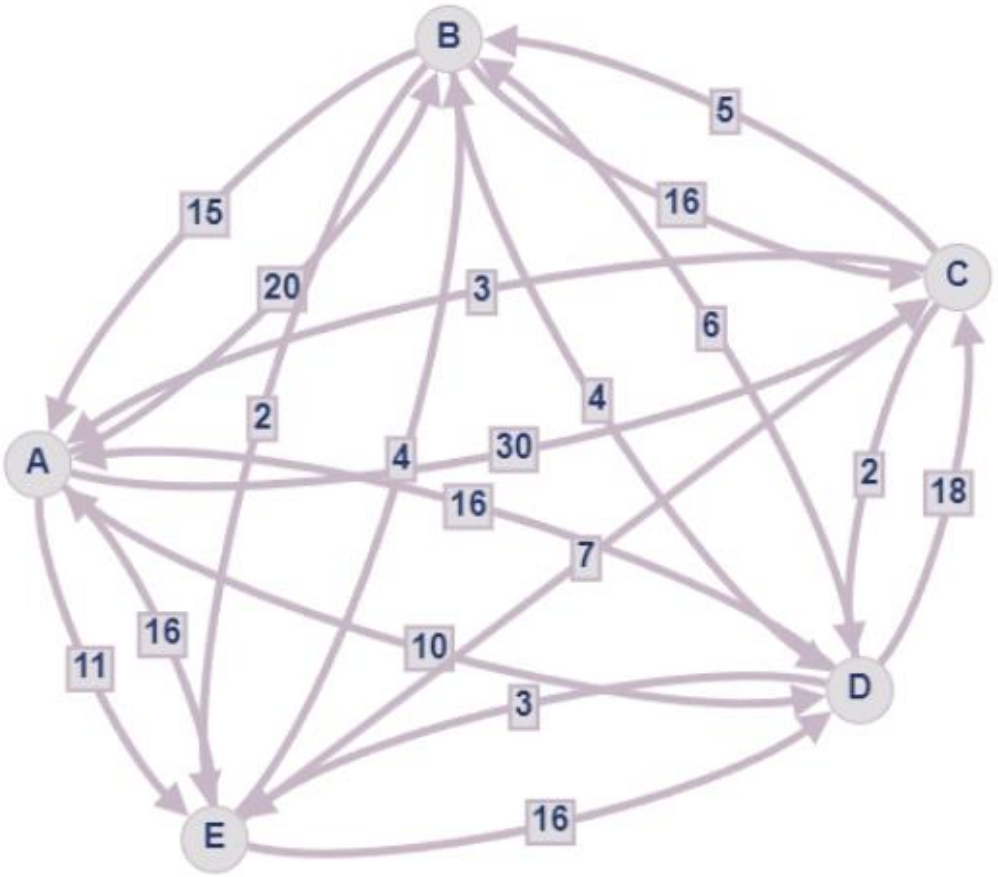


Рисунок 27 – Граф для тестирования работы программы

Далее запишем этот граф в файле с исходным кодом. Для инициализации графа будем использовать структуру двумерного массива из библиотеки NumPy. Так как на главной диагонали матрицы будут располагаться пути из одного города в этот же город, то в данных элементах

зададим значение «np.inf», что равно бесконечности. После инициализации двумерного массива передадим его функцию «tsp». Возвращаемое значение этой функции запишем в переменные «path» и «cost». После чего с помощью функции «print» выведем эти значения в консоль (рисунок 28).

```
64  graph = np.array([[np.inf, 20, 30, 10, 11],
65                    [15, np.inf, 16, 4, 2],
66                    [3, 5, np.inf, 2, 4],
67                    [19, 6, 18, np.inf, 3],
68                    [16, 4, 7, 16, np.inf]])
69
70
71  path, cost = tsp(graph)
72  print(f"Путь: {path}")
73  print(f"Стоимость: {cost}")
```

Рисунок 28 – Использование функции «tsp»

Результат выполнения программы показан на рисунке 29.

```
Путь: [0, 3, 1, 4, 2, 0]
Стоимость: 28.0
```

Рисунок 29 – Результат

Проврем полученную стоимость по найденному маршруту. Исходя из заданного графа она будет равна:

$$S = 10 + 6 + 2 + 7 + 3 = 28. \quad (14)$$

Как можно заметить, программа работает корректно. Однако на практике данная задача зачастую решается с количеством вершин, во много

превышающем пяти, поэтому целесообразно протестировать разработанное программное обеспечение с другим набором входных данных. Чтобы не вводить вручную каждый путь в матрице смежностей при тестировании реализуем функцию «create_graph», которая будет задавать граф случайным образом. В качестве параметров данной функции укажем:

– n - Размер генерируемого графа,
– min_val - Минимальное возможное значение стоимости пути,
– man_val - Максимальное значение стоимости пути.

Первым этапом создадим пустой массив размером $n_x n$ с помощью функции «np.empty» [20]. Далее создадим два вложенных цикла, в котором будем перебирать все элементы массива. Если номер строки и номер столбца на данной итерации не совпадают, то значение этого элемента будет вычислять с помощью функции «np.random.randint» [23], в которую передадим минимальное и максимальное значение при генерации числа. Если же в цикле встретился элемент на главной диагонали, то его значение указываем как «np.inf». После цикла будет возвращать полученный массив. Реализация данный функции изображена на рисунке 30.

```
def create_graph(n, min_val, max_val):
    graph = np.empty((n,n))
    for i in range(0,n):
        for j in range(0,n):
            if (i!=j):
                graph[i][j]=np.random.randint(min_val, max_val)
            else:
                graph[i][j]=np.inf

    return graph
```

Рисунок 30 –Генерация матрицы смежности

Попробуем сгенерировать небольшой граф и выведем его в консоль,

чтобы проверить корректность работы данной функции (рисунок 31).

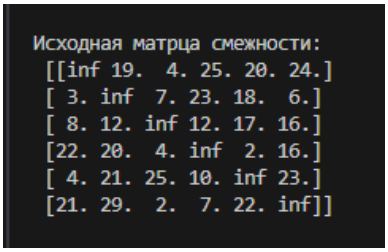


Рисунок 31 –Сгенерированная матрица

Теперь попробуем запустить нашу программу для графа с 18 вершинами. Также, чтобы убедиться в правильности работы нашей программы, выведем сгенерированную матрицу в консоль.

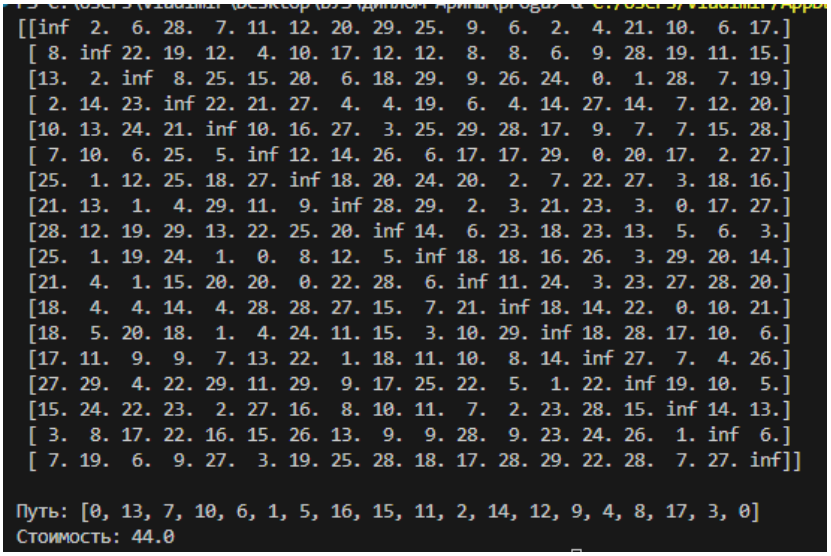


Рисунок 31 – Результат работы программы

Теперь посчитаем стоимость полученного пути и сравним ее со значением, которое посчитала программа:

$$S = 4 + 1 + 2 + 0 + 1 + 4 + 2 + 1 + 2 + 4 + 1 + 1 + 3 + 1 + 3 + 3 + 9 + 2 = 44. \quad (15)$$

Как мы можем убедиться из примера выше, программа работает корректно с большими графами.

### 3.3 Корректировка разработанного программного обеспечения.

Разработанная программа работает корректно, но результат ее работы выводится в консоль. Данное решение не слишком практично по двум причинам:

Во-первых, найденное решение никуда не сохраняется и при завершении работы программы посмотреть его результат не представляется возможным.

Во-вторых, полученный результат не получится отправить другому человеку по современным средствам коммуникациям, такими как электронная почта и мессенджеры. Из-за чего, передавать найденное решения представляется не очень удобным.

Для исправления данных недостатков целесообразно будет сохранять полученный результат в отдельный файл, который можно будет с легкостью отправить другим работникам предприятия.

Чтобы реализовать данный функционал будем использовать функцию «print» с параметром «file», чтобы указать файловый объект, в который будем записывать полученные данные [25]. Название выходного файла будет «output», а его формат «txt». Также для корректного отображения русского языка укажем кодировку файла UTF-8 [16] (рисунок 32).

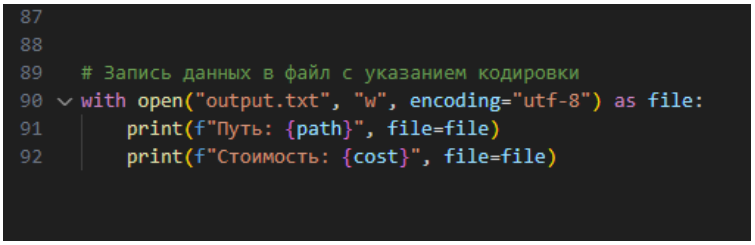


Рисунок 32 – Запись результатов в файл

Теперь при завершении работы программы будет создаваться текстовый файл с полученными результатами. Стоит также отметить, что результирующий файл будет сохраняться в той же директории, что и файл с исходным кодом программы. Полученный файл представлен на рисунке 33.

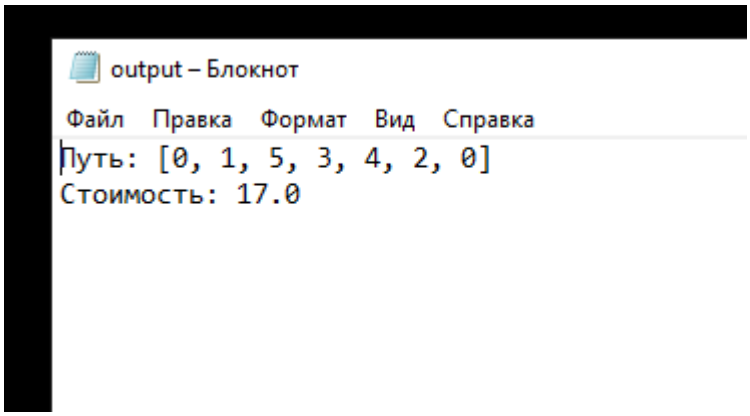


Рисунок 33 –Полученный файл

Полный исходный код всей программы представлен в приложении А.

Выводы по третьему разделу:

В данном разделе было проведено тестирование разработанного программного обеспечения на разном наборе данных. Также была добавлена функция записи результатов в файл, что позволяет сохранять полученный результат.

## Заключение

В процессе выполнения данной выпускной работы было проведено исследование, в ходе которого было выявлено, что в нынешнее время, когда число оказываемых услуг доставки и обслуживание на дому растет высокими темпами, остро стоит вопрос о выборе наилучшего маршрута для коммерческих организаций, для максимального сокращения материальных и временных расходов. Также высокая конкуренция заставляет максимально оптимизировать имеющиеся временные и чусовские ресурсы.

После постановки проблемы, были рассмотрены четыре алгоритма для решения задачи о поиске кратчайшего пути в графе: жадный алгоритм, метод ветвей и границ, метод динамического программирования, метод возврата к исходной вершине. Далее был проведен анализ и выбран наилучший метод исходя из особенности задачи, а именно метод ветвей и границ. После чего был рассмотрим алгоритм этого метода и решена задача коммивояжера с помощью него.

Следующим этапом был выбран язык программирования Python, а затем разработано программное обеспечение, реализующее рассмотренный до этого алгоритм с использованием библиотеки NumPy. Данная программа принимает на вход матрицу смежностей, и возвращает наилучшей маршрут и его стоимость.

Далее разработанная программа была протестирована как на маленьком, и так и на большом наборе данных. Также для более удобного применения была реализована функция записи полученных результатов в текстовый файл.

## Список используемой литературы

1. Васильев Ф. П. "Python. Полное руководство" Питер, 2018

2. Васильев Ф. П. "Дискретная оптимизация: Методы и алгоритмы." БХВ-Петербург, 2002.

3. Вихрев В. П., Григорьев А. Л. "Оптимальное управление и дискретная математика: Учебное пособие." Лань, 2010.

4. Гаврилов А. П., Диковский А. В. "Оптимизация коммивояжера." Физматлит, 2011. URL: https://studfile.net/preview/16462302/page:37/

5. Гальперин Е. О., Гудков А. А. "Метод ветвей и границ в задачах и упражнениях" МЦНМО, 2012. URL: https://studref.com/680007/ekonomika/metod_vetvey_granits

6. Зелев Ю. "Практическое руководство по методу ветвей и границ" Бином, 2019. URL: https://studylib.ru/doc/2303909/metod-vetvej-i-granic

7. Корнев В. Г. "Комбинаторные задачи дискретной математики." КомКнига, 2006. URL: https://reader.lanbook.com/book/214982

8. Краснов Е. А. "Метод ветвей и границ: теория и приложения" КомКнига, 2016. URL: https://studizba.com/files/show/doc/203476-1-metod-vetvey-i-granic.html

9. Ломов С. Н. "Python: программирование игр и головоломок" Питер, 2015

10. Мишустин С. В. "Метод ветвей и границ в теории и задачах" Феникс, 2007. URL: https://galyautdinov.ru/post/zadacha-kommivoyazhera

11. Пучков Л. А. "Метод ветвей и границ. Оптимизация параметров и задачи с ограничениями" Лань, 2003.

12. Романовский И. М., Чумаченко Е. Н. "Python. Карманный справочник" БХВ-Петербург, 2019

13. Сулимов А. В., Добровольский С. И. "Python для сложных задач. Наука о данных и машинное обучение" ДМК Пресс, 2020

14. Шалыто А. А. "Программирование на Python 3" БХВ-Петербург, 2017

15. Черноусько, Ф. Л. "Комбинаторная оптимизация: алгоритмы и аппаратные средства." БИНОМ. Лаборатория знаний, 2016.

16. Sweigart Al "Automate the Boring Stuff with Python" No Starch Press, 2015 URL: https://studylib.ru/doc/6364397/svejgart-e-l.-avtomatizaciya-rutinnyh-zadach-s-pomoshh._yu-pytho...

17. Applegate, D. L., et al. "The Traveling Salesman Problem: A Computational Study." Princeton University Press, 2006 URL: https://archive.org/details/travelingsalesma0000unse

18. Eric Matthes "Python Crash Course" No Starch Press, 2015. URL: https://howdyho.net/static/uploads/files/E_Metiz_-_Izuchaem_Python_Programmirovanie_igr_vizualizatsia_dannykh_veb-prilozhenia_-_2017.pdf

19. Gutin G., and Punnen A. "The Traveling Salesman Problem and Its Variations." Springer, 2006. URL: https://link.springer.com/book/10.1007/b101971

20. VanderPlas J. "Python Data Science Handbook" O'Reilly Media, 2016. https://inprogrammer.com/wp-content/uploads/2022/01/Jake-VanderPlas-Python-Data-Science-Handbook_-Essential-Tools-for-Working-with-Data-2016-OReilly-Media-libgen.lc_.pdf

21. Lawler E. L., et al. "Combinatorial Optimization: Networks and Matroids." Courier Corporation, 2013. URL:

http://inis.jinr.ru/sl/vol1/CMC/Lawler,_Combinatorial_Optimization,_Networks%26Matroids,1976.pdf

22. Lawler E. L., et al. "The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization." Wiley, 1985. URL: https://archive.org/details/travelingsalesma00lawl

23. Mark L. "Learning Python" O'Reilly Media, 2013. URL: https://archive.org/details/learning-python-by-mark-lutz

24. Papadimitriou C. H., and Steiglitz K. "Combinatorial Optimization: Algorithms and Complexity." Dover Publications, 1998. URL: https://archive.org/details/combinatorialopt0000papa

25. McKinney Wes "Python for Data Analysis" O'Reilly Media, 2017. URL: https://inprogrammer.com/wp-content/uploads/2023/02/Wes-McKinney-Python-for-Data-Analysis_-Data-Wrangling-with-Pandas-NumPy-and-IPython.pdf

=

Приложение А

**Исходный код программы**

```python
import numpy as np

def reduce_matrix(matrix):

    # Находим минимальные значения для строк
    row_min = np.min(matrix, where=~np.isinf(matrix), initial=np.inf, axis=1)
    for i in range(len(matrix)):
        if row_min[i] != np.inf:
            matrix[i] -= row_min[i]

    # Находим минимальные значения для столбцов
    col_min = np.min(matrix, where=~np.isinf(matrix), initial=np.inf, axis=0)
    for i in range(len(matrix)):
        if col_min[i] != np.inf:
            matrix[:, i] -= col_min[i]

    return matrix, np.sum(row_min[row_min != np.inf]) + np.sum(col_min[col_min != np.inf])

def tsp(matrix, parent=None):
    nodes = []
    if parent is None:
        parent = Node(matrix)

    nodes.append(parent)

    while nodes:
        node = nodes.pop(0)
        if len(node.path) == len(matrix):
            node.path.append(node.path[0])
            return node.path, node.bound

        for i in node.expand():
            nodes.append(i)

        nodes.sort(key=lambda x: x.bound)

class Node:
    def __init__(self, matrix, path=None):
        if path is None:
            path = [0]

        self.matrix, self.bound = reduce_matrix(matrix)
        self.path = path

    def expand(self):
        children = []
        i = self.path[-1]
```

```
48
49          for j in range(len(self.matrix)):
50              if self.matrix[i][j] != np.inf:
51                  matrix = np.copy(self.matrix)
52                  matrix[:, j] = np.inf
53                  matrix[i] = np.inf
54                  matrix[j][i] = np.inf
55
56                  child = Node(matrix, self.path + [j])
57                  child.bound += self.matrix[i][j] + self.bound
58
59                  children.append(child)
60
61          return children
```

```
70
71  def create_graph(n, min_val, max_val):
72      graph = np.empty((n,n))
73      for i in range(0,n):
74          for j in range(0,n):
75              if (i!=j):
76                  graph[i][j]=np.random.randint(min_val, max_val)
77              else:
78                  graph[i][j]=np.inf
79
80      return graph
81
82
83  graph = create_graph(6, 0, 30)
84
85  print('\nИсходная матрца смежности:\n', graph)
86  path, cost = tsp(graph)
87
88
89  # Запись данных в файл с указанием кодировки
90  with open("output.txt", "w", encoding="utf-8") as file:
91      print(f"Путь: {path}", file=file)
92      print(f"Стоимость: {cost}", file=file)
```