**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №8**

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

**на тему «Решение задачи о коммивояжере с помощью муравьиного алгоритма**»

**Студент гр. 23Б15-пу**

**Бек В.А.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc195371756)
2. [Описание задачи (формализация задачи) 4](#_Toc195371757)
3. [Теоретическая часть 5](#_Toc195371758)

[**Задача коммивояжера:** 5](#_Toc195371759)

[**Муравьиный алгоритм:** 5](#_Toc195371760)

[**Модификация (Блуждающая колония):** 6](#_Toc195371761)

1. [Основные шаги программы 7](#_Toc195371762)
2. [Блок схема программы 9](#_Toc195371763)
3. [Описание программы 11](#_Toc195371764)
4. [Рекомендации пользователя 13](#_Toc195371765)
5. [Рекомендации программиста 15](#_Toc195371766)
6. [Контрольный пример 16](#_Toc195371767)
7. [Анализ работы алгоритма с модификацией и без 18](#_Toc195371768)
8. [Сравнение с другими методами решения задачи коммивояжёра 19](#_Toc195371769)
9. [Вывод 21](#_Toc195371770)
10. [Источники 22](#_Toc195371771)

# Цель работы

Цель данной работы — изучение и реализация решения задачи о коммивояжере с использованием муравьиного алгоритма. 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 муравьиного алгоритма при решении задачи коммивояжера. Задача сформулирована 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 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 длину пути.

**В рамках работы необходимо**:

* Разработать программную реализацию муравьиного алгоритма и его модификации: Блуждающая колония.
* Визуализировать процесс построения маршрута.
* Провести анализ качества решения, сравнив его с другими методами минимизации.

В качестве модификации муравьиного алгоритма используется блуждающая колония, отличающийся от муравьиного алгоритма тем, что в каждый момент времени, т.е. на каждой итерации, вся колония перемещается в случайно выбранную вершину.

# 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.

### **Муравьиный алгоритм:**

Муравьиный алгоритм вдохновлен поведением реальных муравьев, которые, исследуя окружающую среду, оставляют феромоны на пути и выбирают кратчайшие маршруты к источникам пищи. В контексте задачи коммивояжера, муравьи случайным образом выбирают маршруты, оставляют феромоны, и со временем склоняются к кратчайшим путям, так как феромоны усиливаются на более коротких маршрутах.

1. **Основной цикл**:
   * Каждый муравей строит маршрут, выбирая вершины с учётом феромонов и расстояний.
   * После всех циклов обновляются феромоны: на лучших маршрутах феромоны увеличиваются.
   * Феромоны постепенно испаряются, чтобы избежать застревания в локальных минимумах.
2. **Завершение**: Алгоритм завершается, когда количество итераций достигает заданного лимита.

### **Модификация (Блуждающая колония):**

Модификация «Блуждающая колония» отличается тем, что вся колония муравьев на каждой итерации перемещается в случайно выбранную вершину, а не следует из собственной вершины. Это позволяет усилить исследование пространства решений и избежать преждевременной сходимости 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 алгоритмом имитации отжига.
5. **Отображение результата**
   * Если путь найден, он выводится в текстовом виде и рисуется на выходном холсте.
   * Если решения нет, отображается соответствующее сообщение.
6. **Дополнительные кнопки**

* **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.
* **Кнопка "Модификация"** позволяет выбрать алгоритм будет стартовать только из первой вершины или попробует из всех.
* **Кнопка "Сохранить граф"** открывает проводник и позволяет сохранить уже введённый граф в xlsx файл.
* **Кнопка "Загрузить граф"** открывает проводник и позволяет выбрать xlsx файл в котором уже записан граф.

# Блок схема программы

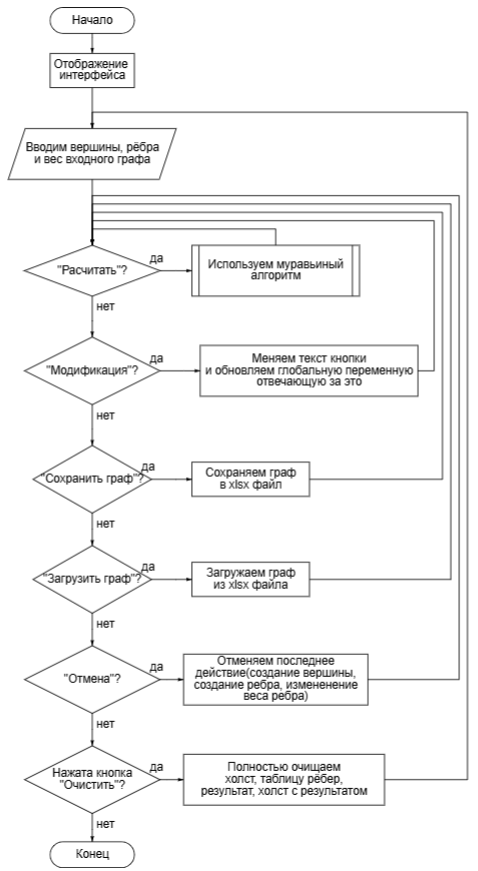


Рис. 1 Блок-схема основной программы

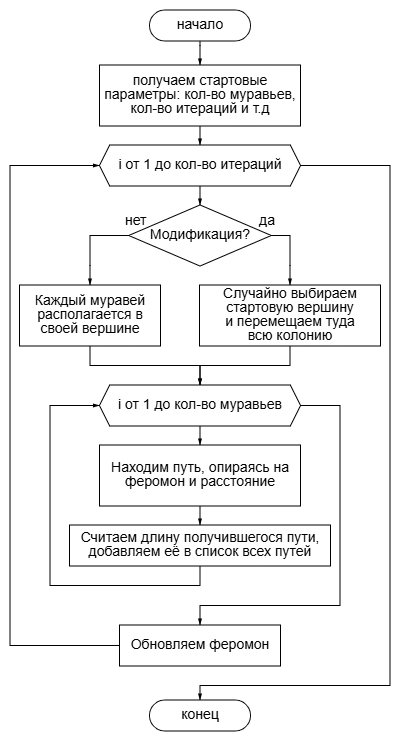


Рис. 2 Блок-схема подпрограммы

# Описание программы

Программная реализация задачи коммивояжёра написана на языке **Python 3.12.7** с использованием библиотек **tkinter, ttk, math, openpyxl**. Программа представляет собой графический интерфейс, который позволяет 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, сохранения и загрузки графа. Боле того предусмотрено отображает решение задачи коммивояжёра, выводя оптимальный маршрут и его длину.

Таблица 1. simulated\_annealing\_algorithm.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| toggle\_mode() | Меняет модификацию(Вкл/Выкл) | None |
| 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. | | None |
| ant\_colon\_algorithm(adj\_matrix) | Применяет муравьиный алгоритм или модификацию – Блуждающая колония | Кратчайший путь, его размер, кол-во итераций |
| 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.

* **Сохранение графа**:

Для сохранения введённого графа используйте кнопку "Сохранить граф". Откроется проводник, в котором вы сможете ввести название сохраняемого файла и его расположение.

* **Загрузка графа**:

Для загрузки введённого ранее графа используйте кнопку "Загрузить граф". Откроется проводник, в котором вы сможете выбрать файл, в который ранее сохраняли граф.

* **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.

* **Модификация**:

Нажатие на кнопку: “Модификация: Вкл.” позволит поменять режим работы алгоритма.

* **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 весь маршрута, а результаты расчета будут отображены в блоке "Результаты". Здесь будет показан оптимальный путь, общий вес маршрута и время выполнения алгоритма.

# 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 **3.1** 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 ant\_colony\_algorithm.py. Программа откроет графический интерфейс (Рис. 3).

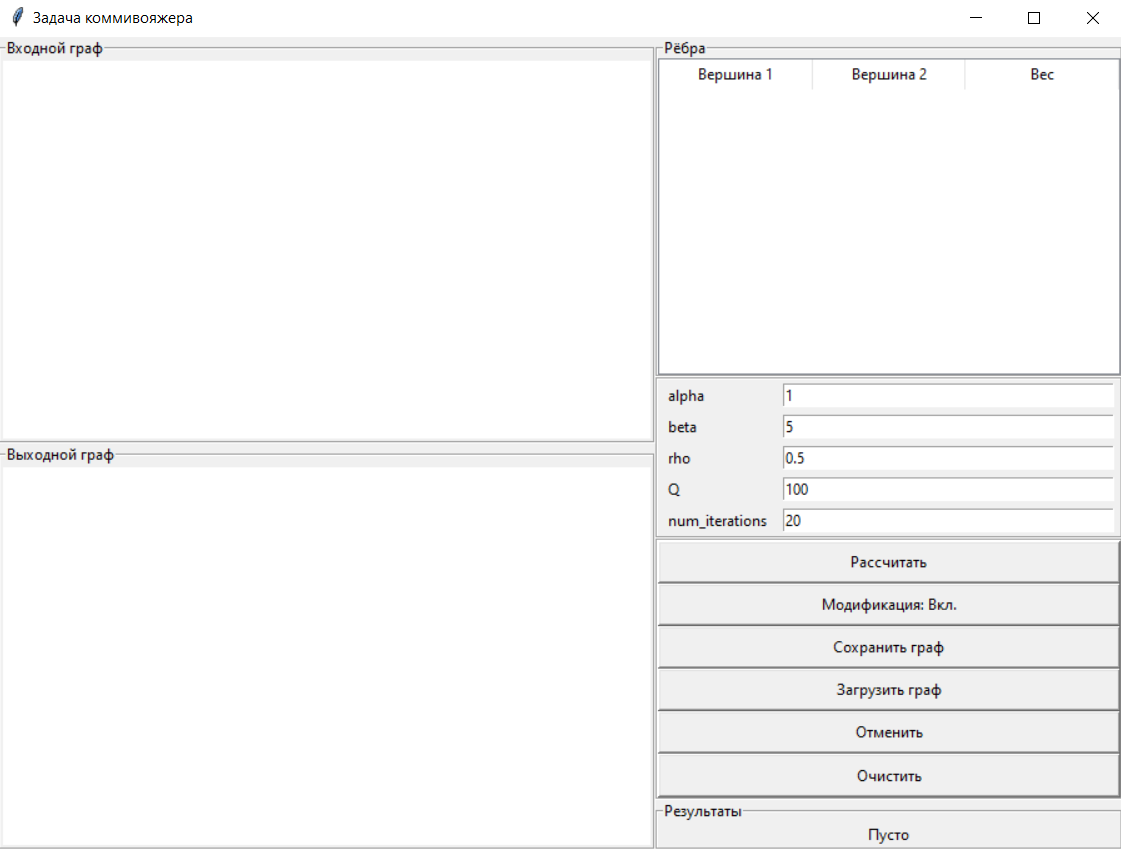


Рис. 3 Графический интерфейс программы

* **Инициализация частиц и параметров:** В фрейме “Входной граф” необходимо при помощи нажатий на правую и левую кнопки мыши изобразить нужный граф, под таблицей ребер при необходимости можно поменять параметры. (Рис. 4)

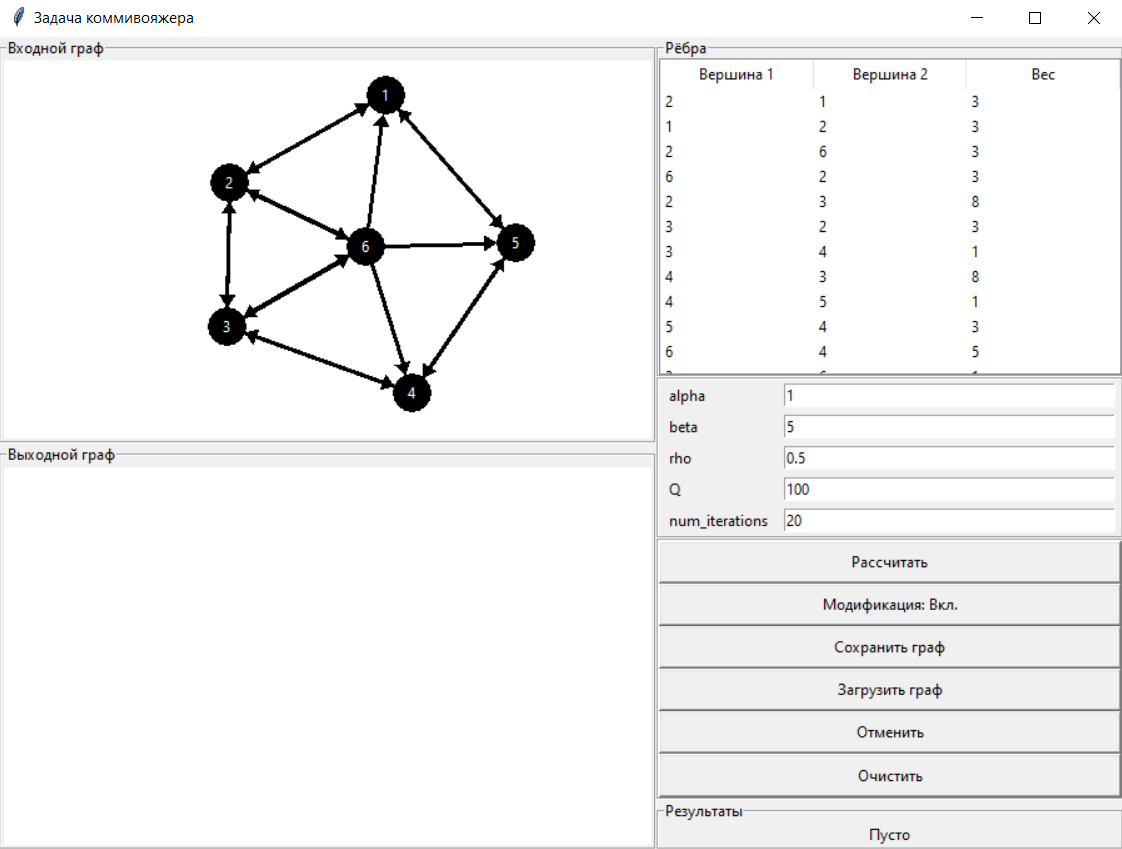


Рис. 4 Ввод графа и параметров

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

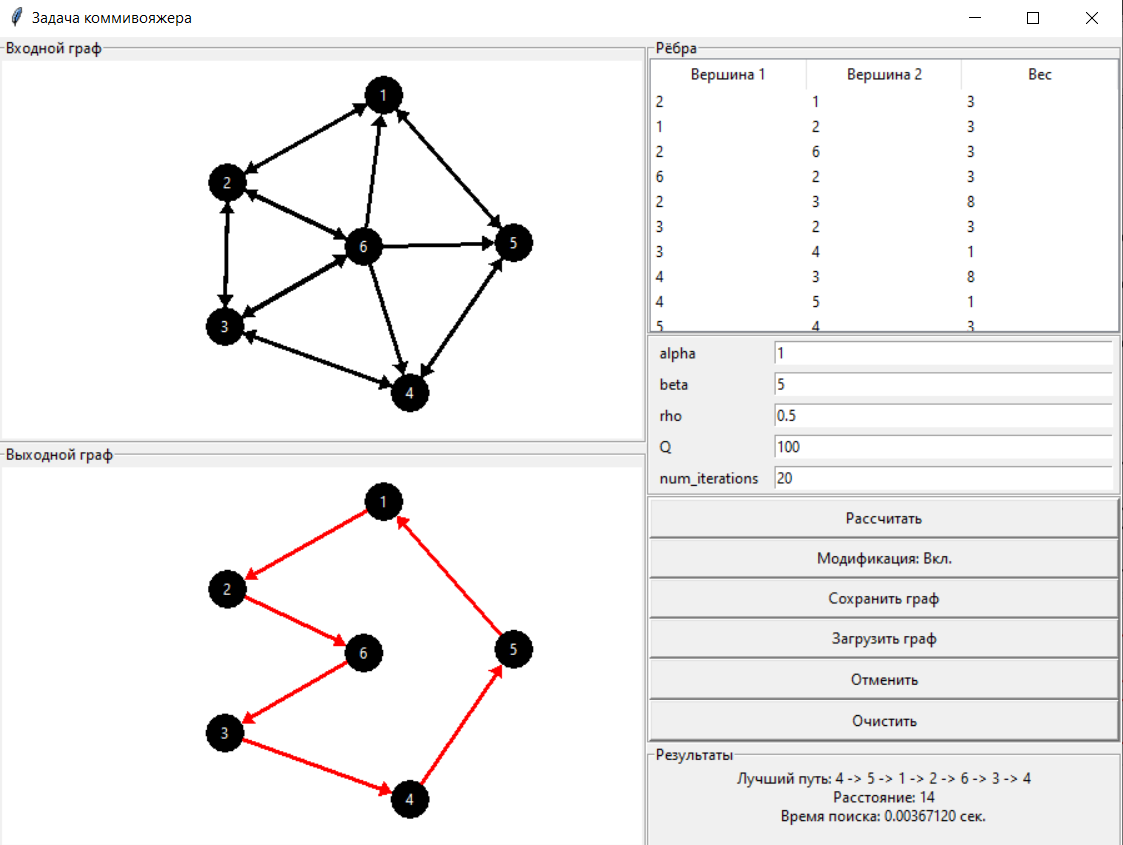


Рис. 5 Результат

# Анализ работы алгоритма с модификацией и без

В качестве модификации использовалась блуждающая колония, в которой способ выбора стартовой вершины изменён: вместо старта каждого муравья из своей вершины, вся колония начинает обход из одной, случайной вершины.

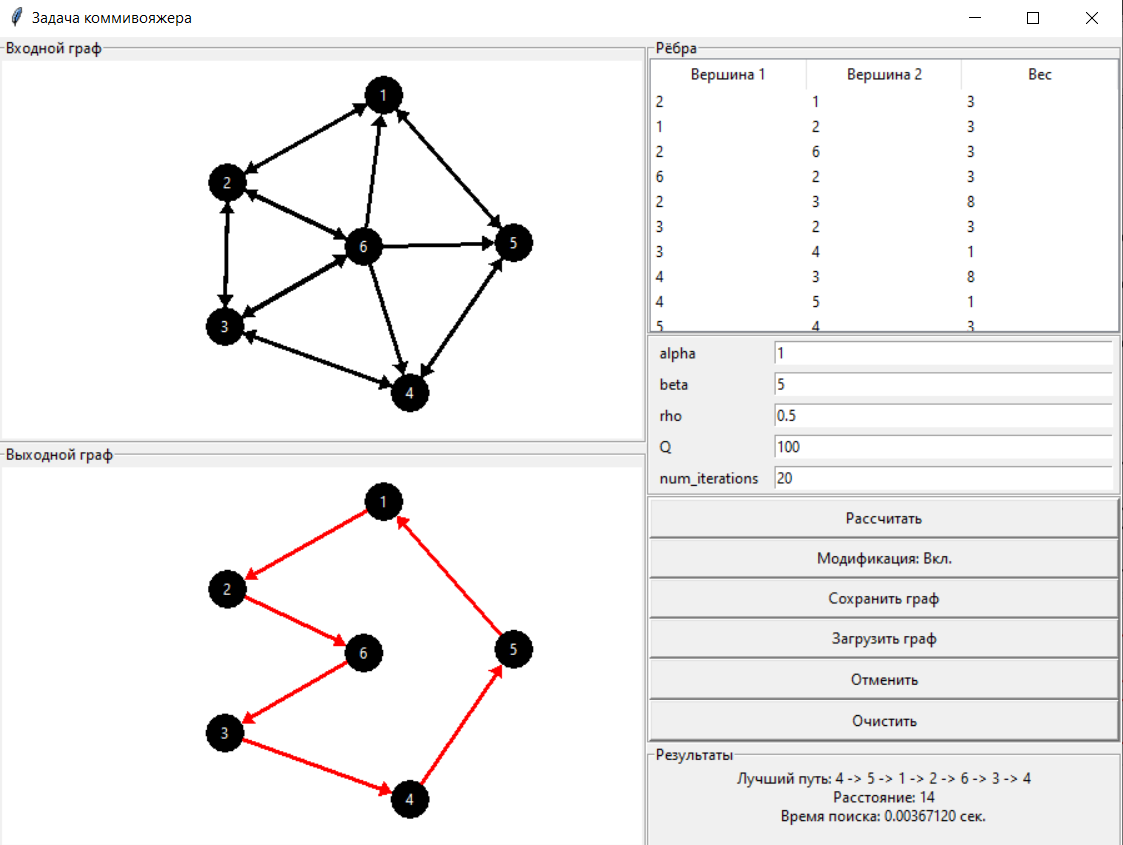


Рис. 7 Результат работы с модификацией

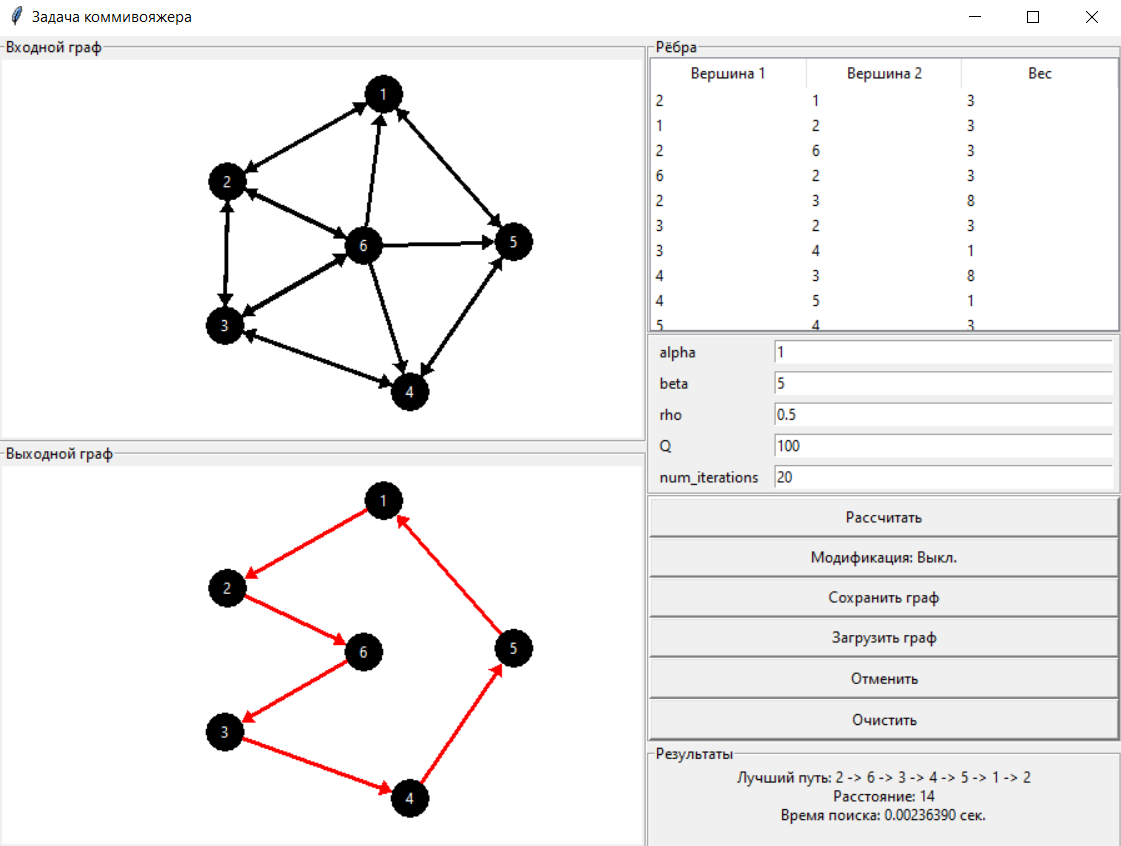


Рис. 8 Результат работы без модификации

# Сравнение с другими методами решения задачи коммивояжёра

Таблица 2. Сравнение работы алгоритма ближайшего соседа с модификацией и без

|  |  |  |  |
| --- | --- | --- | --- |
| Модификация | Время | Длина пути | Кол-во вершин |
| С модификацией | 0.00031… | 4607 | 10 |
| Без модификации | 0.00005… | 7136 | 10 |
| С модификацией | 0.00081… | 3451 | 15 |
| Без модификации | 0.00008… | 3451 | 15 |
| С модификацией | 0.00402… | 4862 | 30 |
| Без модификации | 0.00020… | 7810 | 30 |

Таблица 3. Сравнение работы алгоритма имитации отжига с модификацией и без

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модификация | Время | Длина пути | Кол-во вершин | Кол-во итераций |
| С модификацией | 0.006… | 3500 | 10 | 20000 |
| Без модификации | 0.005… | 2600 | 10 | 18412 |
| С модификацией | 0.080… | 3450 | 15 | 30000 |
| Без модификации | 0.060… | 3450 | 15 | 18412 |
| С модификацией | 0.202… | 5500 | 30 | 60000 |
| Без модификации | 0.080… | 6500 | 30 | 18412 |

Таблица 4. Сравнение работы муравьиного алгоритма с модификацией и без

|  |  |  |  |
| --- | --- | --- | --- |
| Модификация | Время | Длина пути | Кол-во вершин |
| С модификацией | 0.018… | 2800 | 10 |
| Без модификации | 0.014… | 3600 | 10 |
| С модификацией | 0.039… | 3270 | 15 |
| Без модификации | 0.034… | 3400 | 15 |
| С модификацией | 0.172… | 3260 | 30 |
| Без модификации | 0.160… | 3700 | 30 |

Исходя из полученных результатов можно сделать вывод, что Модификация алгоритмов по-разному влияет на эффективность решения задачи коммивояжёра в зависимости от метода:

* **Алгоритм ближайшего соседа** с модификацией показывает значительно меньшую длину пути, что свидетельствует о повышении качества решения, но время выполнения увеличивается. Особенно это заметно при увеличении числа вершин.
* **Алгоритм имитации отжига** демонстрирует примерно одинаковое качество решений как с модификацией, так и без неё, но модификация требует большего числа итерацийвремени и, что может снижать эффективность.
* **Муравьиный алгоритм** с применением модификации, основанной на случайном выборе следующей вершины (блуждающая колония), показывает лучшие результаты по сравнению с обычной версией, но требует немного больше времени.

Анализируя представленные данные, можно также сравнить эффективность различных алгоритмов между собой:

* **Алгоритм ближайшего соседа** — самый быстрый по времени исполнения, но при этом дает наихудшие по качеству решения, особенно при увеличении числа вершин. Он подходит для быстрого получения грубой оценки маршрута, но неэффективен в поиске оптимального пути.
* **Алгоритм имитации отжига** демонстрирует хороший баланс между временем и качеством решения. Он стабильно находит короткие маршруты при умеренных затратах времени. Особенно эффективен при небольшом и среднем числе вершин.
* **Муравьиный алгоритм** как без модификации, так и с ней, показывает наилучшее качество решений на большинстве тестов, особенно при большом числе вершин. Однако он значительно медленнее остальных методов. Это делает его эффективным для задач, где приоритет — качество решения, а не скорость.

# Вывод

В ходе работы был реализован и протестирован муравьиный алгоритм для решения задачи коммивояжёра, а также его модифицированная версия, основанная на случайном выборе следующей вершины (блуждающая колония). Результаты экспериментов показали, что классическая версия алгоритма демонстрирует высокое качество решений, особенно на графах с большим числом вершин. Это подтверждает эффективность модели, основанной на феромоне и вероятностном выборе маршрутов.

Модифицированная версия алгоритма оказалась более точной, но менее быстрой. Это свидетельствует о том, что случайный выбор следующей вершины ухудшает время выполнения, но улучшает его качество решений.

Исходя из сравнений, можно сказать, что и классический, и модифицированный, муравьиный алгоритм подтвердили свою пригодность для решения задачи коммивояжёра в условиях, когда критически важна точность, а не скорость.

# Источники

* Редактор блок-схем.

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

*дата обращения: (12.04.2025)*

* 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: (12.04.2025)*

* 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: (12.04.2025)*

* openpyxl — Библиотека для записи и чтения файлов Excel.

[*https://openpyxl.readthedocs.io/en/stable/*](https://openpyxl.readthedocs.io/en/stable/)

*дата обращения: (12.04.2025)*