**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №3**

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

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

**Студент гр. 23Б15-пу**

**Голуб Г.Я.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

**Оглавление**

1. [Цель работы 2](#_j7ckij8kudef)
2. [Описание задачи (формализация задачи) 3](#_x0x3mtc5vsh5)
3. [Теоретическая часть 3](#_373pvn5mvuzi)
4. [Основные шаги программы 4](#_js4bgziwbyax)
5. [Блок схема программы 4](#_je4bbz2qhgxa)
6. [Описание программы 6](#_9oynku6k00si)
7. [Рекомендации пользователя 7](#_i3p0gg7gtyod)
8. [Рекомендации программиста 8](#_kwfy7ha81mtn)
9. [Исходный код программы 8](#_mt71clxnnmom)
10. [Контрольный пример 8](#_hgw9hyd2c4hy)
11. [Вывод 10](#_psh4iog389tw)
12. [Источники 11](#_wb8x5y8844ov)

# Цель работы

Целью лабораторной работы является реализация и исследование муравьиного алгоритма для решения задачи коммивояжёра. В рамках работы создаётся программное средство, позволяющее пользователю вводить граф в виде матрицы смежности, решать задачу с использованием муравьиного алгоритма и визуализировать полученные маршруты. Особое внимание уделяется исследованию эффективности муравьиного алгоритма и его модификации «Элитные» муравьи.

# Описание задачи (формализация задачи)

Задача коммивояжёра — это классическая NP-трудная задача комбинаторной оптимизации, в которой необходимо найти кратчайший гамильтонов цикл (путь, проходящий через все вершины ровно один раз и возвращающийся в исходную точку) в взвешенном графе.   
В данной работе рассматривается направленный граф, представленный в виде матрицы смежности, где каждая вершина обозначает пункт назначения, а рёбра имеют веса, соответствующие расстояниям или затратам на переход между вершинами.   
Основное требование — минимизация общей длины маршрута.

# Теоретическая часть

Муравьиный алгоритм (Ant Colony Optimization, ACO) — это эвристический алгоритм, имитирующий поведение муравьёв при нахождении кратчайших путей. В алгоритме множество искусственных "муравьёв" строят решения, ориентируясь на феромоны (историческую привлекательность рёбер) и эвристическую информацию (например, обратную длину рёбер). После итерации феромоны обновляются, усиливая хорошие маршруты.

Модификация с "элитными" муравьями предполагает, что кроме обычного обновления феромонов, дополнительное количество феромонов добавляется на рёбра лучшего маршрута, найденного за всю историю. Это позволяет ускорить сходимость и повысить стабильность алгоритма.

# Основные шаги программы

1. Ввод пользователем списка вершин и матрицы смежности.
2. Запуск алгоритма симуляции отжига для поиска кратчайшего пути.
3. Опционально: включение модификации.
4. Отображение оптимального пути в текстовом формате.
5. Визуализация графа с подсвеченным маршрутом, где используются различные цвета для отображения рёбер, входящих в оптимальный путь.

# Блок схема программы

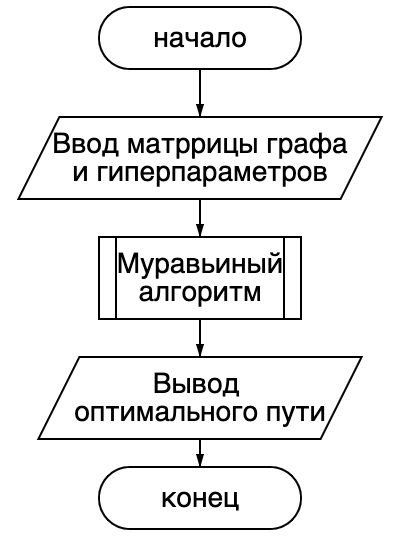


Рис 1. Блок-схема основной программы

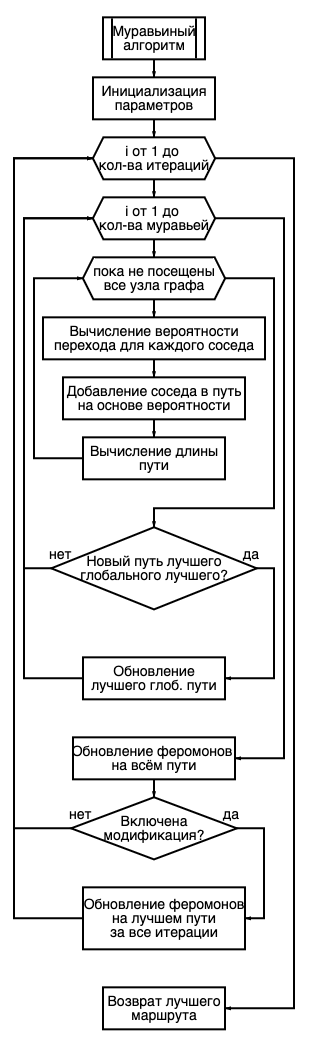


Рис 2. Блок-схема подпрограмм

# Описание программы

Программа реализована на языке Python с использованием библиотек PyQt5[[1]](#_wb8x5y8844ov) для пользовательского интерфейса и NetworkX[[2]](#_wb8x5y8844ov) для построения и визуализации графов.   
 Пользователь вводит граф через текстовое поле, после чего алгоритм ближайшего соседа вычисляет маршрут и отображает его на графе.   
 Программа позволяет переключать режимы: стандартный алгоритм симуляции отжига и улучшенную версию, использующую модификацию «Элитные» муравьи.

Таблица 1. main.py

| Функция | Описание | Возвращаемое значение |
| --- | --- | --- |
| parse\_graph\_input | Считывание графа из поля ввода | None |
| ant\_colony | Муравьиный алгоритм | list |
| draw\_graph | Отображение графа | None |
| initUI | Функция инициализации интерфейса | None |
| GraphEditor | Класс, отвечающий за редактор графа | None |
| solve\_tsp | Запускает решение задачи | None |
| open\_graph\_editor | Запускает редактор графа | None |

# Рекомендации пользователя

- Вводите граф в корректном формате:   
 ```   
 A B C D E   
 0 3 1 4 8   
 2 0 5 1 3   
 6 2 0 3 7   
 4 3 8 0 2   
 5 6 3 4 0   
 ```   
 - Используйте опцию «Использовать улучшенную версию», если хотите получить более оптимальный маршрут.   
 - Следите за корректностью входных данных, чтобы избежать ошибок в вычислениях.

# Рекомендации программиста

1. Используйте актуальные версии библиотек PyQt5[[1]](#_wb8x5y8844ov) и NetworkX[[2]](#_wb8x5y8844ov).
2. Для улучшения визуализации можно добавить анимацию построения маршрута.
3. Возможны дальнейшие модификации алгоритма, например, с использованием динамического программирования для приближения к оптимальному решению.

# Исходный код программы

<https://github.com/zgjhz/algorithms_term_2>

# Контрольный пример

1. Запуск программы

Для запуска программы используйте файл main.py**.**

2. Ввод графа (Рис. 3)

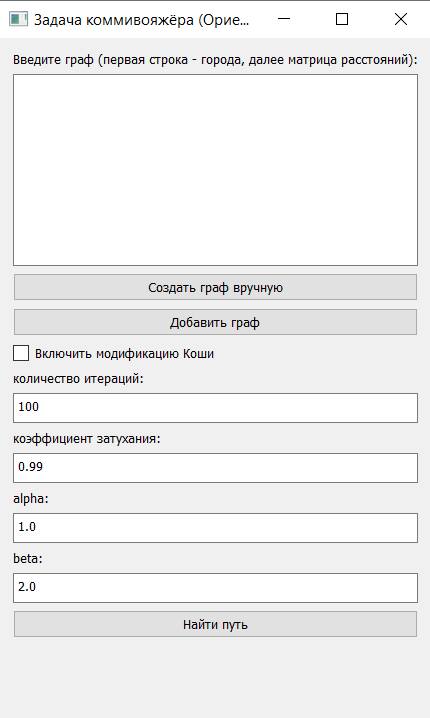
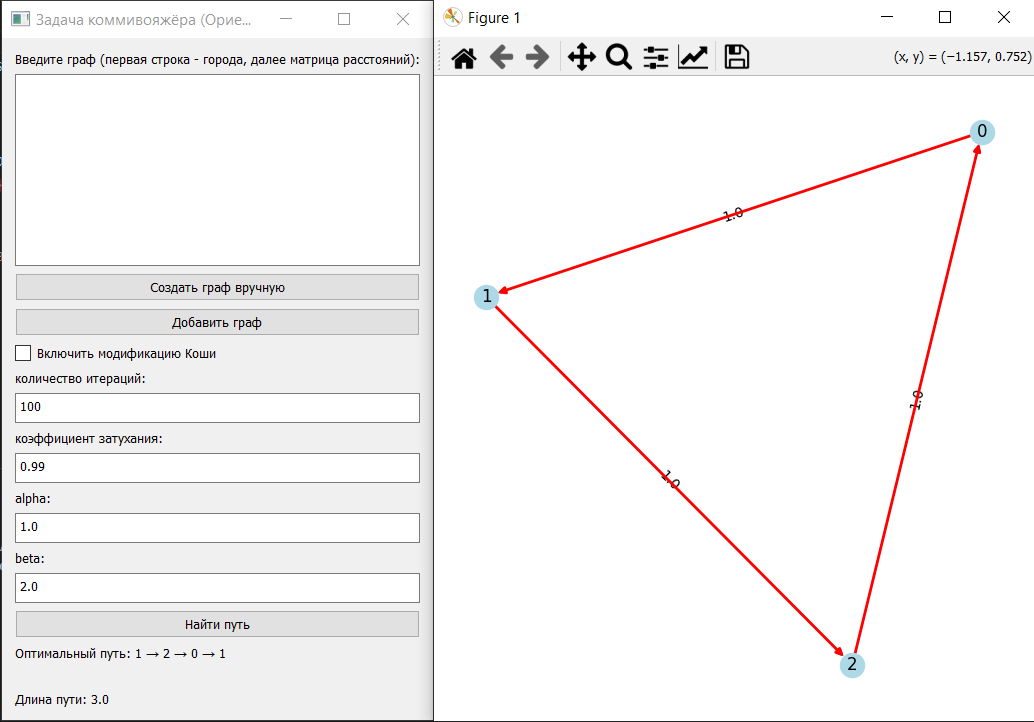


Рис. 3 (Ввод матрицы графа)

3. Нажмите кнопку “Найти путь”. По желание можно включить модификацию и выбрать гиперпарамтеры.

4. Вывод графа с найденным путём (Рис. 4)

Рис. 4 (Вывод программы)

# Сравнение алгоритма с и без модификации

Ниже представлена таблица сравнения алгоритма с модификацией и без. Сравнение проводилось на 30 тестах на графaх, приведённых ниже (Рис. 5-8)

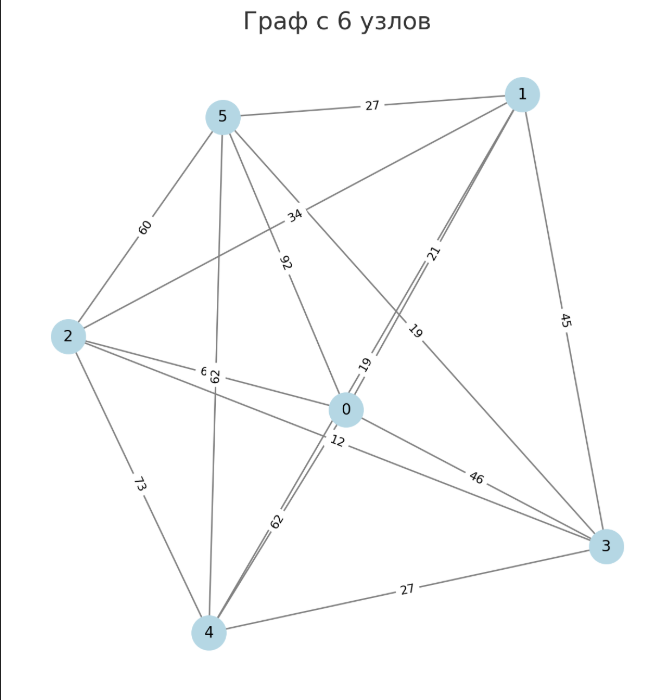


Рис. 5 (граф с 6 вершинами)

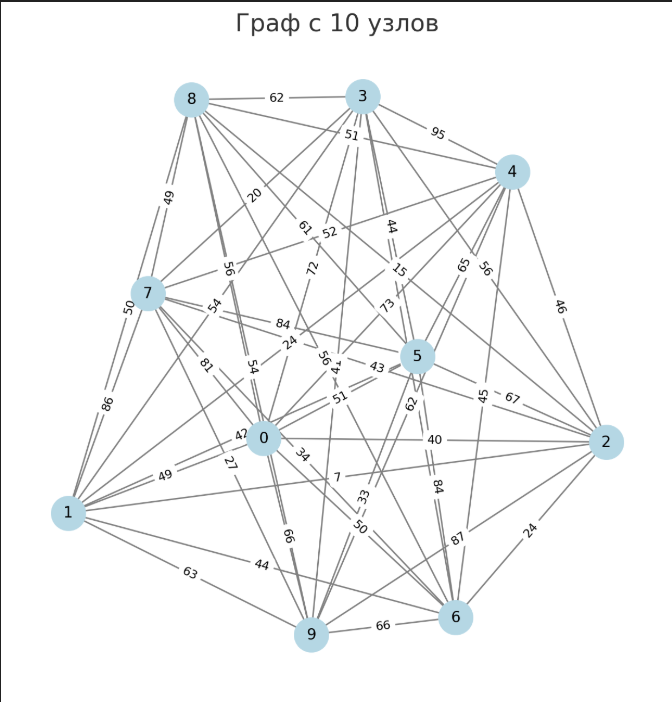


Рис. 6 (граф с 10 вершинами)

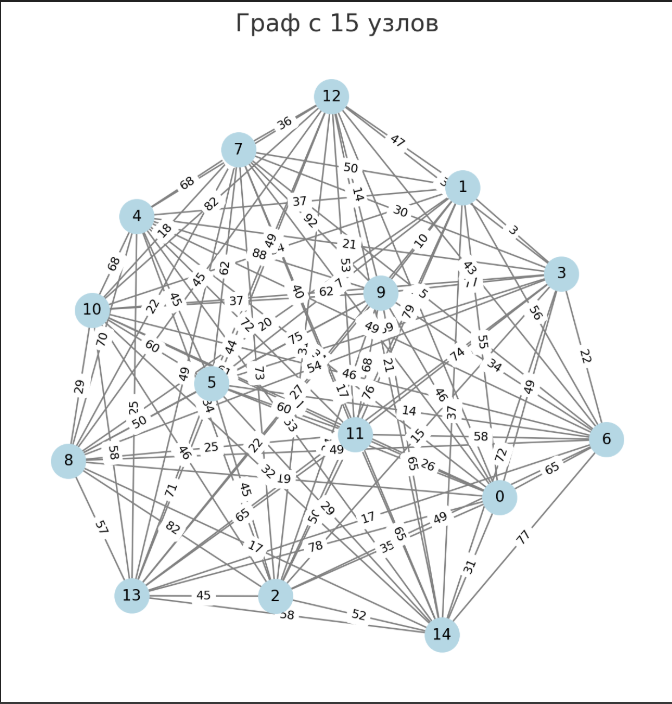


Рис. 7 (граф с 15 вершинами)

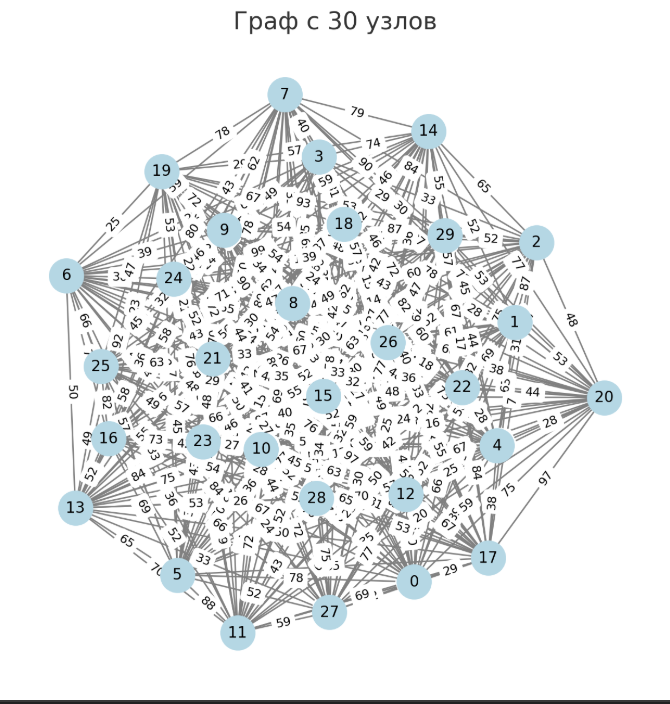


Рис. 8 (граф с 30 вершинами)

Таблица 2. Сравнение результатов алгоритма симуляции отжига с и без модификации

| Размер графа | Стоимость (обычный) | Время (обычный) | Стоимость (Коши) | Время (Коши) |
| --- | --- | --- | --- | --- |
| 6 | 36.07 | 0.01777 | 34.47 | 0.01832 |
| 10 | 32.32 | 0.02223 | 32.08 | 0.02432 |
| 15 | 36.23 | 0.01866 | 31.31 | 0.01954 |
| 30 | 54.34 | 0.03082 | 48.08 | 0.03193 |

Таблица 3. Сравнение результатов алгоритма ближайшего соседа с и без модификации

| Размер графа | Стоимость (обычный) | Время (обычный) в сек. | Стоимость (с модификацией) | Время (с модификацией) в сек. |
| --- | --- | --- | --- | --- |
| 6 | 36.17 | 0.00002 | 35.80 | 0.00006 |
| 10 | 45.67 | 0.00005 | 43.67 | 0.00048 |
| 15 | 57.27 | 0.00009 | 55.80 | 0.00135 |
| 30 | 60.32 | 0.000298 | 56.48 | 0.00823 |

Таблица 4. Сравнение результатов муравьиного алгоритма с и без модификации

| Размер графа | Стоимость (обычный) | Время (обычный) | Стоимость (Модификация) | Время (Модификация) |
| --- | --- | --- | --- | --- |
| 6 | 30.00 | 0.10023 | 30.00 | 0.10067 |
| 10 | 31.03 | 0.31322 | 30.20 | 0.32015 |
| 15 | 27.23 | 0.75157 | 26.97 | 0.76844 |
| 30 | 42.77 | 3.36487 | 40.07 | 3.41521 |

# Вывод

В рамках лабораторной работы было проведено сравнение трёх эвристических алгоритмов решения задачи коммивояжёра — алгоритма ближайшего соседа, симуляции отжига и муравьиного алгоритма. Для каждого из них была реализована и протестирована модификация, направленная на улучшение качества решений.

1. **Алгоритм ближайшего соседа** продемонстрировал наименьшее время выполнения на всех размерах графов. Однако при этом его маршруты оказались наименее оптимальными по стоимости, особенно при увеличении размера графа. Модификация позволила улучшить качество решений (например, для графа из 30 узлов стоимость уменьшилась с 60.32 до 56.48), но сопровождалась ростом времени выполнения, хотя и оставалась в пределах миллисекунд.
2. **Алгоритм имитации отжига** показал сбалансированные результаты. Он уступает ближайшему соседу по времени, но выдаёт более качественные маршруты. Применение модификации с распределением Коши позволило дополнительно снизить стоимость маршрутов (например, для графа из 15 узлов — с 36.23 до 31.31), при этом увеличение времени выполнения было незначительным (в пределах 1–2 мс), что делает модифицированный отжиг эффективным с точки зрения баланса "время-качество".
3. **Муравьиный алгоритм** оказался самым затратным по времени, особенно на графах с числом узлов 30 (время выполнения до 3.36 сек в обычной версии и 3.42 сек в модифицированной). Однако он показал наилучшие результаты по стоимости маршрута, особенно на графах среднего и большого размера. Модификация с применением "элитных" муравьёв обеспечила небольшое улучшение качества решений при минимальном увеличении времени выполнения.

Алгоритм ближайшего соседа может использоваться для сверхбыстрой генерации маршрутов, когда точность не критична. Симуляция отжига с модификацией — разумный компромисс между скоростью и качеством. Муравьиный алгоритм подходит для задач, где приоритет отдаётся качеству маршрута, несмотря на больший объём вычислений.

# Источники

**1. PyQt5 Documentation: https://doc.qt.io/qtforpython/**

**2. NetworkX Documentation: https://networkx.github.io**