**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**

**Черевко М.Е.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc198149576)
2. [Описание задачи (формализация задачи) 4](#_Toc198149577)
3. [Теоретическая часть 5](#_Toc198149578)
4. [Основные шаги программы 8](#_Toc198149579)
5. [Блок схема программы 9](#_Toc198149580)
6. [Описание программы 11](#_Toc198149581)
7. [Рекомендации пользователя 13](#_Toc198149582)
8. [Рекомендации программиста 14](#_Toc198149583)
9. [Исходный код программы 14](#_Toc198149584)
10. [Контрольный пример 15](#_Toc198149585)
11. [Анализ результатов работы алгоритма 18](#_Toc198149586)
12. [Вывод 20](#_Toc198149587)
13. [Источники 21](#_Toc198149588)

# Цель работы

Целью данной лабораторной работы является исследование муравьиного алгоритма для решения задачи коммивояжера (Traveling Salesman Problem, TSP), а также анализ эффективности его модифицированной версии с использованием шаблонов. В ходе работы будет разработана программа, реализующая муравьиный алгоритм, с возможностью визуализации найденного маршрута, обеспечивающая возможность создания графа вручную, загрузки графа из файла, поиска оптимального маршрута с минимальной общей стоимостью и визуализации результатов, а также предоставлять функции отмены действий и сброса данных.

# Описание задачи (формализация задачи)

1. Изучение задачи коммивояжера и муравьиного алгоритма.

Ознакомиться с муравьиным алгоритмом, его применением для поиска приближенного решения и с понятием феромонных следов.

1. Разработка программы для нахождения кратчайшего гамильтонова цикла.

Написать программу, реализующую муравьиный алгоритм для задачи коммивояжера. Добавить возможность использования шаблонов в качестве модификации алгоритма. Реализовать визуализацию построенного маршрута на графе.

1. Тестирование программы на взвешенном орграфе.

Провести тестирование программы на контрольном примере, представленном во взвешенном орграфе. Сравнить полученный маршрут с оптимальным решением, оценив точность и эффективность метода.

1. Анализ результатов

Оценить влияние введения шаблонов на качество решения. Проанализировать поведение алгоритма при различных параметрах (количество муравьёв, уровень испарения, коэффициент усиления шаблонов и др.).

# Теоретическая часть

**Муравьиный алгоритм**

Муравьиный алгоритм (Ant Colony Optimization, ACO) — это метаэвристический метод оптимизации, вдохновлённый коллективным поведением реальных муравьёв при поиске кратчайшего пути от муравейника к источнику пищи. В основе алгоритма лежит идея о том, что муравьи оставляют феромонный след на маршруте, усиливая вероятность его выбора другими особями. Со временем наилучшие маршруты получают наибольшее усиление и формируют основу для приближённого поиска оптимального решения.

**Параметры алгоритма и их роль**

1. **— влияние феромона**

Параметр определяет, насколько сильно при выборе следующего узла муравей ориентируется на *уровень феромона* на рёбрах.

* При = 0 муравьи полностью игнорируют феромон и двигаются только по расстоянию.
* При больших значениях решение становится более жадным, алгоритм быстро концентрируется на одном маршруте. Коэффициент испарения

1. **— влияние расстояния**

Параметр регулирует, насколько муравей учитывает эвристическую информацию — обычно это обратное расстояние:

* При = 0 выбор основан только на феромоне.
* При высоких муравьи тяготеют к ближайшим вершинам.

1. **— коэффициент испарения феромона**

Параметр (0 < < 1) определяет долю испарения феромона после каждой итерации:

Меньшие значения позволяют дольше сохранять старые следы, а большие значения — быстрее «забывать» неэффективные маршруты.

1. **— количество феромона (вклад муравья)**

После прохождения маршрута муравей откладывает феромон вдоль своего пути пропорционально качеству маршрута. Если длина маршрута

то вклад определяется как:

**Вероятность перехода**

После Вероятность того, что муравей в вершине 𝑖 выберет переход в вершину 𝑗 определяется по формуле:

где:

* — количество феромона на ребре
* — эвристическая привлекательность (обратное расстояние)
* — коэффициенты влияния феромона и расстояния

**Модификация алгоритма**

В модифицированной версии алгоритма добавляется механизм запоминания часто используемых рёбер (паттернов), которые получают дополнительное усиление феромоном. Это позволяет укреплять маршруты, повторяющиеся среди лучших муравьёв.

1. После прохождения всех маршрутов формируется словарь, сохраняющий вхождения количества ребер в маршруты.
2. На этапе обновления феромона учитывается это количество:

где:

* 0.1 — коэффициент усиления шаблонов
* — количество повторений ребра в маршрутах
* — базовое количество откладываемого феромона

**Принцип работы**

1. Инициализируется граф с равными начальными феромонами на всех рёбрах.
2. На каждой итерации множество муравьёв независимо строит маршруты, выбирая рёбра на основе вероятности перехода.
3. Завершив маршрут, каждый муравей оставляет феромоны на пройденных рёбрах — чем короче путь, тем больше усиление.
4. Применяется испарение феромонов — значения уменьшаются на всех рёбрах, не усиливающихся текущими решениями.
5. Повторяется процесс до достижения заданного количества итераций или сходимости маршрута.

**Оценка эффективности**

Основные параметры, влияющие на качество решения:

1. Скорость сходимости

Благодаря феромонной памяти и выбору маршрутов с учётом шаблонов, алгоритм быстро концентрируется на хорошем приближении к оптимуму. Однако слишком сильное усиление может привести к преждевременной сходимости.

1. Качество решения

Механизм шаблонов помогает избегать локальных минимумов, удерживая полезные под маршруты даже при высоком испарении феромонов.

1. Число итераций

Алгоритм останавливается после достижения заданного количества итераций, что позволяет управлять временем работы и качеством результатов.

# Основные шаги программы

Основные шаги программы ant\_algorithm\_app.py

1. **Запуск программы:**

Инициализация графического интерфейса (GUI) с элементами управления:

* Область визуализации графа.
* Таблица для редактирования весов рёбер.
* Поля ввода параметров.
* Поле для вывода результатов.
* Флажок для включения модификации.
* Кнопки управления: «Найти решение», «Остановить», «Назад», «Сбросить», «Загрузить Граф», «Сохранить Результат».

1. **Настройка графа:**

* Добавление вершин кликом по области визуализации.
* Создание рёбер путем выбора двух вершин и задание весов ребер.
* Отображение данных о вершинах и рёбрах в таблице.
* Возможность редактирования весов рёбер.

1. **Запуск алгоритма:**

* При нажатии кнопки «Найти решение» выполняется алгоритм поиска пути.
* Если включена модификация, алгоритм использует шаблоны.

1. **Поиск решения:**

* Генерация начального маршрута.
* Итеративное улучшение маршрута методом муравьиного алгоритма.
* Обновление феромонов.
* Испарение феромонов.
* Повторение процесса.

1. **Обновление интерфейса:**

* Визуализация найденного пути на графе.
* Заполнение таблицы актуальными данными.
* Вывод результатов: маршрут, длина пути, время выполнения.

1. **Дополнительные возможности:**

* Отмена последнего действия (удаление вершины или ребра).
* Очистка графа для нового расчёта.
* Загрузка и сохранение графа в форматах txt или json.

# Блок схема программы

Рис 1.1 Блок-схема ant\_algorithm\_app.py

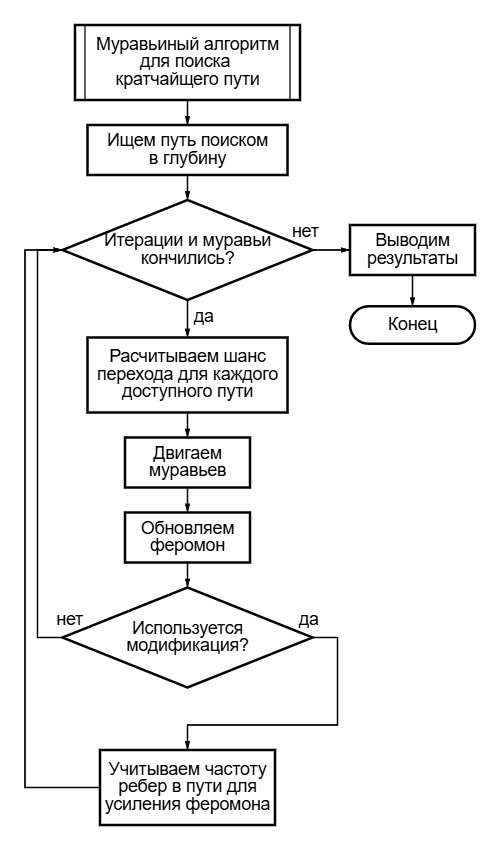


Рис 1.2 Блок-схема ant\_algorithm\_app.py

# Описание программы

Программа написана на языке Python 3.12.6 и использует библиотекe tkinter для реализации графического интерфейса решения задачи о коммивояжере с использованием алгоритма ближайшего соседа. В программе реализован класс – **TSPApp**, который отвечает за создание интерфейса, обработку графических событий, построение графа, вычисление кратчайшего пути и визуализацию результатов. Класс включает в себя методы, каждый из которых выполняет свою задачу в процессе работы приложения:

Таблица 1. TSPApp

|  |  |  |
| --- | --- | --- |
| Метод | Описание | Возвращаемое значение |
| \_\_init\_\_ | Инициализация интерфейса, параметров алгоритма и данных приложения. | None |
| \_setup\_ui | Создание и настройка графического интерфейса: панелей, канвасов, таблиц, кнопок и полей параметров. | None |
| \_create\_canvas\_frame | Создает канвас для отрисовки графа с прокруткой и привязывает его к родительскому элементу. | None |
| \_create\_param\_entry | Создает поле ввода для параметров алгоритма с валидацией. | None |
| \_validate\_param | Проверяет корректность введенных параметров (например, диапазон значений). | None |
| \_center\_window | Центрирует окно приложения на экране. | None |
| \_place\_node | Добавляет узел в граф при клике левой кнопкой мыши на канвасе. | None |
| \_pick\_node\_for\_link | Обрабатывает выбор узлов для создания ребра при клике правой кнопкой мыши. | None |
| \_render\_directed\_link | Отрисовывает направленное ребро между узлами с учетом их координат. | int (ID линии) |
| \_update\_link\_position | Обновляет позицию ребра при перемещении узлов. | None |
| \_solve\_tsp | Запускает алгоритм муравьиной колонии для поиска оптимального маршрута. | None |
| \_stop\_calculation | Прерывает выполнение алгоритма. | None |
| \_is\_connected | Проверяет связность графа | bool |
| ant\_colony\_optimization | Реализует алгоритм оптимизации (основная логика поиска маршрута). | tuple (маршрут, стоимость) |
| \_initialize\_pheromones | Инициализирует феромоны на ребрах графа. | defaultdict |
| \_construct\_routes | Строит маршруты для всех муравьев в текущей итерации. | tuple (маршруты, длины) |
| \_update\_pheromones | Обновляет уровень феромонов на ребрах с учетом найденных маршрутов. | None |
| \_display\_optimal\_route | Визуализирует найденный оптимальный маршрут на отдельном канвасе. | None |
| \_render\_route\_link | Отрисовывает ребро оптимального маршрута с заданным цветом. | None |
| \_load\_graph | Загружает граф из файла (узлы и ребра). | None |
| \_save\_result | Сохраняет результаты работы (маршрут, параметры) в JSON или текстовый файл | None |
| \_save\_json\_result | Сериализует данные в JSON-формат. | None |
| \_save\_text\_result | Сохраняет результат в текстовом формате. | None |
| \_revert\_last\_step | Отменяет последнее действие (добавление узла/ребра, изменение веса). | None |
| \_reset\_all | Полностью сбрасывает состояние приложения: удаляет граф, историю и результаты. | None |

# Рекомендации пользователя

Программа позволяет решить задачу о коммивояжере с помощью муравьиного алгоритма. С ее помощью можно построить граф, добавить узлы и ребра, а затем найти кратчайший гамильтонов цикл.

1. Запустите программу
2. В графическом интерфейсе:
   * Добавление узлов:

Щелкайте на области графа для создания новых узлов.

* + Добавление ребер:

Для соединения узлов выберите сначала один узел, затем второй – программа автоматически создаст ребро между ними.

* + Редактирование:

При необходимости измените вес ребра в таблице.

* + Ввод параметров:

При необходимости измените параметры в поле над выводом результатов.

* + Выбор модификации:

Для использования модификации шаблонов установите галочку в «Использовать модификацию».

1. Нажмите кнопку **«Найти решение»** для выполнения алгоритма. Программа вычислит кратчайший путь (гамильтонов цикл) и отобразит:
   * В поле **«Оптимальный путь»** — последовательность вершин кратчайшего маршрута.
   * На графике — построенный маршрут с выделенными ребрами.
2. Используйте кнопку **«Назад»** для удаления последнего действия или **«Сбросить»** для сброса всего графа.

# Рекомендации программиста

Для корректной работы программы убедитесь, что установлены следующие компоненты и выполнены необходимые шаги:

* Требования:
  + Python версии 3.12.0 или выше.
  + Необходимые библиотеки: tkinter, math, time, random, json, collections.
  + Проверьте, что графический интерфейс запускается корректно и результаты алгоритма (нахождение кратчайшего гамильтонова цикла) отображаются в соответствии с заданием.

# Исходный код программы

<https://github.com/FraaaM/Solving_the_Traveling_Salesman_Problem_Using_Ant_Colony_Algorithm>

# Контрольный пример

1. Запуск программы и формирование графа

Для запуска программы откройте файл **ant\_algorithm\_app.py**, содержащий код класса TSPApp. Программа откроет графический интерфейс, где в правой части окна расположены таблица для ввода данных о ребрах и панель для отображения вычисленного маршрута, в левой – область для построения и отображения графа (Рис. 2).

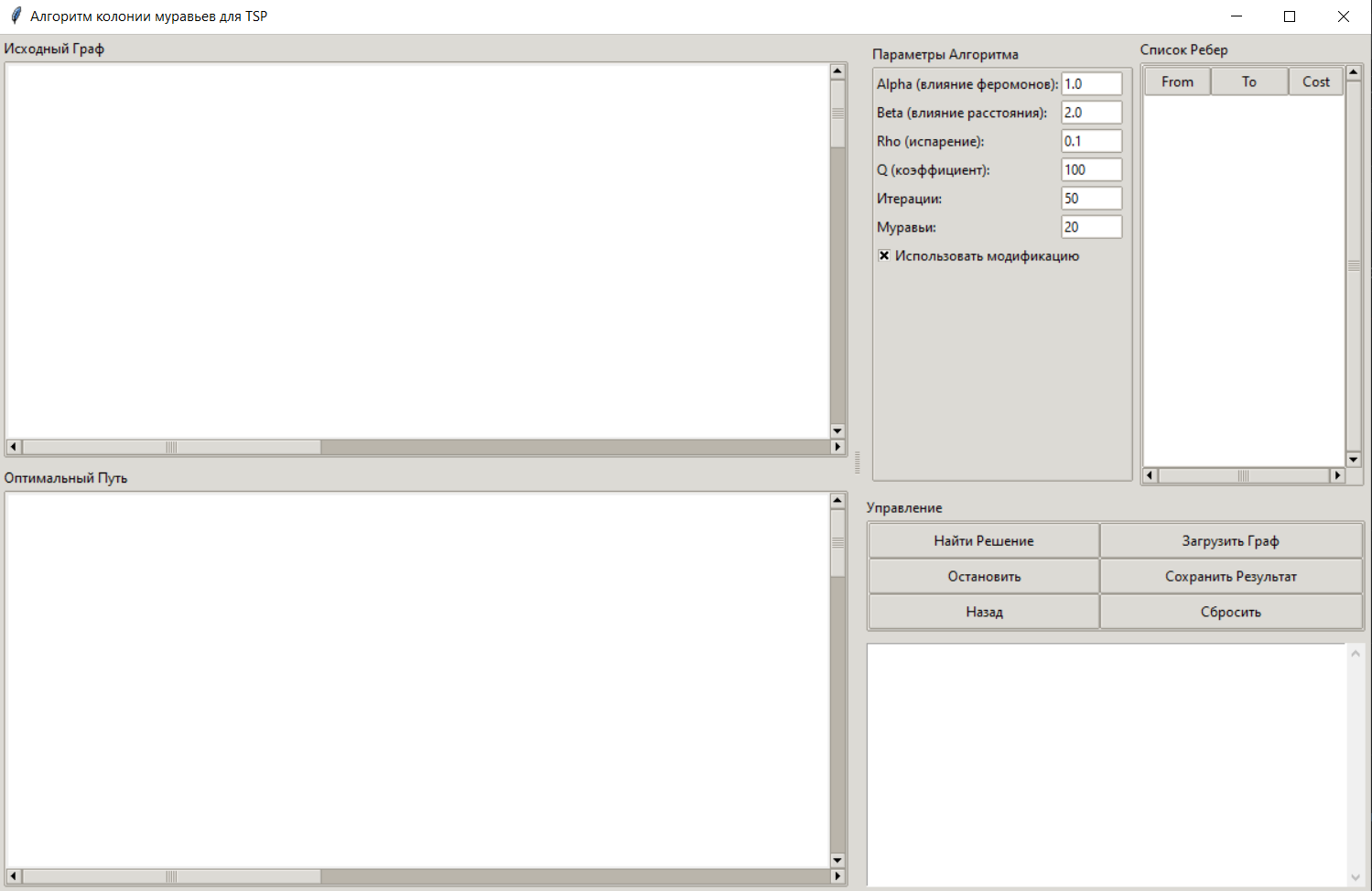


Рис 2. Пример окна программы

1. Ввод исходных данных

После запуска программы можно добавить вершины и соединить их ребрами следующим образом:

* Добавление вершин:

Щёлкните ЛКМ в области графа (левой части окна) для создания новой вершины.

* Соединение вершин:

после создания не менее 2 вершин можно создать ребра, нажав ПКМ по одной из вершин и так же через ПКМ щёлкнув по другой, добавив ребро между ними и задав вес ребра.

1. Запуск алгоритма

После формирования графа нажмите кнопку «Найти решение». Программа выполнит вычисление кратчайшего гамильтонова цикла с помощью муравьиного алгоритма, определяя маршрут, проходящий через все вершины и возвращающийся в исходную точку (Рис. 3). Если поставить крестик «Использовать модификацию», то будет использоваться модификация шаблонов.

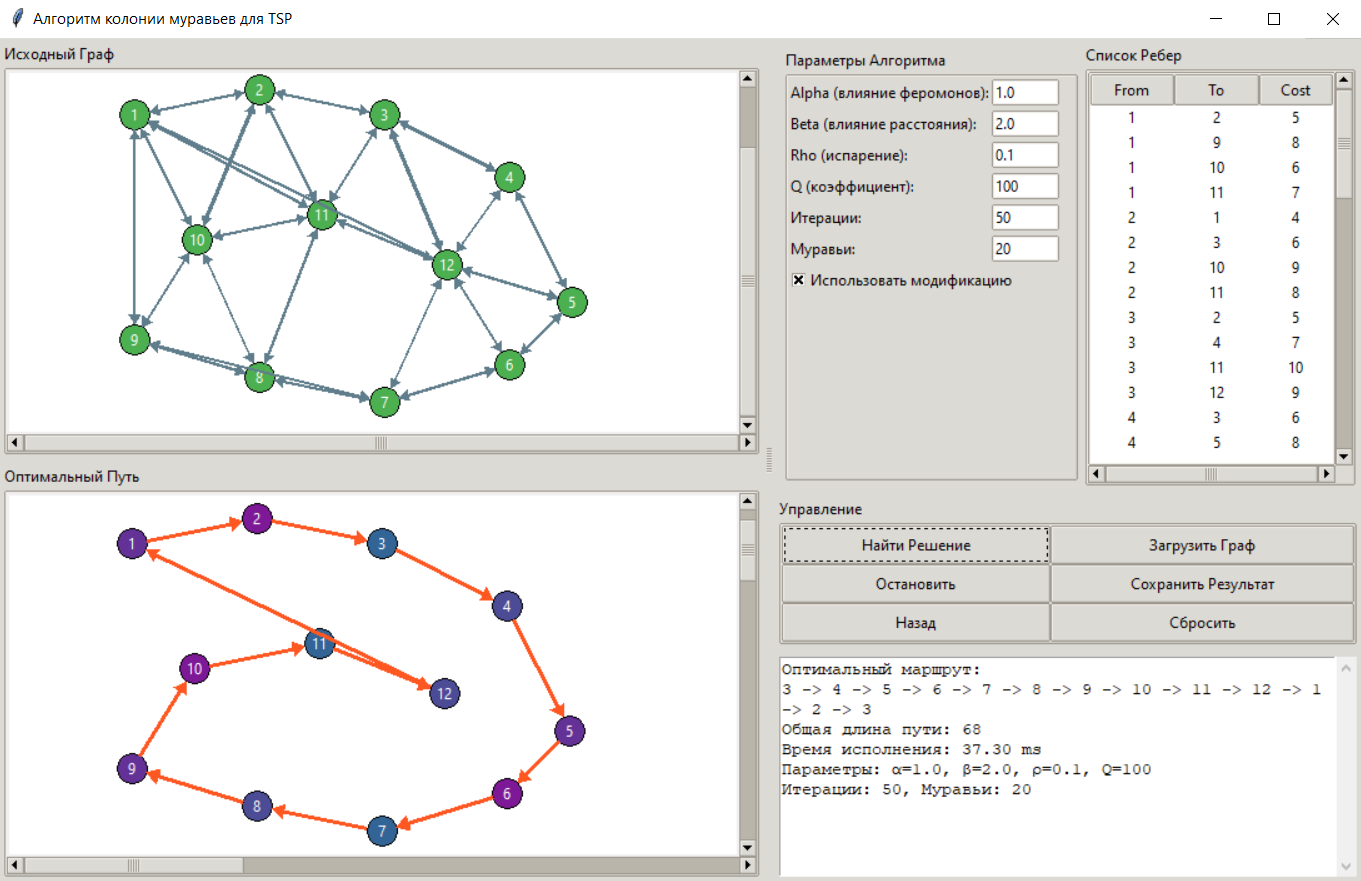


Рис 3. Пример результатов программы (с модификацией)

1. Просмотр результатов

После завершения расчёта в нижней части окна будут выведены следующие данные:

* Найденный кратчайший маршрут (последовательность вершин);
* Общая длина маршрута с заданной точностью;
* Время выполнения алгоритма.

Дополнительно в правой части окна отображается графическое представление маршрута с выделенными вершинами и соединяющими их стрелками.

1. Загрузка и сохранение графа.

Для сохранения нарисованного графа или загрузки его из файла txt нужно нажать кнопки «Загрузить Граф» и «Сохранить результат». Таблица смежность должна иметь следующий вид для корректной загрузки:

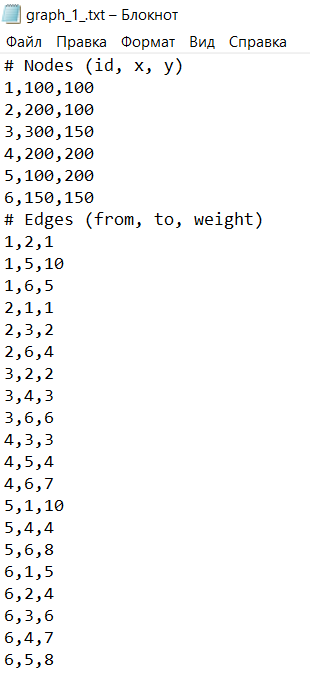


Рис 4. Пример задания таблицы графа

# Анализ результатов работы алгоритма

Для анализа работы муравьиного алгоритма и его модификации приведены примеры результатов для нахождения гамильтонова цикла методом ближайших соседей и методом имитации отжига на разных графах:

Таблица 2. Тесты метода ближайших соседей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Граф | Модификация | Время (в мс) | Длина пути | Кол-во вершин |
| 1 | С модификацией | 0.14 | 23 | 6 |
| 1 | Без модификации | 0.04 | 23 | 6 |
| 2 | С модификацией | 1.43 | 292 | 20 |
| 2 | Без модификации | 0.10 | 302 | 20 |
| 1 | С модификацией | 8.03 | 336 | 40 |
| 1 | Без модификации | 0.23 | 530 | 40 |

Таблица 3. Тесты алгоритма имитации отжига

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Граф | Модификация | Время (в мс) | Длина пути | Кол-во вершин |
| 1 | С модификацией | 0.97 | 23 | 6 |
| 1 | Без модификации | 0.96 | 23 | 6 |
| 2 | С модификацией | 18.99 | 317 | 20 |
| 2 | Без модификации | 18.86 | 410 | 20 |
| 1 | С модификацией | 96.06 | 583 | 40 |
| 1 | Без модификации | 96.02 | 517 | 40 |

Тесты проводились с количеством муравьев = 20, количеством итераций = 40, Alpha = 1, Beta = 2, Коэффициентом испарения = 0.3 и количеством феромона = 100

Таблица 4. Тесты муравьиного алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Граф | Модификация | Время (в мс) | Длина пути | Кол-во вершин |
| 1 | С модификацией | 15.01 | 12 | 6 |
| 1 | Без модификации | 12.02 | 23 | 6 |
| 2 | С модификацией | 122.03 | 265 | 20 |
| 2 | Без модификации | 113.89 | 265 | 20 |
| 1 | С модификацией | 394.31 | 284 | 40 |
| 1 | Без модификации | 386.37 | 285 | 40 |

**Анализ работы муравьиного алгоритма с модификацией и без**

Анализ данных из Таблицы 4 позволяет сделать вывод, что модификация

муравьиного алгоритма демонстрирует неоднозначную эффективность в зависимости от размера графа. Для небольших графов (6 вершин) модификация обеспечивает значительное сокращение длины пути (12 против 23) при умеренном увеличении времени работы (15.01 мс против 12.02 мс), что указывает на её полезность для оптимизации качества решения. Однако на средних и крупных графах (20 и 40 вершин) преимущество модификации становится менее выраженным: при схожем времени выполнения (122.03 мс против 113.89 мс для 20 вершин) длина пути остаётся идентичной (265), а для 40 вершин разница в длине минимальна (284 против 285). Это свидетельствует о том, что модификация, возможно, добавляет избыточные вычисления, которые не приводят к существенному улучшению результатов на сложных графах, что ставит под вопрос её целесообразность для масштабных задач.

**Анализ алгоритмов**  
Метод ближайших соседей демонстрирует наименьшее время выполнения (до 8 мс для 40 вершин), однако качество решений значительно уступает другим методам, особенно на крупных графах (длина пути 530 без модификации). Имитация отжига показывает более сбалансированные результаты: при времени выполнения до 96 мс для 40 вершин модификация улучшает длину пути на 15–20% (напр., 410 → 317 для 20 вершин). Муравьиный алгоритм, в свою очередь, требует значительно больше времени (до 394 мс для 40 вершин), но обеспечивает наилучшие решения для малых графов (длина 12 против 23 у других методов для 6 вершин). При этом на средних и крупных графах его преимущество сокращается: для 20 вершин все алгоритмы сходятся к длине ~265, а для 40 вершин разница между модификациями минимальна (284 vs 285).

Муравьиный алгоритм с модификацией оправдан для задач с небольшим количеством вершин (до 20), где он обеспечивает оптимальные решения, жертвуя временем. Для крупных графов его применение менее целесообразно, если не важна максимальная точность решения: высокие временные затраты (в 40–50 раз выше, чем у ближайших соседей) не компенсируются значительным улучшением качества,

# Вывод

В ходе лабораторной работы была успешно реализована программа для решения задачи коммивояжёра с использованием муравьиного алгоритма, а также создан графический интерфейс для визуализации построения графа и вывода решения.

Экспериментальные результаты показали, что метод эффективнее справляется с поиском оптимального решения нежели метод ближайшего соседа и немного лучше, чем алгоритм имитации отжига, затрачивая при этом хоть и значительно больше времени, он в разумных пределах.

# Источники

* Редактор блок-схем.

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

*дата обращения: (20.04.2025)*

* **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: (20.04.2025)*

* **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: (20.04.2025)*

* Алгоритм имитации отжига :

https://ru.wikipedia.org/wiki/Алгоритм\_имитации\_отжига

(дата обращения: (20.04.2025).