**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №6**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Решение задачи о коммивояжере с помощью метода ближайшего соседа**»

**Студент гр. 23Б15-пу**

**Бек В.А.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc192428322)
2. [Описание задачи (формализация задачи) 4](#_Toc192428323)
3. [Теоретическая часть 5](#_Toc192428324)

[**Задача коммивояжера:** 5](#_Toc192428325)

[**Метод ближайшего соседа:** 5](#_Toc192428326)

[**Оценка эффективности метода:** 5](#_Toc192428327)

[**Модификация метода ближайшего соседа:** 6](#_Toc192428328)

1. [Основные шаги программы 7](#_Toc192428329)
2. [Блок схема программы 9](#_Toc192428330)
3. [Описание программы 11](#_Toc192428331)
4. [Рекомендации пользователя 12](#_Toc192428332)
5. [Рекомендации программиста 13](#_Toc192428333)
6. [Контрольный пример 14](#_Toc192428334)
7. [Анализ 16](#_Toc192428335)
8. [Вывод 18](#_Toc192428336)
9. [Источники 19](#_Toc192428337)

# Цель работы

Цель данной работы — изучение и реализация решения задачи о коммивояжере с использованием метода ближайшего соседа. Задача заключается в нахождении кратчайшего гамильтонова цикла во взвешенном орграфе. Разработанная программа будет протестирована на заданном взвешенном орграфе.

# Описание задачи (формализация задачи)

В данной лабораторной работе необходимо исследовать эффективность метода ближайшего соседа при решении задачи коммивояжера. Задача формулируется следующим образом:

Дан взвешенный ориентированный граф **G=(V,E)**, где **V** — множество вершин (городов), а **E** — множество рёбер, каждое из которых имеет вес, соответствующий расстоянию или стоимости перемещения между вершинами. Требуется найти гамильтонов цикл минимальной длины, то есть такой маршрут, который проходит через каждую вершину ровно один раз и возвращается в исходную точку, минимизируя суммарную длину пути.

**В рамках работы необходимо**:

* Разработать программную реализацию алгоритма ближайшего соседа.
* Визуализировать процесс построения маршрута.
* Провести экспериментальный анализ качества решения, сравнив его с оптимальным маршрутом.

В качестве модификации алгоритм ближайшего соседа стартует из произвольной вершины и на каждом шаге выбирает ближайшую непосещённую вершину, пока не будут посещены все вершины, после чего маршрут замыкается. Этот метод является жадным алгоритмом и не гарантирует нахождение оптимального решения, но обеспечивает быстрое приближённое решение с невысокими вычислительными затратами.

# Теоретическая часть

### **Задача коммивояжера:**

Задача коммивояжёра (**Traveling Salesman Problem** / **TSP**) является одной из классических задач комбинаторной оптимизации. Она формулируется следующим образом: имеется множество городов и известны расстояния между каждой парой городов. Требуется найти кратчайший маршрут, который проходит через каждый город ровно один раз и возвращается в начальную точку.

Задача коммивояжёра относится к классу NP-трудных задач, и при увеличении числа городов найти точное решение становится практически невозможно за разумное время. Поэтому на практике используют специальные приближённые методы и эвристики, которые позволяют быстро находить хорошие, но не обязательно идеальные маршруты.

### **Метод ближайшего соседа:**

Метод ближайшего соседа (**Nearest Neighbor Algorithm**) — это жадный алгоритм, который строит приближённое решение задачи коммивояжёра следующим образом:

1. Выбирается начальная вершина.
2. Из текущей вершины выбирается ближайшая непосещённая вершина, и совершается переход в неё.
3. Процесс повторяется до тех пор, пока не будут посещены все вершины.
4. Завершающий шаг — возвращение в исходную вершину, чтобы образовать гамильтонов цикл.

### **Оценка эффективности метода:**

Метод ближайшего соседа является быстрым и простым, но он не гарантирует нахождения оптимального маршрута. В худшем случае он может давать маршрут, значительно длиннее оптимального.

### **Модификация метода ближайшего соседа:**

В классическом методе ближайшего соседа выбор начальной вершины может сильно влиять на качество найденного маршрута. Для улучшения качества решения в данной лабораторной работе используется модификация метода, при которой начальная вершина выбирается не фиксировано, а поочерёдно стартуем из каждой вершины графа.

Алгоритм модифицированного метода ближайшего соседа:

1. Для каждой вершины графа выполняется метод ближайшего соседа, начиная с этой вершины.
2. Для каждого маршрута вычисляется его общая длина.
3. Среди всех найденных маршрутов выбирается тот, у которого длина минимальна.

Данная модификация позволяет учесть влияние выбора начальной вершины на итоговый маршрут и, как правило, даёт более качественные приближённые решения по сравнению с классическим методом. Однако увеличение количества запусков метода увеличивает общее время работы алгоритма.

# Основные шаги программы

1. **Создание графического интерфейса**
   * Используется библиотека **Tkinter**.
   * Основное окно разделено на два блока: левый (для графического ввода/вывода) и правый (для отображения рёбер и управления).
   * В левом блоке два холста: для входного графа и для вывода решения.
   * В правом блоке — таблица рёбер, кнопки управления и область вывода результата.
2. **Добавление вершин**
   * Левый клик по холсту добавляет вершину, если рядом нет других.
3. **Добавление рёбер**
   * Правый клик по вершине выбирает её как начальную.
   * Следующий правый клик на другой вершине создаёт между ними ребро с заданным пользователем весом.
   * Если ребро уже существует, можно изменить его вес повторно кликая на начальную и конечную вершины
4. **Решение задачи коммивояжера**
   * На кнопку “Рассчитать” запускается процесс решения задачи коммивояжера
   * Создаётся матрица смежности на основе рёбер.
   * Для каждого возможного начального города ищется приближённый путь методом жадного алгоритма.
   * Выбирается кратчайший найденный маршрут.
5. **Отображение результата**
   * Если путь найден, он выводится в текстовом виде и рисуется на выходном холсте.
   * Если решения нет, отображается соответствующее сообщение.
6. **Отмена действия и Очистка**

* **Кнопка "Отменить"** отменяет последнее действие, удаляя последнюю добавленную вершину или ребро.
* **Кнопка "Очистить"** очищает все данные и визуальные элементы, сбрасывая все вершины, рёбра и результаты.

# Блок схема программы

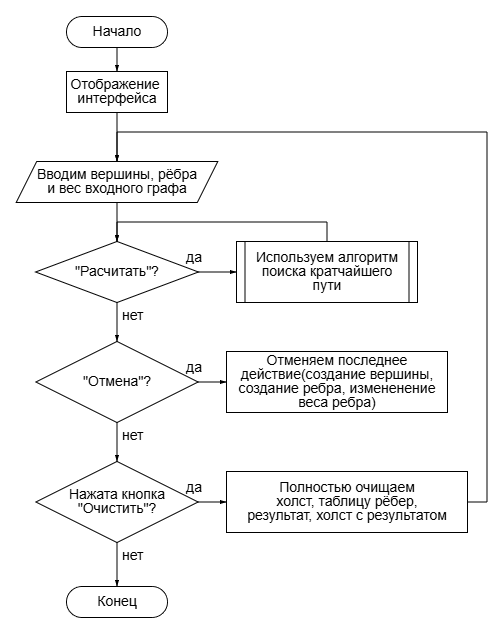


Рис. 3 Блок-схема основной программы

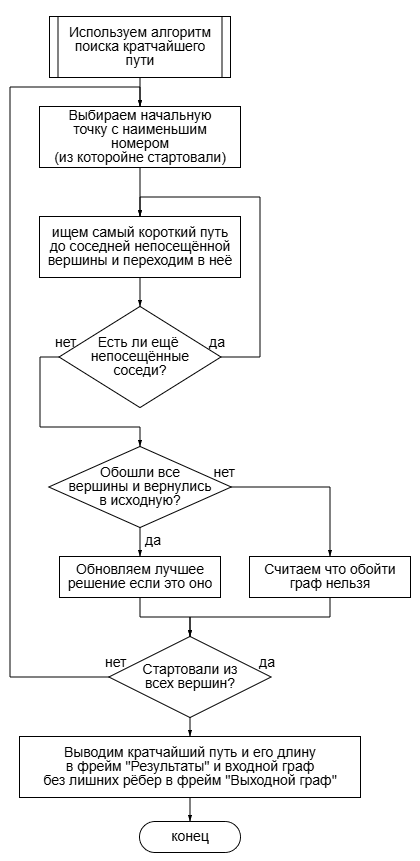


Рис. 4 Блок-схема подпрограммы

# Описание программы

Программная реализация задачи коммивояжёра написана на языке **Python 3.12.7** с использованием библиотек **tkinter, ttk, math**. Программа представляет собой графический интерфейс, который позволяет пользователю создавать и редактировать граф, состоящий из вершин и рёбер. На основе введённых данных программа решает задачу коммивояжёра, находя кратчайший путь, который проходит через все вершины. В графическом интерфейсе пользователи могут добавлять вершины, соединять их рёбрами и указывать веса рёбер. Также предусмотрены возможности для отмены последнего действия и очистки всех данных. Программа отображает решение задачи коммивояжёра, выводя оптимальный маршрут и его длину.

Таблица 1. nearest\_neighbor\_algorithm.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| add\_vertex(event) | Добавляет вершину на канвас в месте клика мышью. | None |
| select\_vertex\_for\_edge(event) | Выбирает вершину для соединения с другой или изменяет вес существующего рёбра. | None |
| draw\_directed\_edge(start\_vertex, end\_vertex) | |  | | --- | |  |  |  | | --- | | Рисует направленное ребро между двумя вершинами. | | ID нарисованного ребра |
| draw\_shortest\_path(path, adj\_matrix) | |  | | --- | | Рисует кратчайший путь, найденный для задачи коммивояжёра. |  |  | | --- | |  | | None |
| calculate\_tsp() | |  | | --- | |  |  |  | | --- | | Рассчитывает решение задачи коммивояжёра, используя алгоритм поиска кратчайшего пути. | | None |
| undo\_last\_action() | Отменяет последнее действие пользователя (удаление вершины или рёбра). | None |
| clear\_all() | |  | | --- | |  |  |  | | --- | | Очищает все данные (все вершины и рёбра), сбрасывает интерфейс. | | None |

# Рекомендации пользователя

Для корректного выполнения программы следуйте приведённым шагам:

* **Добавление вершин**:

Для добавления вершины на граф, кликните левой кнопкой мыши на поле "Входной граф". Каждое нажатие создаст новую вершину с уникальным идентификатором.

* **Создание рёбер**:

Для создания рёбер между вершинами, кликните правой кнопкой мыши по одной вершине, чтобы выбрать её, затем кликните правой кнопкой мыши по другой вершине для создания ребра между ними. После этого вас попросят ввести вес рёбра.

* **Просмотр и изменение весов рёбер**:

Если ребро между двумя вершинами уже существует, вы можете изменить его вес. Программа предложит ввести новый вес, если вы выбрали одну из вершин, между которыми уже есть ребро.

* **Отмена последнего действия**:

Для отмены последнего действия (добавление вершины или рёбра) используйте кнопку "Отменить". Это позволит вернуть граф в состояние до последнего изменения.

* **Очистка графа**:

Для очистки всего графа (удаления всех вершин и рёбер) нажмите кнопку "Очистить". После этого граф будет полностью очищен, и вы можете начать с нуля.

* **Расчёт оптимального пути**:

После добавления всех вершин и рёбер, нажмите кнопку "Рассчитать". Программа найдет оптимальный путь для задачи коммивояжера и отобразит результат на поле "Выходной граф". На поле также будет показан вес маршрута, а результаты расчета будут отображены в блоке "Результаты". Здесь будет показан оптимальный путь и общий вес маршрута.

# Рекомендации программиста

Для корректного функционирования программы рекомендуется выполнить следующие действия:

1. **Установите необходимые библиотеки**:
   * Убедитесь, что у вас установлены библиотеки **tkinter, math, ttk** и **tkinter.simpledialog**. **thinker** идет в стандартной поставке Python, но если она отсутствует, вы можете установить ее через пакетный менеджер вашей операционной системы.
2. **Проверьте версию Python**:
   * Рекомендуется использовать **Python** версии **3.7** или выше, чтобы избежать возможных проблем с совместимостью библиотек и функциональностью **tkinter**.
3. **Проверка функций и интерфейса**:
   * Убедитесь, что все элементы интерфейса (кнопки, текстовые поля и таблицы) работают корректно. Попробуйте вводить различные значения и проверить, что результаты рассчитываются и отображаются правильно.

**Код программы:**

**<https://github.com/Kliooo/Algorithms-and-data-structures>**

# Контрольный пример

* **Запуск программы:** Для запуска программы используйте файл nearest\_neighbor\_algorithm.py. Программа откроет графический интерфейс (Рис. 4).

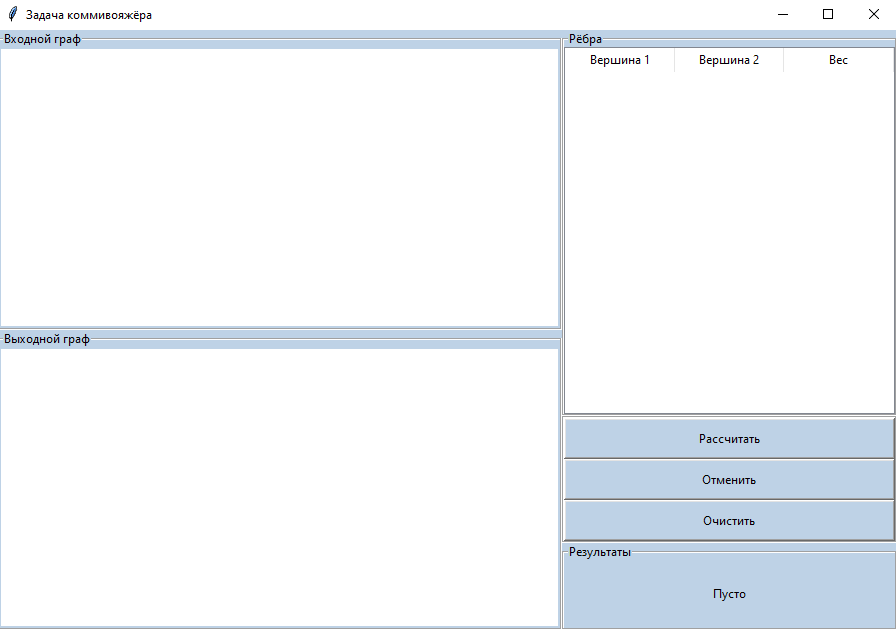


Рис. 3 Графический интерфейс программы

* **Инициализация частиц:** В фрейме “Входной граф” необходимо при помощи нажатий на правую и левую кнопки мыши изобразить нужный граф. (Рис. 4)

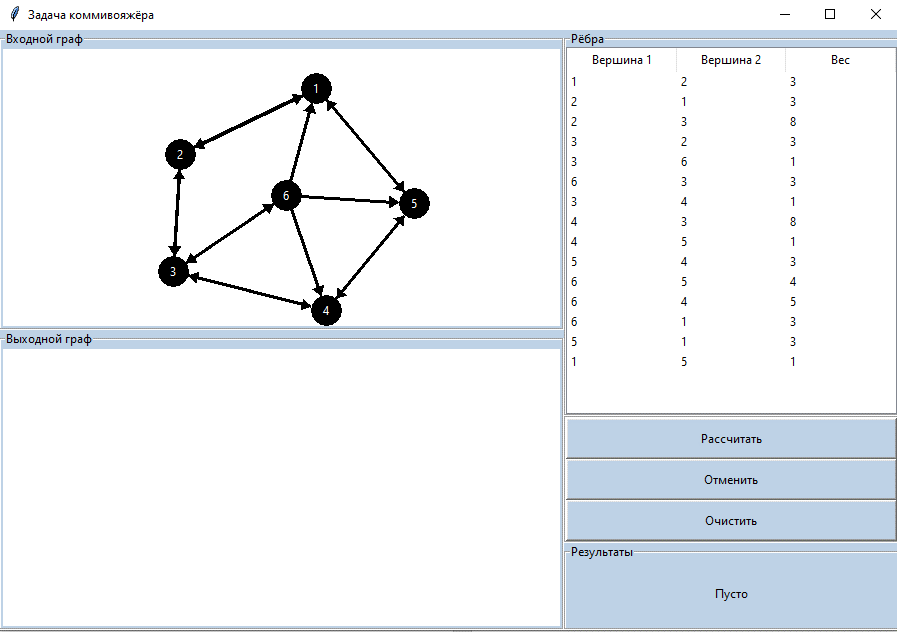


Рис. 4 Ввод графа

* **Запуск алгоритма ближайшего соседа:** При нажатии на кнопку “Рассчитать” произойдёт запуск алгоритма и мы получим какой-то результат в фрейме “Результаты” и его изображение в фрейме “Выходной граф”. (Рис. 5 / Рис. 6)

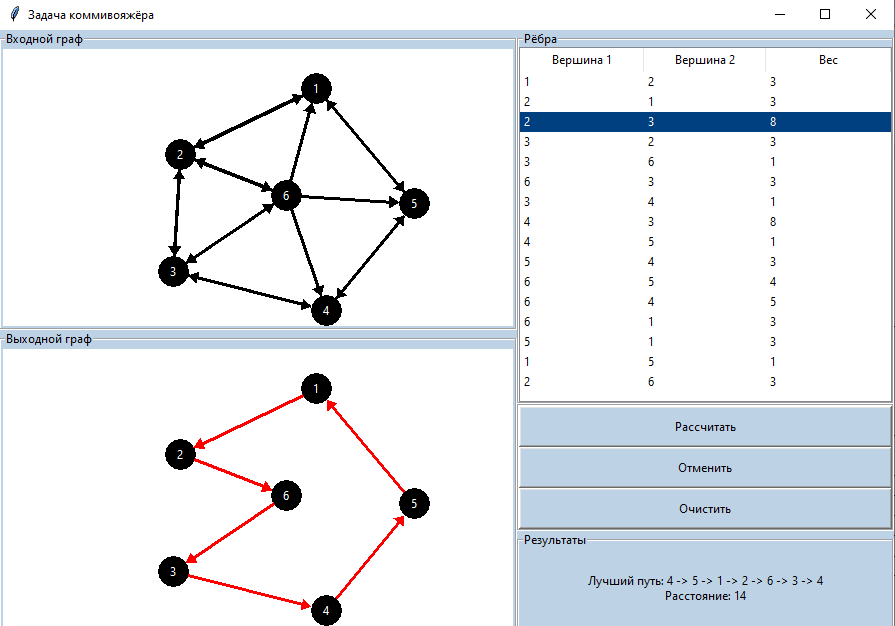


Рис. 5 Удачный результат

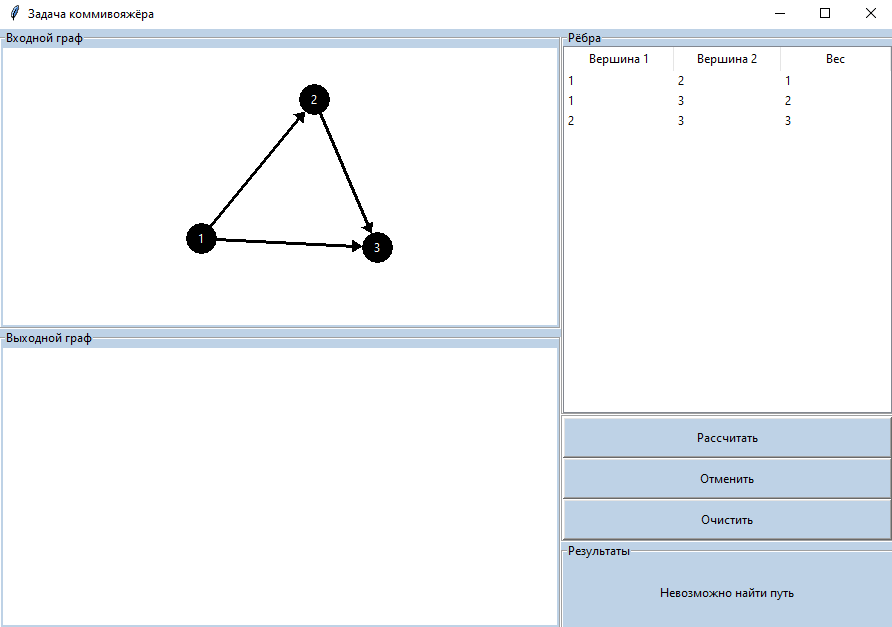


Рис. 6 Неудачный результат

# Анализ

В качестве модификации использовался **метод многократного запуска алгоритма ближайшего соседа** с каждой вершины графа. Вместо того, чтобы начать обход с одной фиксированной вершины, алгоритм выполняется несколько раз, каждый раз начиная с новой вершины, и выбирается оптимальный результат.

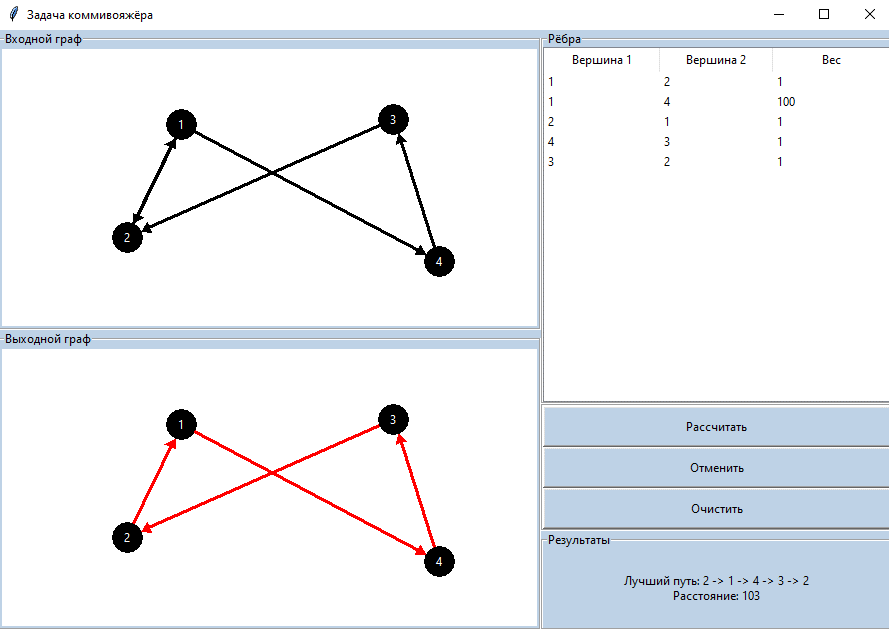


Рис. 7 Результат работы с модификацией

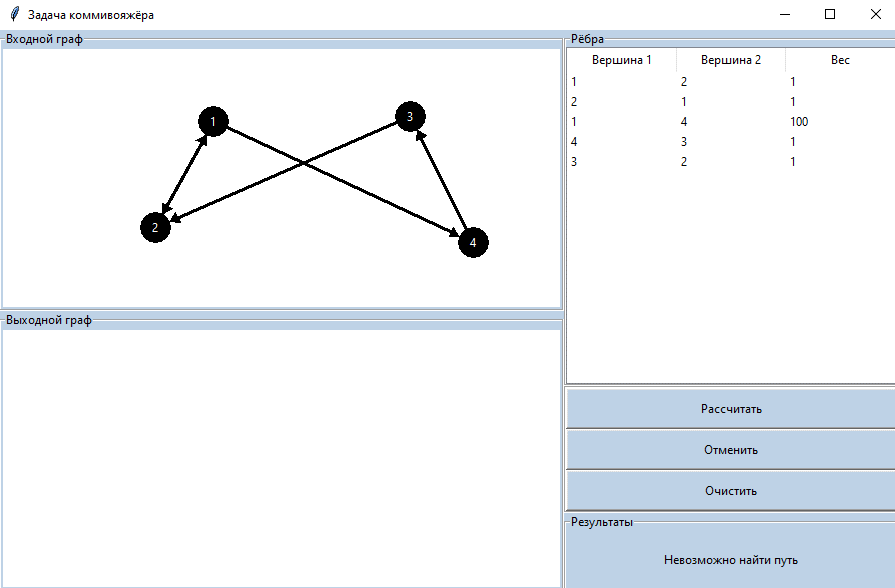


Рис. 8 Результат работы без модификации

Из (Рис. 7) видно, что алгоритм с модификацией, который начинает обход с каждой вершины и выбирает наилучший результат и находит гамильтонов цикл без проблем.

Алгоритм без модификации (Рис. 8), однако, застревает в вершине 2 уже после первого шага, поскольку он не может корректно обработать ситуацию с длинным ребром между вершинами 1 и 4 (вес 100), что приводит к тому, что алгоритм выбирает путь с меньшими весами, но не учитывает возможность прохождения через вершину 4 в дальнейшем.

# Вывод

В ходе работы был реализован и протестирован жадный алгоритм решения задачи коммивояжёра, а также его модифицированный вариант, позволяющий избежать локального застревания. Эксперименты показали, что стандартная версия алгоритма не всегда находит корректный гамильтонов цикл, так как жадный подход выбирает кратчайшее доступное ребро на каждом шаге, что может привести к ситуации, когда продолжение маршрута становится невозможным.

На примере тестового графа без модификации алгоритм застревает в вершине 2 после первого шага, так как дальнейшее движение по кратчайшему пути оказывается невозможным без нарушения условия обхода всех вершин ровно один раз. В то же время модифицированная версия, использующая перебор возможных стартовых вершин, успешно находит корректный гамильтонов цикл, что демонстрирует её эффективность.

Таким образом, предложенная модификация делает жадный алгоритм более универсальным, однако его применимость остаётся ограниченной для сложных графов. Выбор метода решения задачи должен основываться на компромиссе между точностью и скоростью вычислений.

# Источники

* Редактор блок-схем.

[*https://programforyou.ru/block-diagram-redactor*](https://programforyou.ru/block-diagram-redactor)

*дата обращения: (08.03.2025)*

* tkinter — Библиотека для создания графических интерфейсов в Python.

[*https://docs.python.org/3/library/tkinter.html*](https://docs.python.org/3/library/tkinter.html%20)

*дата обращения: (08.03.2025)*

* math — Библиотека для работы с математическими функциями.

[*https://docs.python.org/3/library/math.html*](https://docs.python.org/3/library/math.html)

*дата обращения: (08.03.2025)*