**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №2**

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

**на тему «**Решение задачи коммивояжёра с помощью метода симуляции отжига**»**

**Студент гр. 23Б15-пу**

**Голуб Г.Я.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

**Оглавление**

1. [Цель работы 2](#_ljmtu1zevjnn)
2. [Описание задачи (формализация задачи) 3](#_a9e7ap1vz1zy)
3. [Теоретическая часть 3](#_h147l86tfmbp)
4. [Основные шаги программы 4](#_72iiydzenvan)
5. [Блок схема программы 4](#_ke4z4za0ah97)
6. [Описание программы 6](#_hf8f16mswnpj)
7. [Рекомендации пользователя 7](#_5csnjnq374v4)
8. [Рекомендации программиста 8](#_737y2br7f8sz)
9. [Исходный код программы 8](#_uf6zz9ohkevw)
10. [Контрольный пример 8](#_c8cz8vpfb0z3)
11. [Вывод 10](#_93zc9b8jx91f)
12. [Источники 11](#_hx5r03ghpnnv)

# Цель работы

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

# 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

Метод симуляции отжига (Simulated Annealing) — это вероятностный алгоритм оптимизации, вдохновлённый физическим процессом охлаждения металлов. Алгоритм случайным образом изменяет текущее решение и принимает новое состояние с вероятностью, зависящей от изменения «энергии» (стоимости) и текущей температуры. Со временем температура уменьшается, что снижает вероятность принятия менее оптимальных решений.

Модификация отжига Коши (Cauchy Annealing) изменяет стандартную схему охлаждения, используя распределение Коши для генерации новых решений, что способствует более эффективному исследованию пространства решений и потенциально лучшему конечному результату.

# Основные шаги программы

1. Ввод пользователем списка вершин и матрицы смежности.
2. Запуск алгоритма симуляции отжига для поиска кратчайшего пути.
3. Опционально: включение модификации алгоритма с перебором начальной вершины для улучшения результата.
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

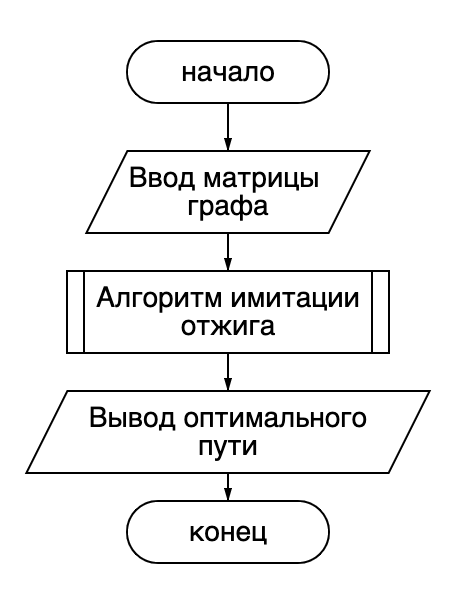


Рис 1. Блок-схема основной программы

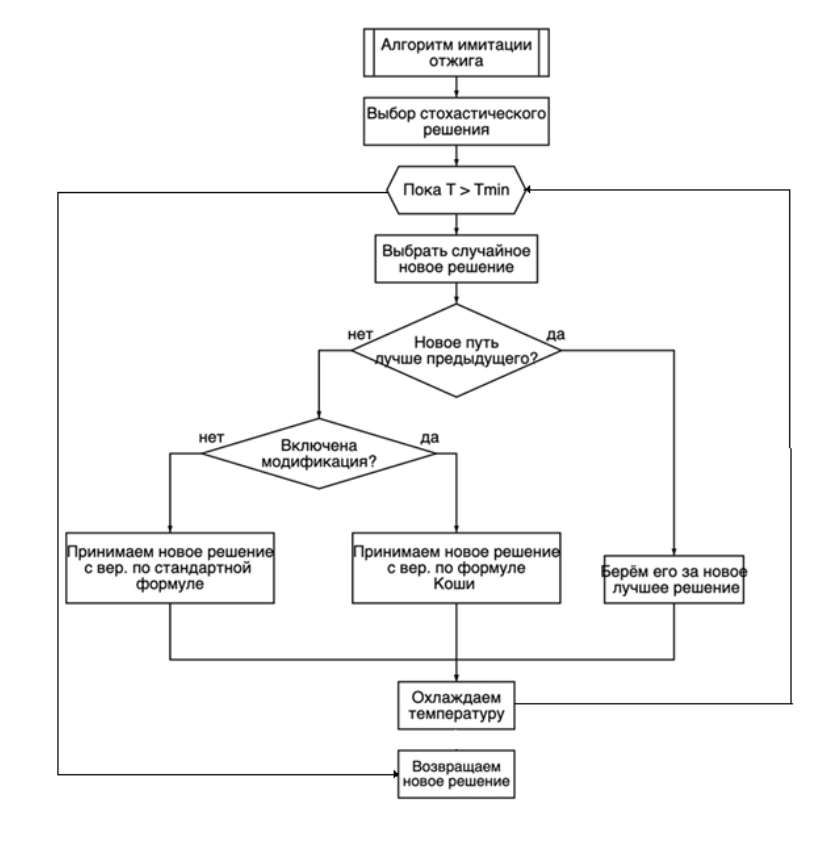


Рис 2. Блок-схема подпрограмм

# Описание программы

Программа реализована на языке Python с использованием библиотек PyQt5[[1]](#_hx5r03ghpnnv) для пользовательского интерфейса и NetworkX[[2]](#_hx5r03ghpnnv) 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 Коши.

Таблица 1. main.py

| Функция | Описание | Возвращаемое значение |
| --- | --- | --- |
| parse\_graph\_input | Считывание графа из поля ввода | None |
| simulated\_annealing\_tsp | Алгоритм симуляции отжига | 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]](#_hx5r03ghpnnv) и NetworkX[[2]](#_hx5r03ghpnnv).
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)

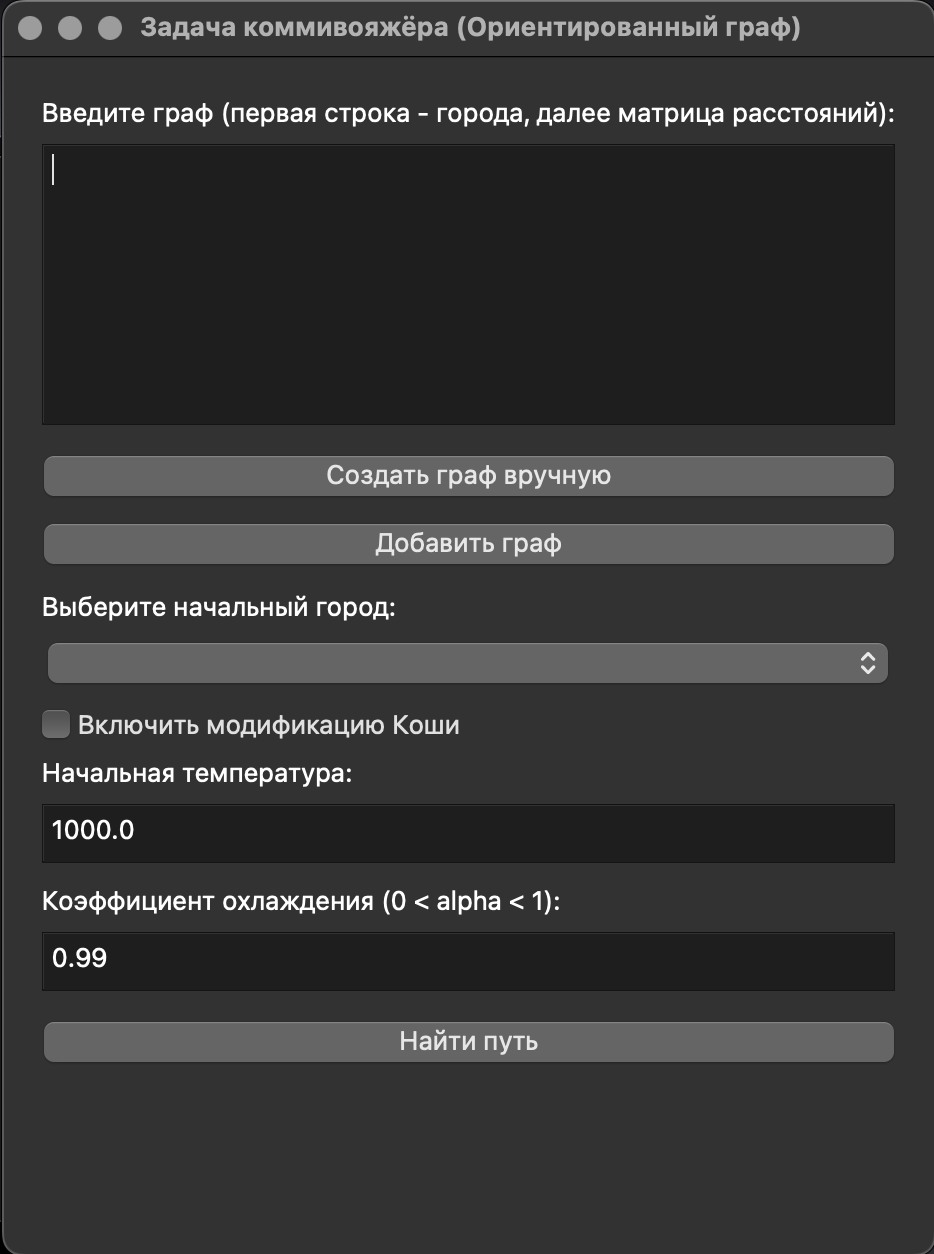
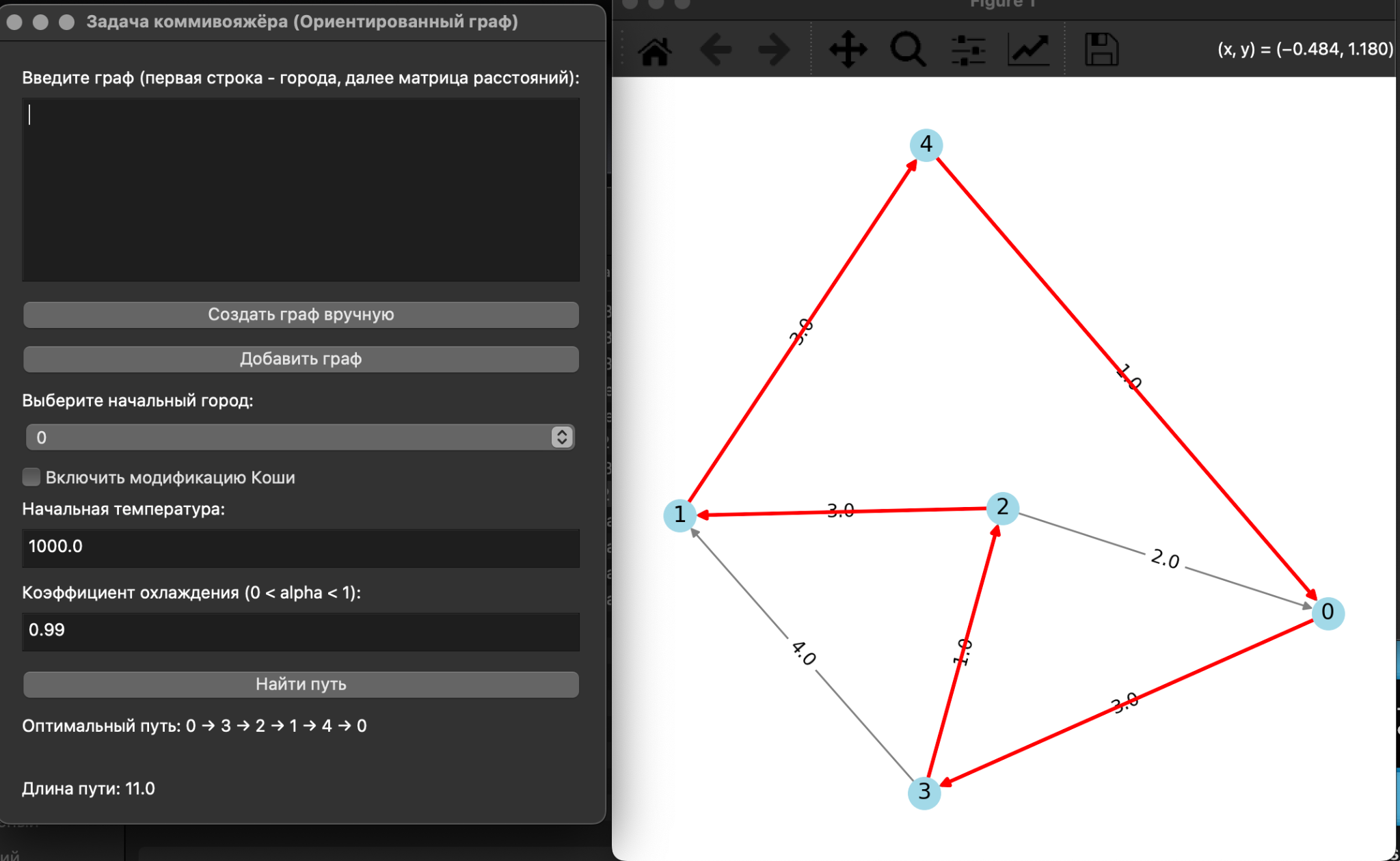


Рис. 3 (Ввод матрицы графа)

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

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)

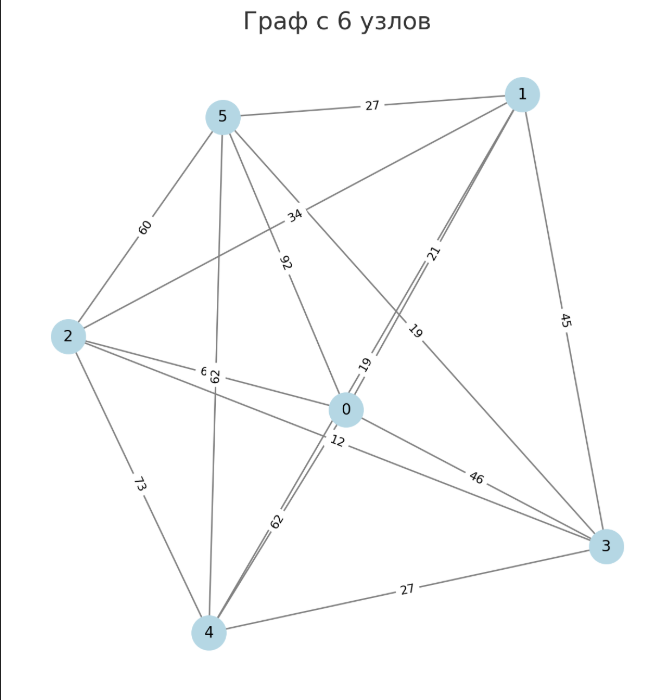


Рис. 5 (граф с 6 вершинами)

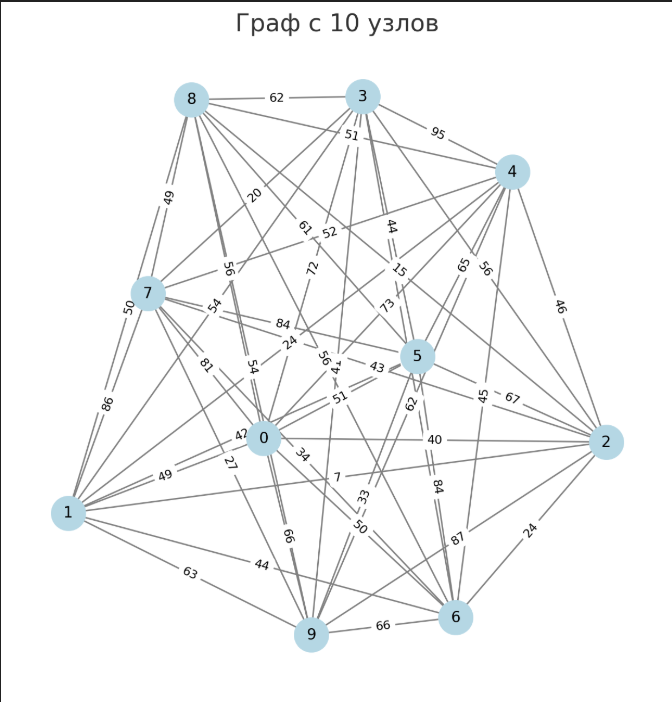


Рис. 6 (граф с 10 вершинами)

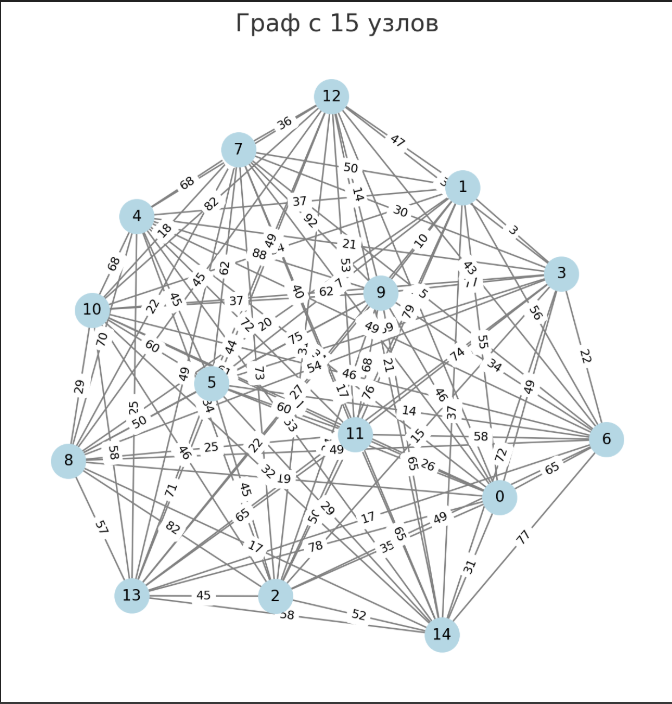


Рис. 7 (граф с 15 вершинами)

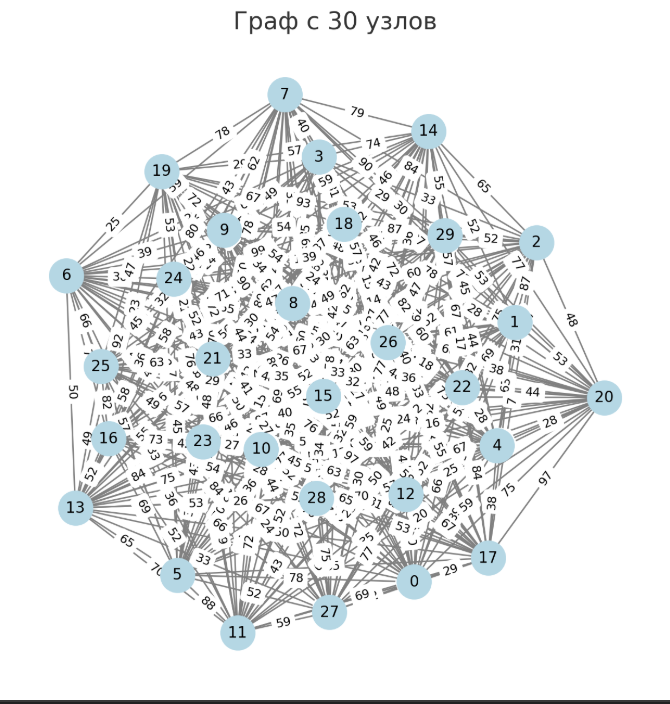


Рис. 8 (граф с 30 вершинами)

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 |

# Вывод

В ходе лабораторной работы были разработаны и протестированы два эвристических метода решения задачи коммивояжёра: алгоритм имитации отжига и алгоритм ближайшего соседа. Для каждого из них была также создана модификация с применением распределения Коши или другой модифицирующей эвристики.

1. **Алгоритм имитации отжига:**

Сравнение результатов (см. таблицу 2) показало, что модификация с использованием распределения Коши даёт преимущество по стоимости маршрута практически для всех размеров графа. Особенно заметно улучшение для графов размером 15 и 30: для графа из 15 узлов стоимость уменьшилась с 36,23 до 31,31; для графа из 30 узлов — с 54,34 до 48,08.

Время работы в обоих случаях осталось примерно одинаковым, разница не превышает нескольких тысячных долей секунды. Это позволяет утверждать, что модификация с использованием распределения Коши улучшает результат без значительного увеличения времени выполнения.

2. **Алгоритм ближайшего соседа:**

Согласно таблице 3, модификация также положительно влияет на стоимость маршрута: на графе из 30 узлов стоимость снижается с 60,32 до 56,48; на графе из 15 узлов — с 57,27 до 55,80.

Однако в отличие от имитации отжига, время работы с модификацией у ближайшего соседа увеличивается в разы (например, при размере 30: с 0,000298 до 0,00823 секунды). Тем не менее, абсолютное значение остаётся крайне малым, что делает прирост времени несущественным в большинстве случаев.

По качеству решения (стоимости маршрута) алгоритм имитации отжига стабильно показывает лучшие результаты по сравнению с алгоритмом ближайшего соседа, как в стандартной версии, так и с модификацией.

По времени выполнения алгоритм ближайшего соседа значительно быстрее, что делает его подходящим для задач с ограничениями по времени и допустимой погрешностью.

Модификации алгоритмов, в частности с использованием распределения Коши, вносят реальное улучшение в качество решений при незначительном увеличении времени выполнения.

В зависимости от задачи и требований к скорости и качеству можно выбрать подходящий алгоритм. Если требуется быстрое приближённое решение, то лучше использовать алгоритм ближайшего соседа. Если же критически важно качество маршрута, то предпочтительнее использовать имитацию отжига, особенно в модифицированной версии.

# Источники

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

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