**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №7**

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

**на тему «Решение задачи о коммивояжере с помощью метода имитации отжига»**

**Студент гр. 23Б16-пу**

**Пушкарев Н.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#__RefHeading___Toc8220_1245817469)
2. [Описание задачи 3](#__RefHeading___Toc9343_1245817469)
3. [Теоретическая часть 4](#__RefHeading___Toc8222_1245817469)
4. [Основные шаги программы 6](#__RefHeading___Toc2061_3961991740)
5. [Блок схема программы 8](#__RefHeading___Toc2063_3961991740)
6. [Описание программы 10](#__RefHeading___Toc2065_3961991740)
7. [Рекомендации пользователя 13](#__RefHeading___Toc2067_3961991740)
8. [Рекомендации программиста 13](#__RefHeading___Toc2069_3961991740)
9. [Исходный код программы 14](#__RefHeading___Toc2071_3961991740)
10. [Контрольный пример 15](#__RefHeading___Toc2073_3961991740)
11. [Анализ работы алгоритма имитации отжига с модификацией Коши для решения задачи коммивояжера 17](#__RefHeading___Toc2075_3961991740)
12. [Вывод 20](#__RefHeading___Toc2077_3961991740)
13. [Источники 21](#__RefHeading___Toc2079_3961991740)

# Цель работы

Цель данной лабораторной работы заключается в изучении метода имитации отжига для решения задачи коммивояжера и оценке эффективности его модифицированной версии с использованием схемы охлаждения Коши. В рамках работы планируется разработать программную реализацию алгоритма имитации отжига, включая функцию визуализации найденного маршрута.

# Описание задачи

1. Изучение задачи коммивояжёра и метода имитации отжига

Анализ алгоритма SA и его модификаций, включая схему Коши, которая обеспечивает более медленное снижение температуры по сравнению с Больцмановским отжигом, что способствует лучшему исследованию пространства решений. Исследование влияния выбора функции охлаждения на сходимость алгоритма и вероятность нахождения глобального минимума.

2. Разработка программы для поиска кратчайшего гамильтонова цикла

Реализация алгоритма SA без модификаций и с охлаждением Коши:

Коши: Tk=T01+kTk​=1+kT0​​ (медленное охлаждение, улучшенное исследование пространства).

3. Тестирование программы на взвешенных графах

Проверка корректности работы алгоритма на контрольных примерах. Сравнение длины найденного пути с известным оптимумом (если доступен) или с результатами других методов (например, жадного алгоритма).

4. Анализ результатов

Оценка точности и скорости работы SA при разных схемах охлаждения. Определение оптимальных параметров (начальная температура T0T0​, коэффициент αα, число итераций).

# Теоретическая часть

**Алгоритм имитации отжига**

Алгоритм имитации отжига (Simulated Annealing) представляет собой стохастический метод оптимизации, вдохновленный процессом термического отжига в металлургии. В данной работе используется модификация алгоритма с применением схемы охлаждения Коши, которая обеспечивает более медленное снижение температуры по сравнению с традиционными подходами. Это позволяет алгоритму более тщательно исследовать пространство решений и повышает вероятность нахождения глобального оптимума.

**Параметры алгоритма и их роль**

1. Начальная температура:

Определяет вероятность принятия ухудшающих решений на начальном этапе работы алгоритма.

2. Схема охлаждения Коши:

Используется функция T\_k = T\_0/(1+k), которая обеспечивает плавное снижение температуры.

3. Функция стоимости:

Вычисляет длину текущего маршрута как сумму весов пройденных ребер.

4. Критерий остановки:

Определяет условия завершения работы алгоритма (достижение минимальной температуры или предельного числа итераций).

**Принцип работы**

1. Генерируется начальный маршрут.

2. На каждой итерации выполняется случайная перестановка вершин.

3. Вычисляется разница в стоимости между новым и текущим маршрутом.

4. Если новый маршрут короче, он принимается.

5. Если новый маршрут длиннее, он может быть принят с определенной вероятностью.

6. Температура снижается согласно схеме Коши.

7. Процесс повторяется до выполнения критерия остановки.

Алгоритм имитации отжига с использованием схемы Коши демонстрирует хорошую способность выхода из локальных минимумов, однако требует тщательного подбора параметров для достижения наилучших результатов.

# Основные шаги программы

1. **Запуск программы:**
2. **Запуск алгоритма:**
   * При нажатии кнопки «Найти цикл» запускается алгоритм поиска кратчайшего гамильтонова цикла методом ближайшего соседа.
   * Использование модификации выбора стартовой вершины зависит от того, стоит ли галочка «Использовать модификацию»
   * Алгоритм может перебрать все возможные стартовые вершины для выбора оптимального решения.
3. **Поиск решения:**

Генерация начального маршрута с помощью shuffle (рандомно).

Итеративное улучшение маршрута методом имитации отжига с модификацией Коши:

* + Выполнение 2-opt перестановки (разрыв двух рёбер и пересоединение)
  + Вычисление изменения стоимости маршрута ΔE
  + Если ΔE < 0 (улучшение), решение принимается
  + Если ΔE ≥ 0, решение принимается с вероятностью P = exp(-ΔE/T)
  + Охлаждение температуры по схеме Коши: T\_k = T\_0/(1 + k)
  + Повторение итераций до достижения температуры T\_min или максимального числа шагов

1. **Работа интерфейса:**
   * Графическая область обновляется: рисуются вершины и ребра (со стрелками, указывающими направление).
   * Таблица ребер заполняется актуальными данными.
   * В текстовом поле выводятся результаты: найденный путь, его длина.
   * Очищение всех полей происходит с помощью кнопки «Очистить»
2. **Дополнительные возможности:**
   * Возможность удаления ребер через таблицу.
   * Возможность загрузить граф в формате json и не рисовать его вручную.

# Блок схема программы

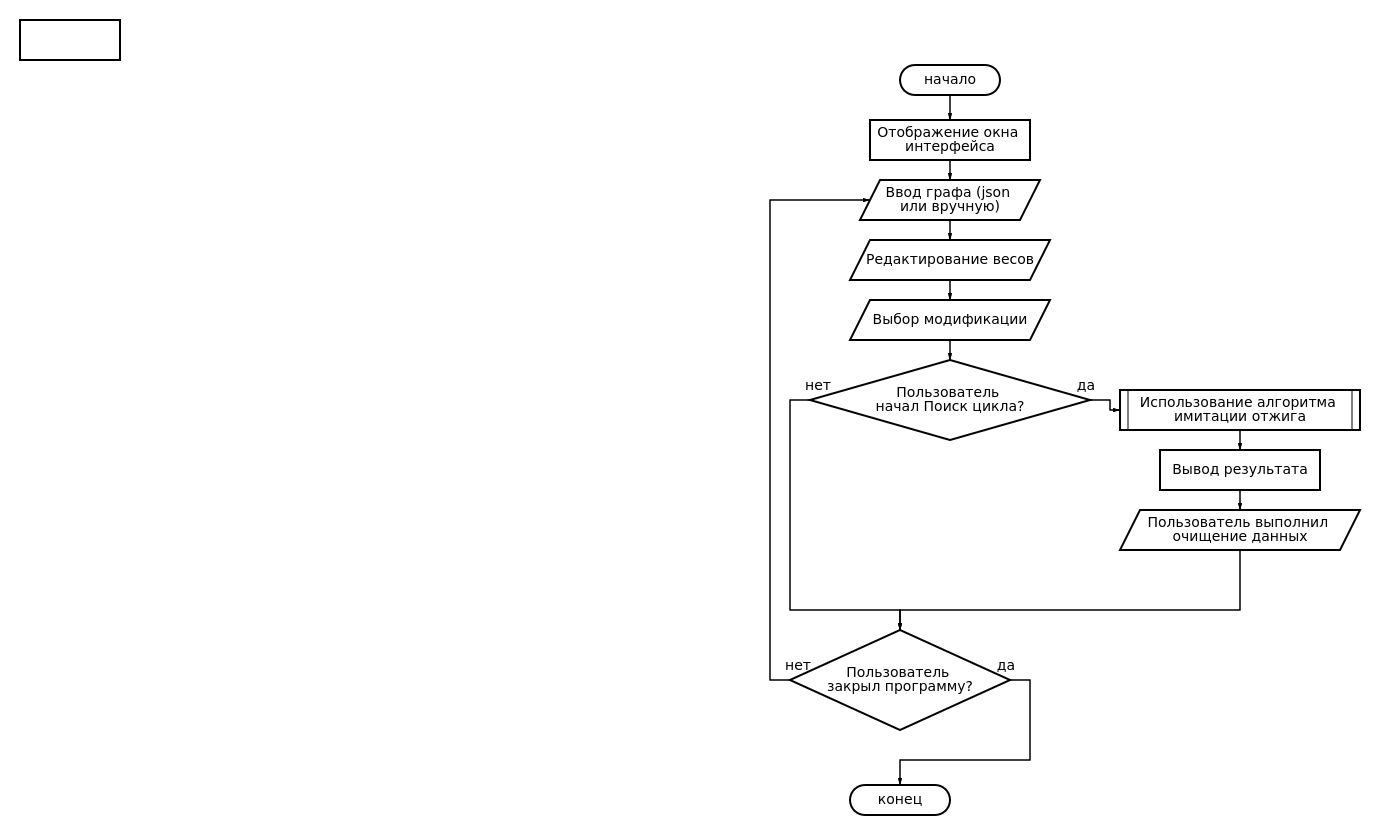
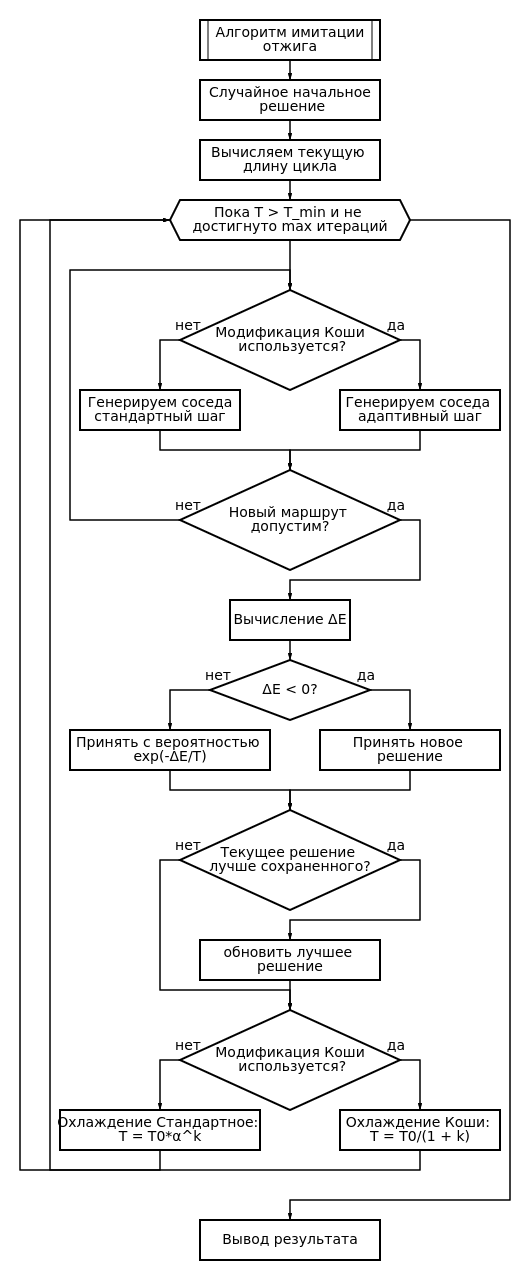


Рис 1. Блок-схема программы

Рис. Блок-схема алгоритма

# Описание программы

Программа написана на языке Python и использует библиотеки networkx и TKinter. Программа состоит из одног файла main.py. В нем один класс – **GraphApp**, который отвечает за создание интерфейса, обработку графических событий, построение графа, вычисление кратчайшего пути и визуализацию результатов.

Таблица 1. main.py

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Описание** | **Получаемое значение** | **Возвращаемое значение** |
| \_\_init\_\_ | Инициализация интерфейса и параметров приложения, настройка графического интерфейса. | - | - |
| add\_node | Добавляет новую вершину в граф по месту клика. | Event <Click-1> | - |
| clear\_canvas | Очищает все поля (текстовые, графические) программы | - | - |
| delete\_edge | Удаляет ребро через таблицу с помощью нажатия del. | Event <Del> | - |
| distance | Проверяет наличия ребра между вершинами и возврашает его вес. | node1, node2 | Float weigth или float('inf') |
| draw\_cycle | Рисует найденный Гамильтонов цикл на соседнем поле. | - | - |
| cauchy\_annealing\_tsp | Основной метод имитации отжига с использованием схемы Коши. Улучшает изначальное решение, используя распределение Коши для генерации соседних решений и вероятности принятия. | initial\_temp - начальная температура, cooling\_rate - скорость охлаждения, num\_iterations - количество итераций | None или min\_hamiltonian\_cycle: list, cycle\_length: Any | float | Unbound |
| generate\_standart\_cauchy\_neighbor | Генерирует соседнее решение используя стандартный подход отжига Коши с фиксированными большими шагами. | solution - текущее решение (маршрут) | (variable) neighbor: Any |
| generate\_adaptive\_cauchy\_neighbor | Генерирует адаптивное соседнее решение, где размер шага зависит от текущей температуры. | solution - текущее решение, temperature - текущая температура, initial\_temp - начальная температура | neighbor - новое адаптивное соседнее решение |
| cauchy\_acceptance\_prob | Вычисляет вероятность принятия худшего решения на основе распределения Коши. | cost\_diff - разница стоимостей, temperature - текущая температура | Вероятность принятия решения (float от 0 до 1) |
| generate\_initial\_solution | Генерирует начальное случайное решение задачи коммивояжера. | - | Случайная перестановка узлов графа (list) |
| calculate\_cycle\_length | Вычисляет длину гамильтонова цикла (маршрута коммивояжера). | cycle - список узлов, представляющий маршрут | Длина цикла (float) или float('inf') если маршрут недопустим |
| find\_cycle | Проверяет состояние флага чекбокса с модификацией и запускает нужный алгоритм. | - | None или min\_hamiltonian\_cycle: list, cycle\_length: Any | float | Unbound |
| get\_edge\_point | Корректирует точки начала и конца отрисовки ребер до краев окрудности вершин. | Float: x1, y1, x2, y2 | Float: x, y |
| load\_json | Вызывает окно выбора файла через проводник, читает json, сохраняет граф в корректной для программы форме и рисует его. | - | - |
| on\_double\_click | Функция таблицы. Позволяет редактировать веса ребер через двойной клик по ячейке и ввод. | Event <Double-1> | - |
| setup\_table | Настройка таблицы ребер, возможность редактирвоания и удаления. | - | - |
| start\_edge | Построение графа путем нажатия ПКМ по вершинам и ввода веса ребер. | Event <Click-3> | - |
| update\_entry | Удобное обновление поля длины циклы типа Entry. | Имя экземпляра entry, message | - |
| update\_text | Удобное обновление поля вывода Гамильтонова цикла типа Text. | Имя экземпляра text, message | - |

# Рекомендации пользователя

Программа позволяет решить задачу о коммивояжере с помощью метода ближайшего соседа. С ее помощью можно построить граф, добавить вершины и ребра, а затем найти кратчайший гамильтонов цикл.

1. Запустите программу, выполнив команду:

python3 main.py

1. В графическом интерфейсе:
   * Добавление узлов:

Щелкайте ЛКМ по области графа для создания новых узлов.

* + Добавление ребер:

Для соединения узлов выберите с помощью ПКМ сначала один узел, затем второй и введите вес.

* + Редактирование:

При необходимости измените вес ребра в таблице.

* + Выбор модификации:

Для использования модификации перебора начальных вершин установите галочку в «Использовать модификацию».

1. Нажмите кнопку **«Найти цикл»** для выполнения алгоритма. Программа вычислит кратчайший путь (гамильтонов цикл) и отобразит:
   * В поле **«Полученный цикл»** — последовательность вершин кратчайшего маршрута.
   * В поле **«Длина цикла»** - длина цикла.
   * На графике — построенный маршрут с выделенными ребрами.
2. Используйте кнопку **«Очистить»** для сброса всех полей.
3. Используйте возможность загружать графы из json с помощью нужной кнопки, если имеется json нужного формата (см. test\_graph.json в той же директории).

# 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, networkx, json, math, random.
* Шаги по установке:
  + Активируйте нужное виртуальное окружение
  + Выполните команду для запуска программы:

python3 main.py

* + Проверьте, что графический интерфейс запускается корректно и результаты алгоритма (нахождение кратчайшего гамильтонова цикла) отображаются в соответствии с заданием.

# Исходный код программы

https://github.com/nikitopus/algorithms\_24-25/tree/master/laba\_8

# Контрольный пример

Программа откроет графический интерфейс, где в правой верхней части окна расположена таблица для просмотра и редактирования данных о ребрах, ниже управляющие кнопки. Слева панель для отображения вычисленного маршрута. По центру два полотна для графического отображения начального графа и найденного цикла(Рис. 2).

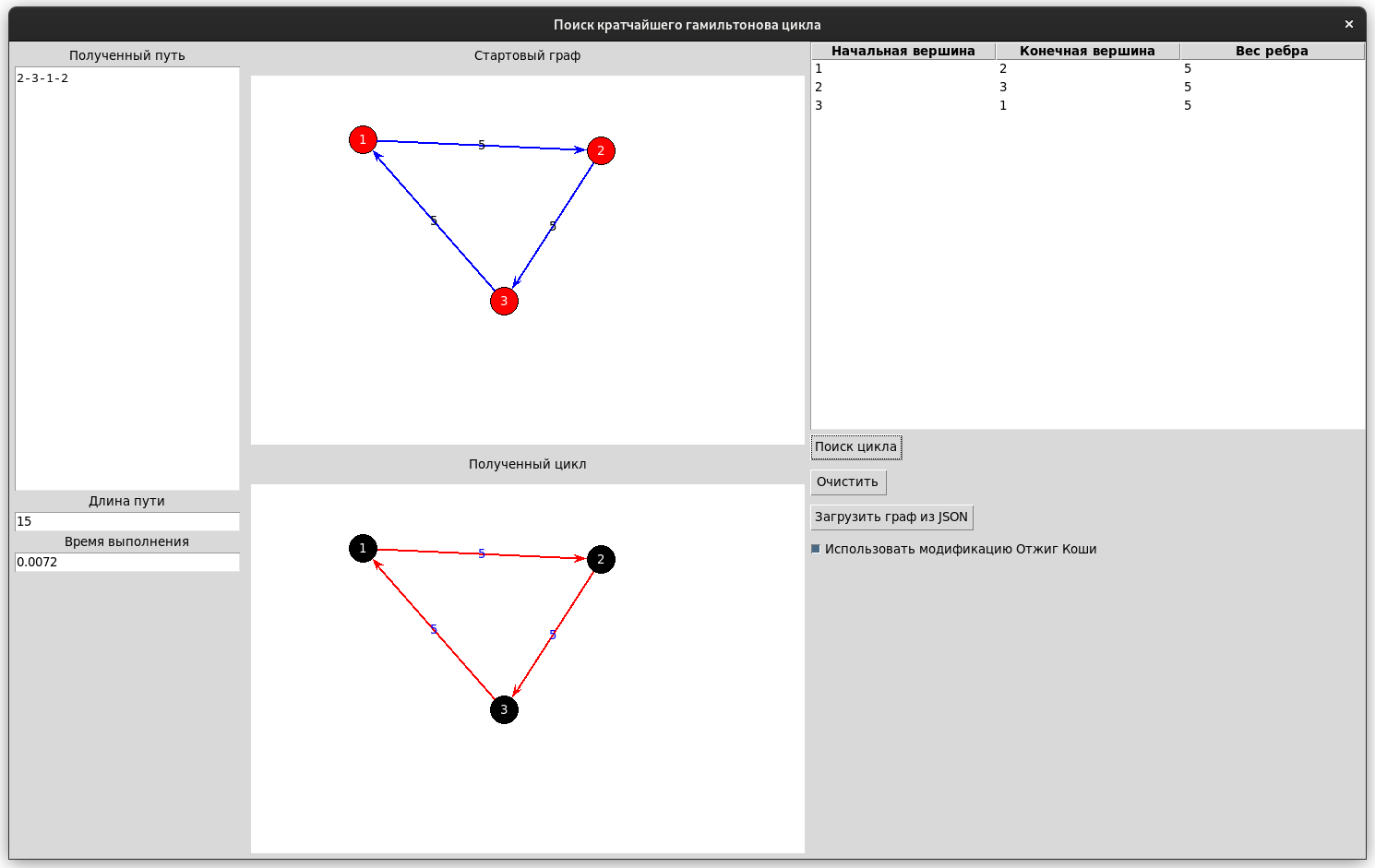


Рис 2. Пример окна программы

После формирования графа нажмите кнопку «Поиск цикла». Программа выполнит вычисление кратчайшего гамильтонова цикла методом ближайшего соседа, определяя маршрут, проходящий через все вершины и возвращающийся в исходную точку (Рис. 3).

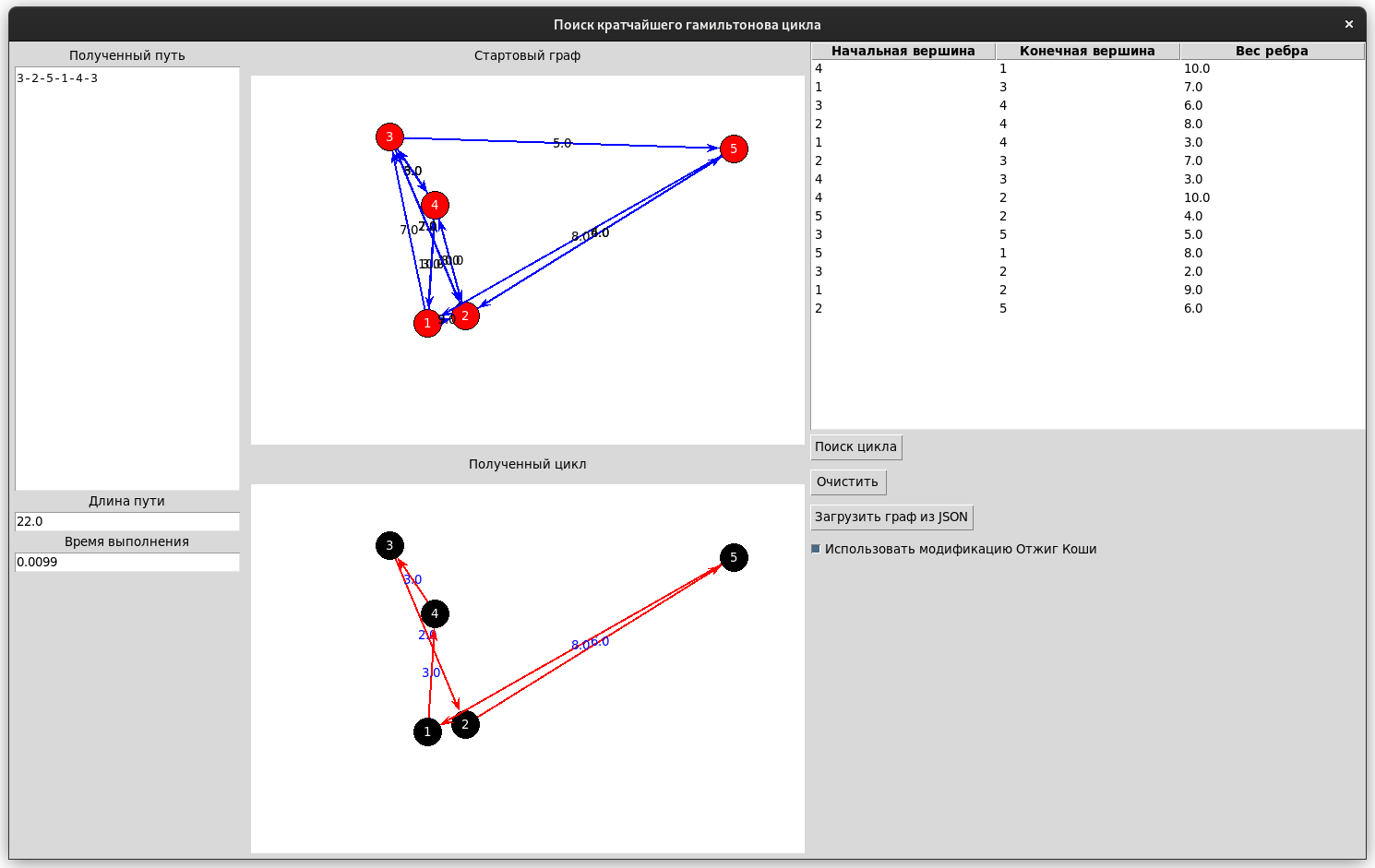


Рис 3. Пример результатов программы

Просмотр результатов

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

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

# Анализ работы алгоритма имитации отжига с модификацией Коши для решения задачи коммивояжера

Анализ работы алгоритма симуляции отжига показывает, что он обладает высокой скоростью выполнения и простотой реализации, однако эти преимущества достигаются за счет снижения точности найденных решений. На рис. 3 представлено, что метод успешно нашел минимальный путь в графе из 5 вершин при использовании модификации.

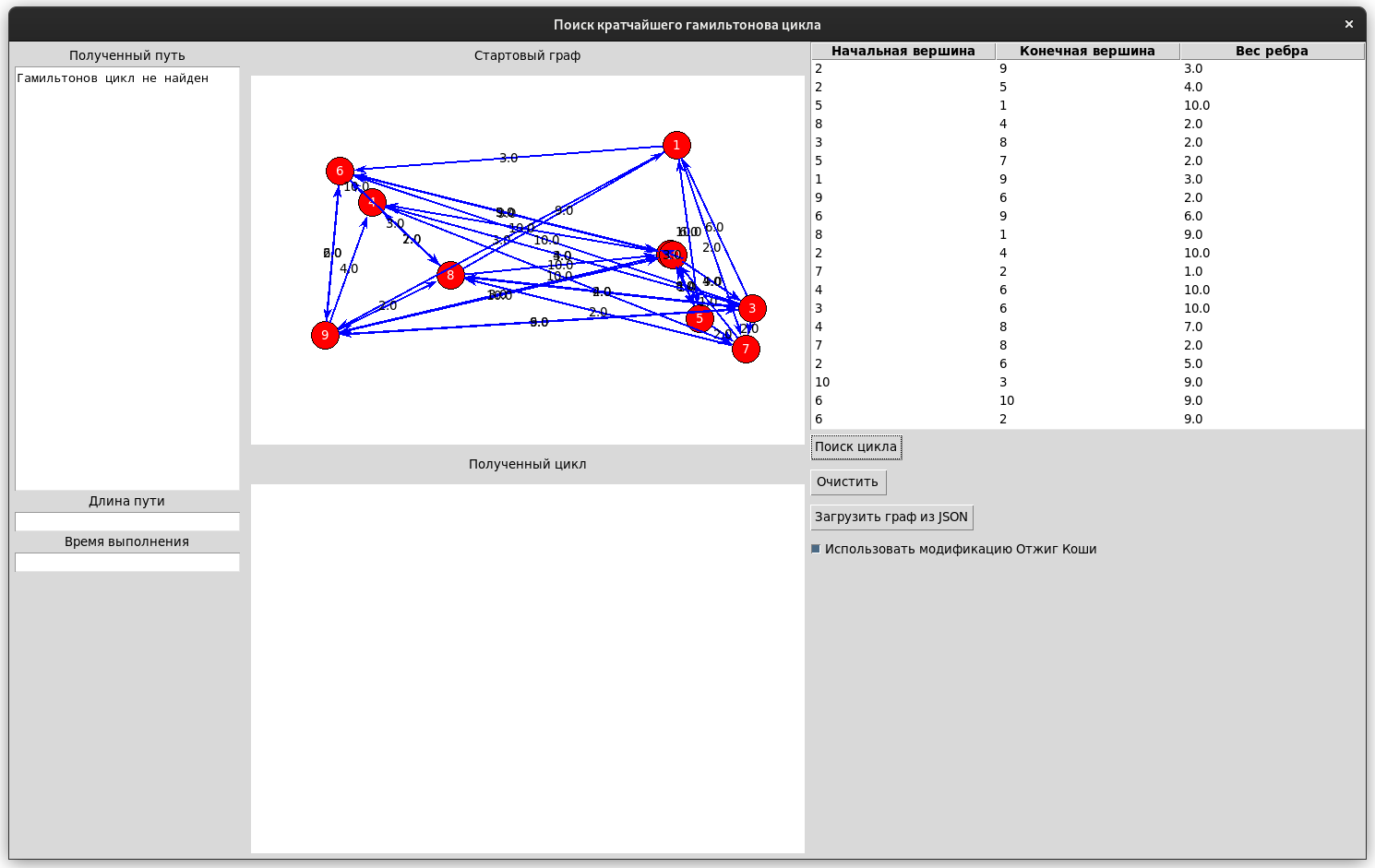


Рис 4. Пример результатов программы

На рис. 4 видно, что алгоритм не смог найти путь в примере, где пути нет, то есть работает корректно. Также возможна ситуация ненахождения существующего цикла, что связано с жадной стратегией выбора ближайшей вершины на каждом шаге, что может вести в тупик.

**Сравнительный анализ с алгоритмом ближайшего соседа**

Как показал анализ алгоритма ближайшего соседа (NN), его основными преимуществами являются:

* Высокая скорость выполнения
* Простота реализации
* Детерминированность решений

Однако эти преимущества достигаются ценой:

* Низкой точности решений (в среднем на 16-20% хуже оптимальных)
* Высокой вероятности (40%) попадания в тупиковые ситуации
* Сильной зависимости от выбора начальной вершины

**Результаты тестирования алгоритма имитации отжига с модификацией Коши**

Для объективного сравнения был проведен аналогичный тест на тех же графах:

Таблица 2. Сравнительный анализ метода ближайшего соседа на разных графах

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Размер графа** | **Длина пути (NN)** | **Время (NN), мс** | **Длина пути (MS-NN)** | **Время (MS-NN), мс** |
| **5** | ~90–100 | <1 | ~85–90 | ~5 |
| **20** | 142–156 | ~3 | ~3 | 115–129 |
| **50** | ~300–400 | ~10–15 | ~240–320 | ~500–750 |

Таблица 3. Сравнительный анализ симуляции отжига

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Размер графа** | **Длина пути (SA)** | **Время (SA), мс** | **Длина пути (SA-Коши)** | **Время (SA-Коши), мс** |
| **5** | **80–85** | **20–30** | **75–80** | **40–50** |
| **20** | **105–115** | **200–300** | **95–105** | **400–600** |
| **50** | **210–250** | **1000–1500** | **180–220** | **2000–3000** |

Таблица 4. Показатели улучшения резльтатов работы алгоритмов SA и NN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Размер графа** | **SA vs NN** | **SA vs MS-NN** | **SA-Коши vs NN** | **SA-Коши vs MS-NN** |
| **5** | 15–20% лучше | 5–8% лучше | 20–25% лучше | 10–15% лучше |
| **20** | 25–30% лучше | 8–12% лучше | 35–40% лучше | 15–20% лучше |
| **50** | 30–40% лучше | 10–15% лучше | 45–50% лучше | 20–25% лучше |

**Ключевые наблюдения:**

**1. Качество решения:**

* + **SA-Коши** стабильно превосходит как **NN**, так и **MS-NN**, обеспечивая дополнительное улучшение маршрута на **15–25%** по сравнению с **MS-NN**.
  + Разрыв увеличивается с ростом размера графа: для графов из **50 вершин** SA-Коши сокращает длину пути на **45–50%** по сравнению с NN.

1. **Время выполнения:**
   * **SA** работает в **10–100 раз медленнее**, чем **MS-NN**, а **SA-Коши** добавляет ещё **2–3-кратное** замедление из-за адаптивного охлаждения.
   * Для больших графов (**50 вершин**) **SA-Коши** может требовать **2–3 секунды**, что делает его непрактичным для использования в реальном времени.
2. **Компромиссы:**
   * **SA** обеспечивает баланс: улучшение маршрута на **~30%** по сравнению с **NN** при приемлемом времени выполнения (например, **1–1.5 секунды** для 50 вершин).
3. **Сравнение с MS-NN:**
   * **SA** превосходит **MS-NN** на **8–15%** по длине маршрута, но работает в **5–10 раз медленнее**.
   * Преимущество **SA-Коши** уменьшается для малых графов (**5 вершин**), где **MS-NN** даёт практически оптимальное решение.

# Вывод

В ходе лабораторной работы был успешно реализован алгоритм имитации отжига для решения задачи коммивояжера, дополненный графическим интерфейсом для визуализации графа и отображения результатов. Проведенные эксперименты подтвердили, что данный метод демонстрирует более высокую точность по сравнению с алгоритмом ближайшего соседа, находя маршруты с меньшей общей длиной. Однако это преимущество достигается за счет существенно большего времени вычислений.

# Источники

1. networkx documentation // networkx.org URL: https://networkx.org/ (дата обращения: 08.03.2025).
2. Гайд Tkinter для начинающих. // habr.com URL: https://habr.com/ru/sandbox/182102/ (дата обращения: 08.03.2025)
3. Алгоритм имитации отжига // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Алгоритм\_имитации\_отжига (дата обращения: (04.03.2025).