**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 алгоритма имитации отжига**»

**Студент гр. 23Б15-пу**

**Черевко М. Е.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc193498872)
2. [Описание задачи (формализация задачи) 4](#_Toc193498873)
3. [Теоретическая часть 6](#_Toc193498874)
4. [Основные шаги программы 9](#_Toc193498875)
5. [Блок схема программы 13](#_Toc193498876)
6. [Описание программы 15](#_Toc193498877)
7. [Рекомендации пользователя 16](#_Toc193498878)
8. [Рекомендации программиста 17](#_Toc193498879)
9. [Контрольный пример 18](#_Toc193498880)
10. [Анализ 20](#_Toc193498881)
11. [Вывод 22](#_Toc193498882)
12. [Источники 23](#_Toc193498883)

# Цель работы

Разработка программного приложения с графическим интерфейсом для решения задачи коммивояжёра (Traveling Salesman Problem, TSP). Приложение должно обеспечивать 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)

В данной лабораторной работе необходимо исследовать эффективность метода имитации отжига 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 алгоритма имитации отжига и его модификации: Отжига Коши.
* Визуализировать процесс построения маршрута.
* Провести анализ качества решения, сравнив его с оптимальным маршрутом.

В качестве модификации алгоритма имитации отжига используется отжиг Коши, отличающийся от имитации отжига тем, что вероятность перехода рассчитывается иначе: P = 1/(1+delta/T). Кроме того, в этой модификации температура меняется медленнее что позволяет более тщательно искать лучшее решение. Формула изменения температуры: T = T0/(iterations)

# Теоретическая часть

* **Введение**

Задача коммивояжера (Traveling Salesman Problem, TSP) заключается в нахождении кратчайшего замкнутого маршрута, проходящего через заданный набор узлов ровно один раз с возвращением в начальную точку. TSP относится к классу NP-трудных задач, что делает точные методы решения непрактичными для больших графов. Для её решения применяются метаэвристические алгоритмы, такие как имитация отжига (Simulated Annealing, SA), реализованный в данном коде. Алгоритм SA позволяет находить близкие к оптимальным решения, избегая локальных минимумов за счет вероятностного подхода.

* **Имитация отжига**

Имитация отжига вдохновлена физическим процессом охлаждения металлов, при котором материал медленно охлаждается для достижения состояния с минимальной энергией. В оптимизации SA ищет глобальный минимум целевой функции (в TSP — общей стоимости маршрута) через следующие шаги:

1. Текущее решение: На каждой итерации рассматривается текущий маршрут, который может быть улучшен.
2. Генерация соседей: Новые маршруты создаются путем небольших изменений текущего, например, перестановки двух узлов.
3. Критерий принятия: Новое решение принимается, если оно уменьшает стоимость маршрута. Худшие решения могут приниматься с вероятностью, зависящей от температуры:

где — разница стоимостей (новая минус текущая), — текущая температура.

1. Охлаждение: Температура постепенно снижается, уменьшая вероятность принятия худших решений и направляя алгоритм к сходимости.

* **Реализация алгоритма**

Алгоритм SA в коде решает TSP для графа с заданными узлами и весами ребер. Основные этапы:

1. **Инициализация:** Формируется случайный маршрут, вычисляется его стоимость. Начальная температура коэффициент охлаждения
2. **Поиск начального маршрута**: Генерируются случайные перестановки узлов, пока не найдется маршрут с конечной стоимостью (все ребра существуют).
3. **Основной цикл:**

* Создается новый маршрут перестановкой двух узлов.
* Новая стоимость сравнивается с текущей. Если она ниже или удовлетворяет критерию принятия, маршрут обновляется.
* Лучший найденный маршрут и его стоимость сохраняются.
* Температура снижается по геометрическому закону или в случае модификации по распределению Коши — номер итерации.
* **Параметры и их роль**
* : Высокое значение позволяет исследовать разнообразные маршруты на старте.
* : Медленное охлаждение способствует тщательному поиску минимума.
* : Быстрое снижение температуры ускоряет сходимость, но может упустить оптимальное решение.
* Адаптированы к размеру графа для баланса между качеством и временем.

# Основные шаги программы

Программа, реализованная в виде класса TSPApp с использованием библиотеки Tkinter, представляет собой приложение для решения задачи коммивояжёра. Ниже приведены основные шаги её работы, соответствующие структуре кода и функциональности:

1. **Инициализация приложения**

* **Цель:** Подготовка графического интерфейса и начальных данных.
* **Реализация:** Метод \_\_init\_\_:
  1. Создаётся главное окно приложения с заголовком "Решение задачи маршрута" и размером 1000x750 пикселей.
  2. Инициализируются списки для хранения узлов (self.nodes), связей (self.connections), истории действий (self.history) и удалённых узлов (self.deleted\_nodes).
  3. Устанавливаются начальные значения: node\_id\_tracker = 0, active\_node = None, флаг модификации (use\_modification\_var = True).
  4. Вызывается метод \_setup\_ui для создания интерфейса и \_center\_window для центрирования окна на экране.

1. **Создание пользовательского интерфейса**

* **Цель:** Обеспечение удобного ввода данных и управления программой.
* **Реализация:** Метод \_setup\_ui:
  1. Создаются два основных холста: input\_area (ввод графа) и output\_area (отображение результата).
  2. Формируется таблица связей (edge\_table) с колонками "Начало", "Конец", "Вес".
  3. Добавляются элементы управления: кнопки "Найти маршрут", "Назад", "Сбросить", "Загрузить граф" и переключатель модификации.
  4. Устанавливаются привязки событий: клик левой кнопкой мыши для добавления узлов, правой — для создания связей.

1. **Добавление узлов**

* **Цель:** Создание вершин графа пользователем.
* **Реализация:** Метод \_place\_node:
  1. При клике левой кнопкой мыши на холсте input\_area проверяется минимальное расстояние до существующих узлов (MIN\_SPACING).
  2. Если расстояние достаточно, создаётся новый узел с уникальным идентификатором (переиспользуется из deleted\_nodes или увеличивается node\_id\_tracker).
  3. Узел рисуется как зелёная окружность с белой меткой идентификатора.
  4. Информация об узле добавляется в self.nodes и self.history.

1. **Создание связей**

* **Цель:** Установка направленных рёбер между узлами с заданными весами.
* **Реализация:** Метод \_pick\_node\_for\_link:
  1. При клике правой кнопкой мыши выбирается первый узел (если active\_node пуст), отображается временная метка.
  2. При повторном клике на другом узле запрашивается вес связи через диалоговое окно (askinteger).
  3. Если связь уже существует, предлагается изменить вес; иначе создаётся новая связь.
  4. Связь отображается стрелкой (\_render\_directed\_link), добавляется в self.connections, self.edge\_table и self.history.

1. **Загрузка графа из файла**

* **Цель:** Импорт графа из текстового файла.
* **Реализация:** Метод \_load\_graph:
  1. Открывается диалог выбора файла (filedialog.askopenfilename).
  2. Файл читается построчно: узлы (формат: id,x,y) и рёбра (формат: from\_id,to\_id,weight) разделяются секциями # Nodes и # Edges.
  3. Узлы и связи отрисовываются на холсте, добавляются в соответствующие структуры данных.
  4. Обрабатываются ошибки (дубликаты узлов, некорректный формат) с выводом сообщений через messagebox.

1. **Решение задачи коммивояжера**

* **При нажатии кнопки "Найти маршрут" запускается процесс решения задачи коммивояжера.**
* **Формируется граф на основе списка узлов и связей, где каждый узел представлен идентификатором, а связи — весами ребер.**
* **Создается словарь содержащий информацию о доступных ребрах и их весах для всех узлов.**
* **Вызывается алгоритм имитации отжига для поиска оптимального маршрута:**

1. **Генерируется случайный начальный маршрут путем перестановки узлов.**
2. **Проверяется допустимость маршрута (все ребра существуют, стоимость конечна).**
3. **В цикле итераций:**
4. **Создается новый маршрут перестановкой двух случайных узлов.**
5. **Вычисляется стоимость нового маршрута как сумма весов ребер, включая возврат в начальную точку.**
6. **Новый маршрут принимается, если его стоимость ниже текущей или удовлетворяет вероятностному критерию — разница стоимостей, — температура.**
7. **Температура снижается по геометрическому закону или по распределению Коши , в зависимости от выбранного режима.**
8. **Сохраняется лучший найденный маршрут и его стоимость.**

* **Результат (маршрут, стоимость, время выполнения) отображается в текстовом виде в интерфейсе.**
* **Если маршрут существует, он визуализируется на холсте; в противном случае выводится сообщение об ошибке.**

1. **Визуализация результата**

* **Цель:** Отображение найденного маршрута.
* **Реализация:** Метод \_display\_optimal\_route:
  1. Очищается холст output\_area.
  2. Рисуются узлы из маршрута как зелёные окружности с метками.
  3. Рисуются связи между последовательными узлами красными стрелками (\_render\_link\_on\_output).
  4. Добавляется финальная связь от последнего узла к первому.

1. **Отмена последнего действия**

* **Цель:** Возврат к предыдущему состоянию графа.
* **Реализация:** Метод \_revert\_last\_step:
  1. Извлекается последнее действие из self.history.
  2. Для "node\_added": удаляется узел, его фигуры стираются, идентификатор добавляется в deleted\_nodes.
  3. Для "link\_added": удаляется связь из self.connections и edge\_table, стирается стрелка.
  4. Для "link\_updated": восстанавливается старый вес связи.

1. **Сброс всех данных**

* **Цель:** Очистка графа и возврат к начальному состоянию.
* **Реализация:** Метод \_reset\_all:
  1. Удаляются все элементы из edge\_table, input\_area, output\_area, result\_container.
  2. Очищаются списки nodes, connections, сбрасываются node\_id\_tracker и active\_node.
  3. Отображается сообщение "Очищено".

1. **Сохранение результатов**

* **Цель:** Сохраняет найденный маршрут в файл .txt
* **Реализация**: **Метод** \_save\_result
  1. Проверяет, есть ли решение (если нет — ругается)
  2. Показывает окно выбора папки
  3. Пишет в файл: координаты точек, связи между ним, итоговую длину маршрута
  4. Отчитывается: сохранилось или ошибка

1. **Запуск программы**

* **Цель:** Инициализация и запуск основного цикла приложения.
* **Реализация:** Блок if \_\_name\_\_ == "\_\_main\_\_":
  + Создаётся экземпляр Tk, инициализируется TSPApp, запускается mainloop для обработки событий.

# Блок схема программы

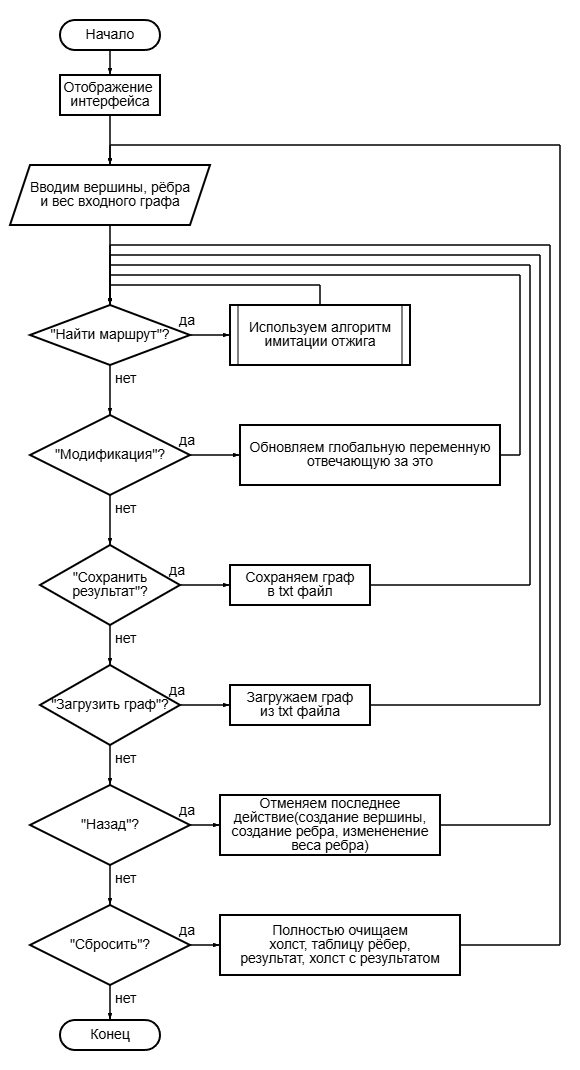


Рис. 3 Блок-схема основной программы

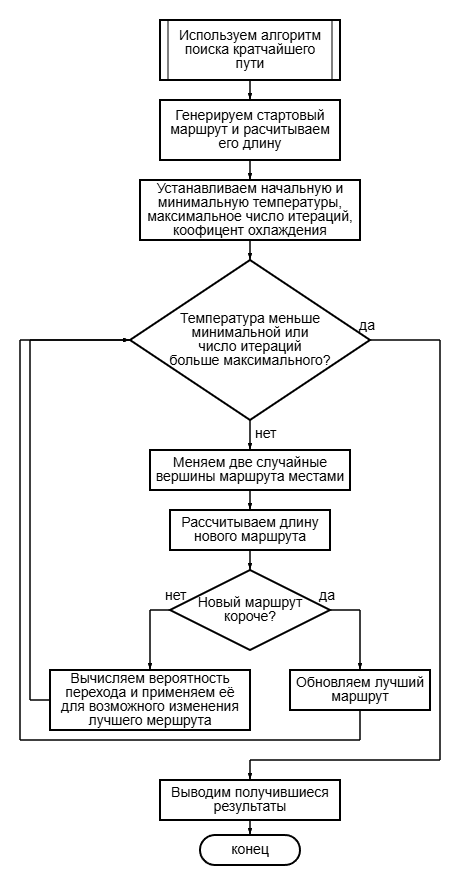


Рис. 4 Блок-схема подпрограммы

# Описание программы

Программная реализация задачи коммивояжёра разработана на языке Python 3.12.7 с применением библиотек tkinter, ttk и math. Приложение представляет собой графический интерфейс, позволяющий пользователю формировать и изменять граф, состоящий из узлов и связей. На основе введённой информации программа решает задачу коммивояжёра, определяя локальный кратчайший маршрут. В интерфейсе реализованы функции добавления узлов, создания рёбер между ними с указанием весов, а также возможности отмены последнего шага и полного сброса данных. Программа визуализирует результат, отображая оптимальный маршрут и его общую стоимость.

Таблица 1. Functions

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| \_\_init\_\_ | Инициализирует приложение, создавая интерфейс и переменные. | None |
| \_setup\_ui | Создает пользовательский интерфейс (окна, кнопки, холсты) | None |
| \_center\_window | Центрирует главное окно на экране | None |
| \_load\_graph | Рисует кратчайший путь, найденный для задачи коммивояжёра. | None |
| \_place\_node | Добавляет новый узел на холст по клику мыши | None |
| \_pick\_node\_for\_link | Выбирает узлы для связи (по правому клику) | None |
| \_render\_directed\_link | Очищает все данные (все вершины и рёбра), сбрасывает интерфейс. | None |
| \_display\_optimal\_route | Показывает оптимальный маршрут на выходном холсте | None |
| \_render\_link\_on\_output | Отрисовывает связь между узлами на выходном холсте | None |
| calculate\_tour\_cost | Вычисляет общую стоимость маршрута, включая возврат в начальную точку | Стоимость маршрута (число или ∞) |
| perturb\_tour | Генерирует новый маршрут, меняя местами два случайных узла | Новый маршрут (список узлов) |
| simulated\_annealing | Выполняет алгоритм имитации отжига для поиска оптимального маршрута | Кортеж: оптимальный маршрут, его стоимость, время выполнения |
| \_solve\_tsp | Решает задачу коммивояжера (TSP) и отображает результат | None |
| \_save\_result | Сохраняет найденный маршрут в файл | None |
| \_revert\_last\_step | Отменяет последнее действие (добавление узла/связи) | None |
| \_reset\_all | Полностью сбрасывает приложение в исходное состояние | None |

# 

# Рекомендации пользователя

Для корректного выполнения программы следуйте приведённым шагам:

* **Добавление вершин**:

Для добавления вершины на граф, кликните левой кнопкой мыши на поле "Граф ввода". 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.

* **Отмена последнего действия**:

Для отмены последнего действия (добавление вершины или рёбра) используйте кнопку "Назад". Это позволит вернуть граф в состояние до последнего изменения.

* **Очистка графа**:

Для очистки всего графа (удаления всех вершин и рёбер) нажмите кнопку "Сбросить". После этого граф будет полностью очищен, и вы можете начать с нуля.

* **Расчёт оптимального пути**:

После добавления всех вершин и рёбер, нажмите кнопку "Рассчитать". Программа найдет оптимальный путь для задачи коммивояжера и отобразит результат на поле "Граф результата". На поле также будет показан вес маршрута, а результаты расчета будут отображены в блоке "Результат". Здесь будет показан оптимальный путь и общий вес маршрута.

# 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 **3.7** 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/FraaaM/Solving\_the\_Traveling\_Salesman\_Problem\_Using\_an\_Annealing\_Simulation\_Algorithm**](https://github.com/FraaaM/Solving_the_Traveling_Salesman_Problem_Using_an_Annealing_Simulation_Algorithm)

# Контрольный пример

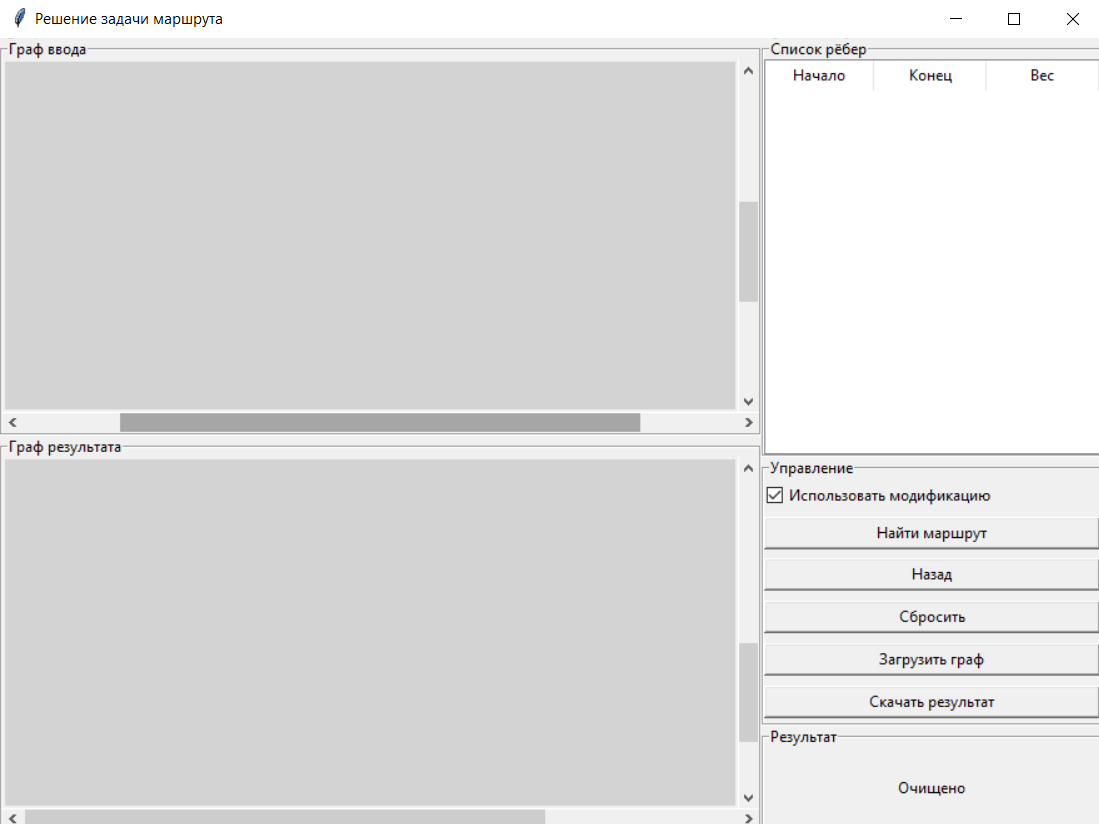
* **Запуск программы:** Для запуска программы используйте файл code.py. Программа откроет графический интерфейс (Рис. 4).

Рис. 3 Графический интерфейс программы

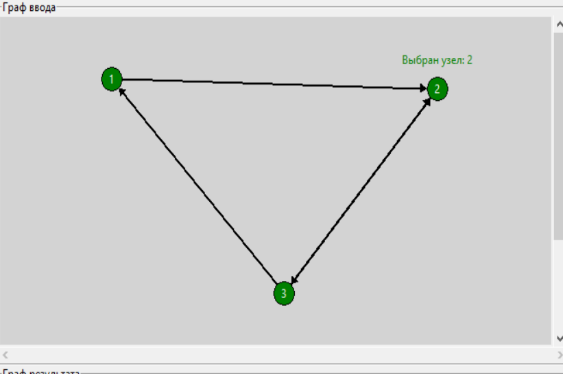
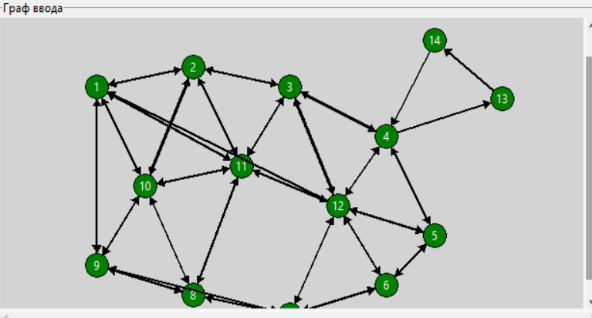
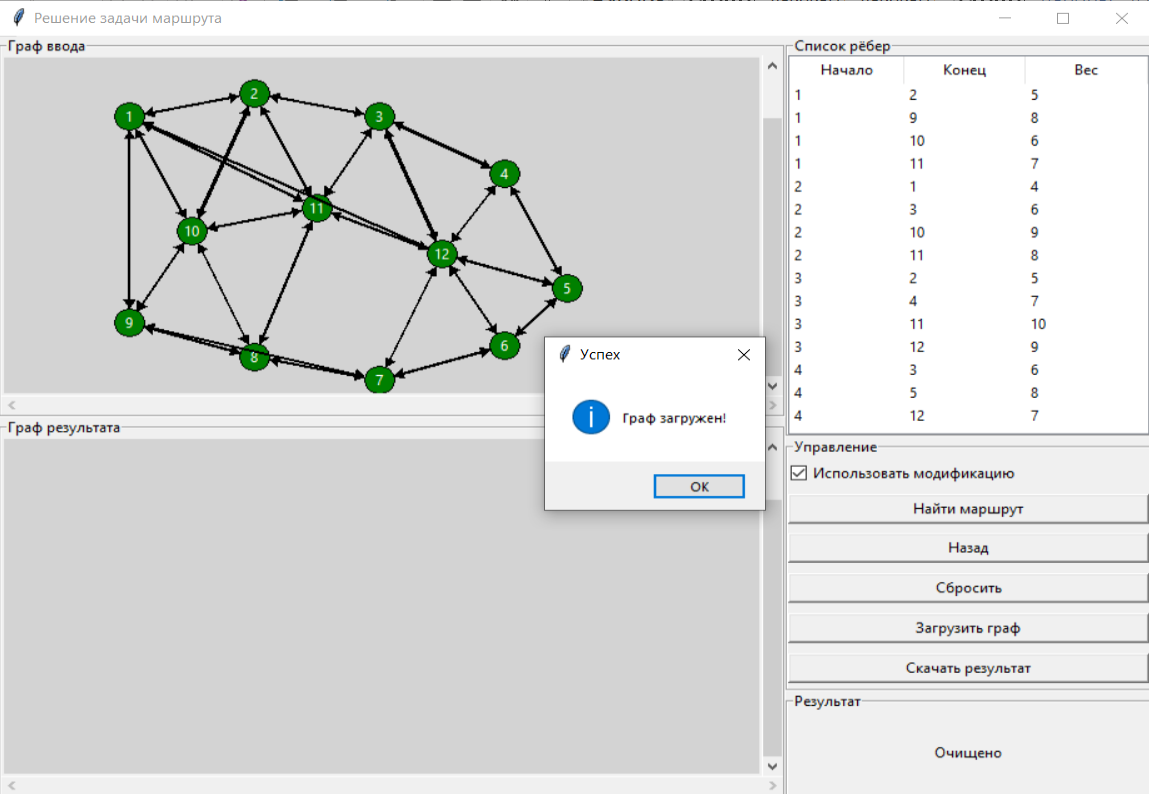
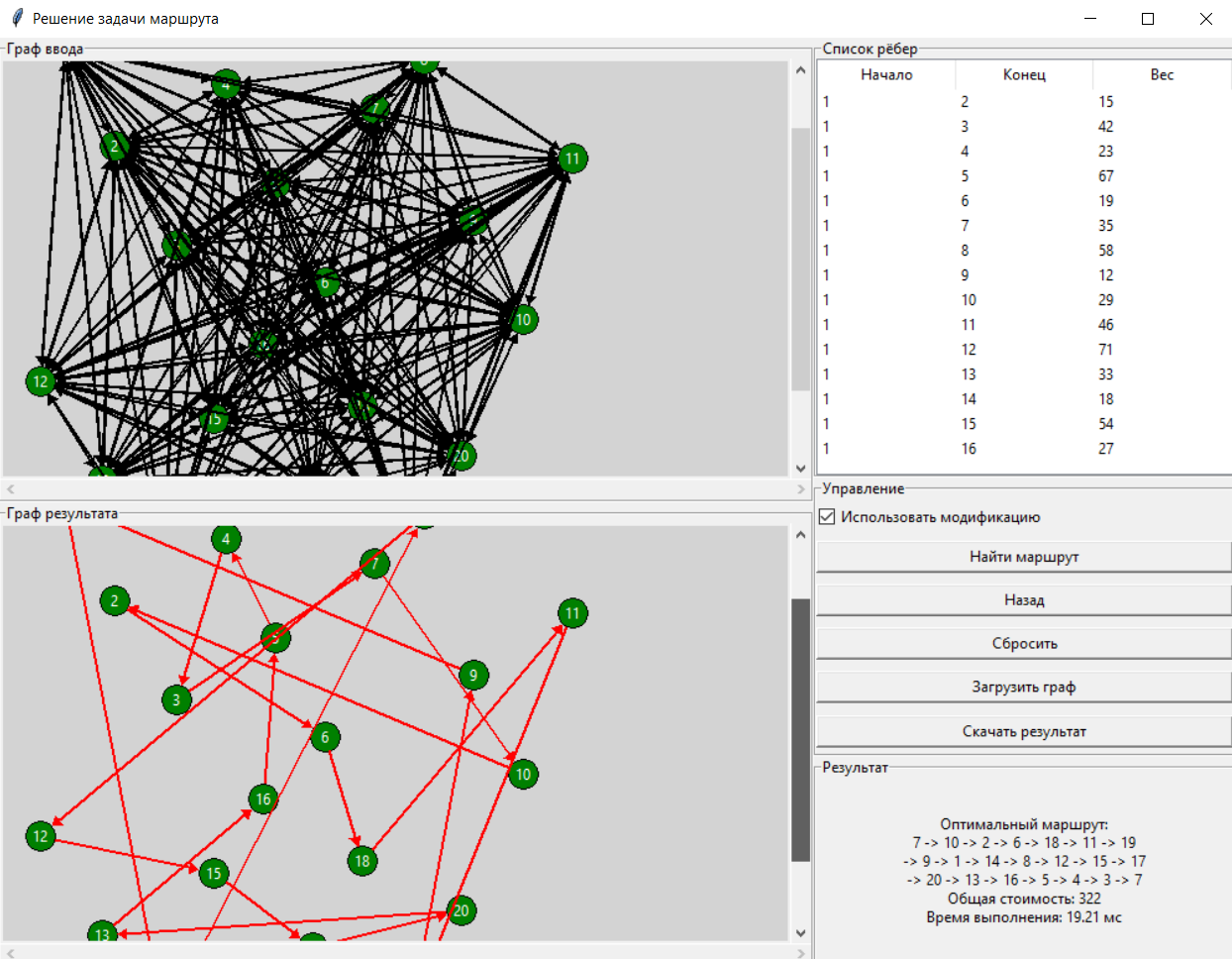
* **Инициализация частиц:** В фрейме “Граф ввода” необходимо при помощи нажатий на правую и левую кнопки мыши изобразить нужный граф. (Рис. 4)
* **Загрузка граф:** так же можно загрузить граф из текстового документа и изменять его. (Рис. 4)
*  Рис. 4.1, 4.2, 4.3 Ввод графа
* **Запуск алгоритма имитации отжига:** При нажатии на кнопки “Найти маршрут” произойдёт запуск алгоритма и мы получим результат в фрейме “Результаты и его изображение в фрейме “Граф результата”. (Рис. 5 / Рис. 6)

Рис. 5 Удачный результат

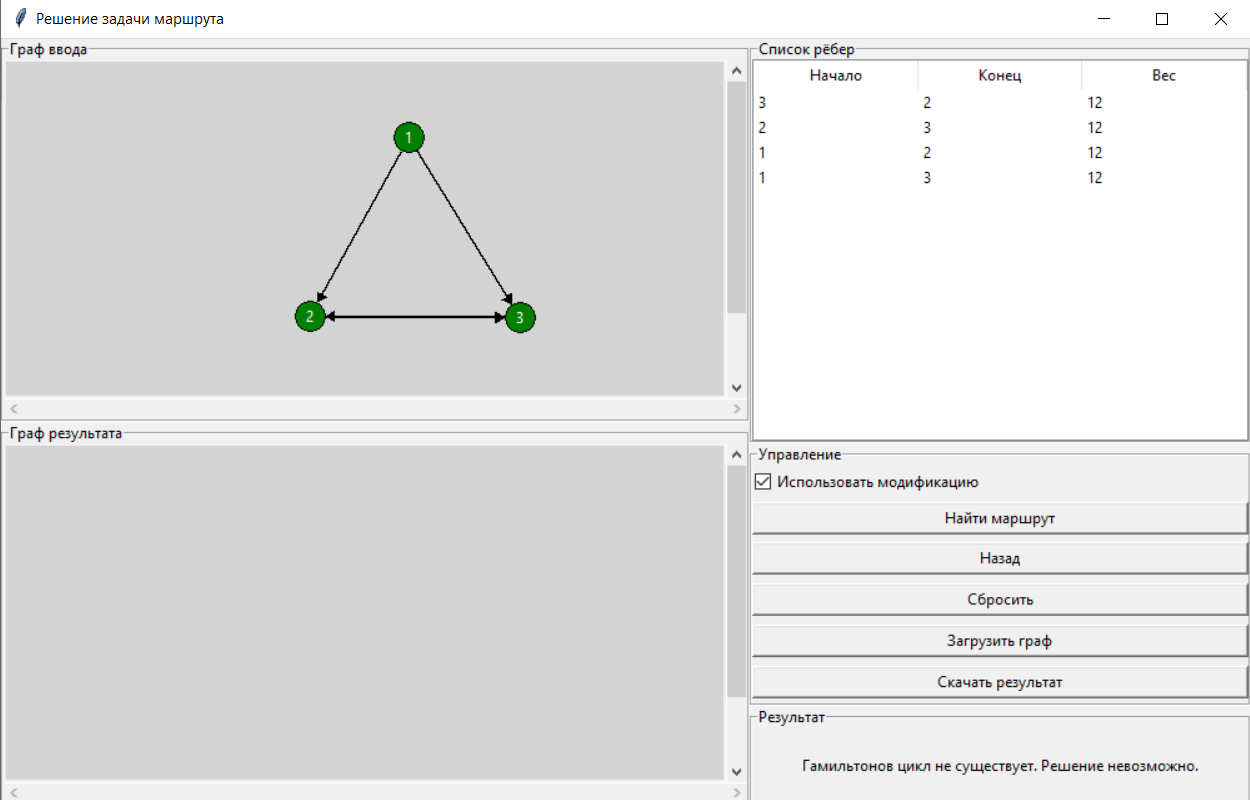


Рис. 6 Неудачный результат

# Анализ работы алгоритма с модификацией и без

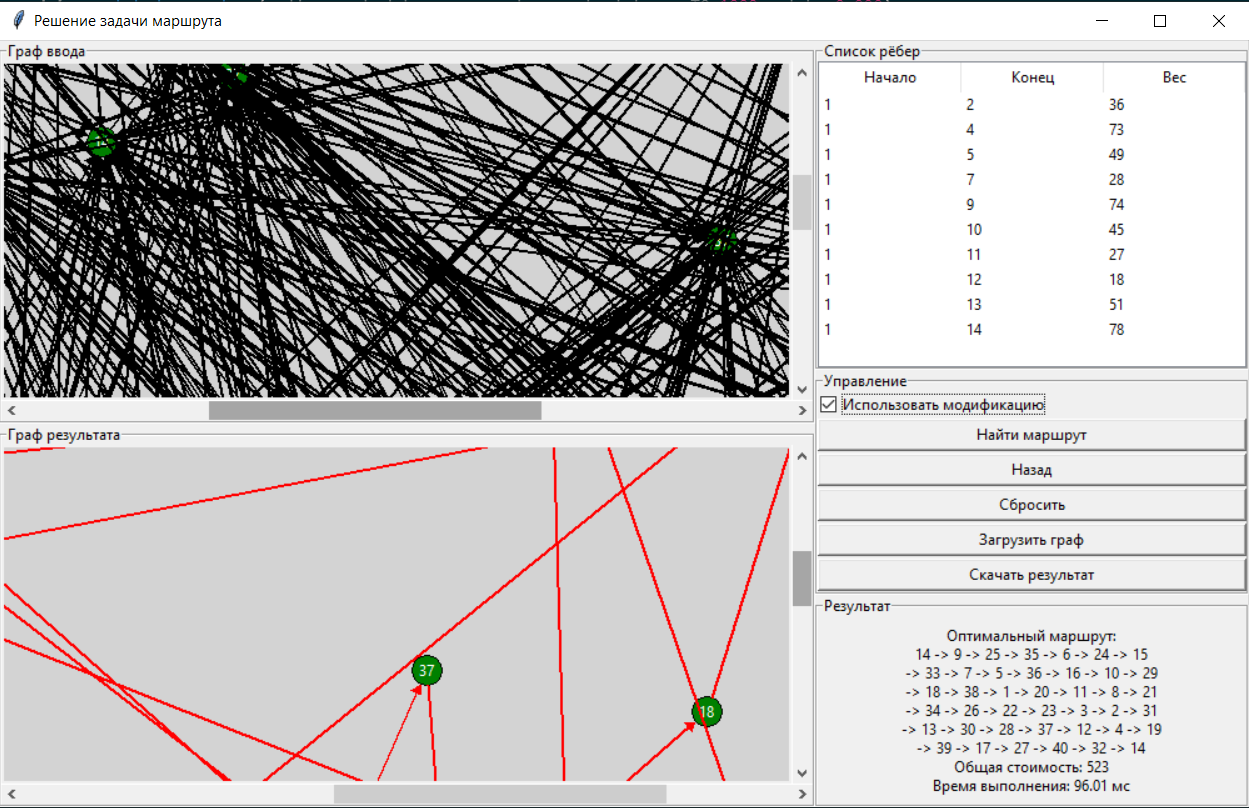
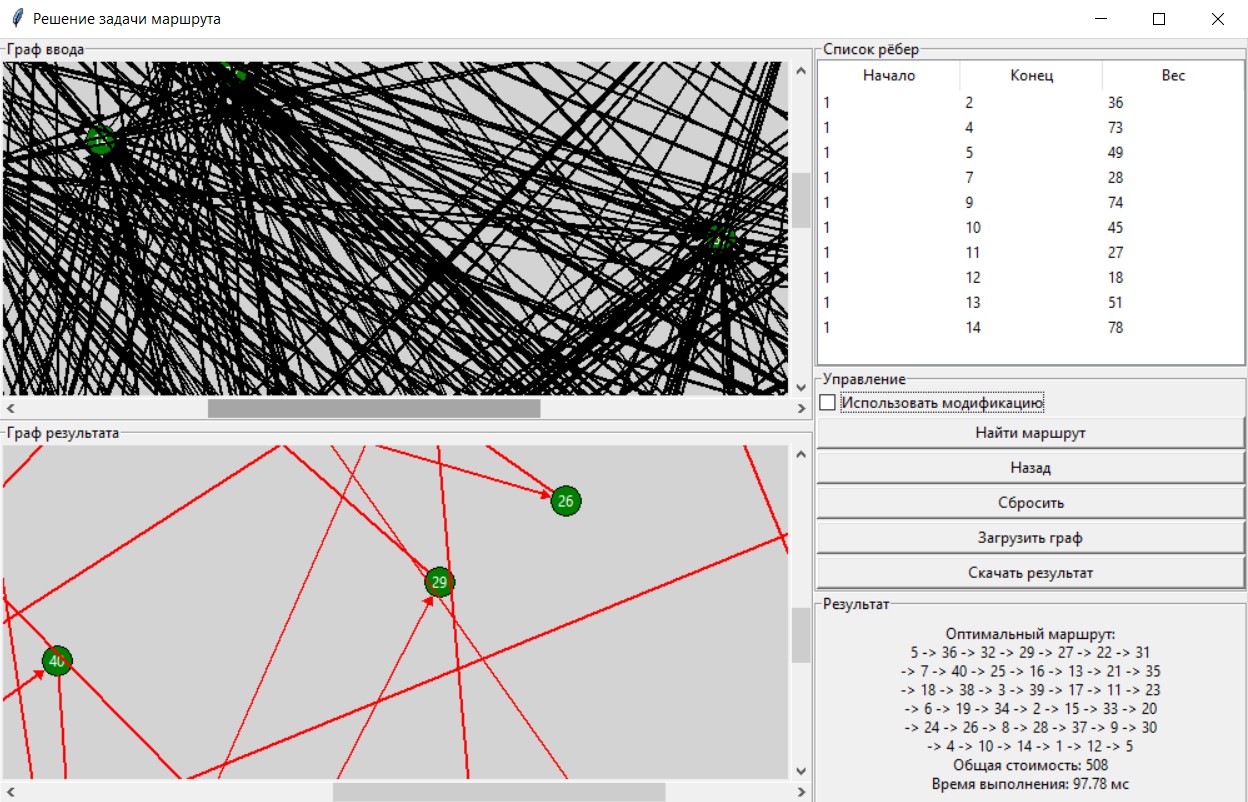
В качестве модификации использовался алгоритм **имитации отжига Коши.**

Рис. 7 Результат работы с модификацией

Рис. 8 Результат работы без модификации

Сравнение работы алгоритма с модификацией и без на графах с большим количеством вершин:

Таблица 2. Сравнение работы алгоритма с модификацией и без

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

Анализ таблицы показывает, что выбор режима охлаждения существенно влияет на качество решения задачи коммивояжера, но не на время выполнения. Распределение Коши эффективнее для графов среднего размера, обеспечивая сокращение длины пути на ~22.7% для 20 вершин. Геометрическое охлаждение лучше подходит для крупных графов, улучшая результат на ~12.7% для 40 вершин. Для малых графов оба подхода эквивалентны. Эти результаты подчёркивают важность адаптации параметров алгоритма к размеру задачи для достижения оптимального баланса между скоростью и качеством решения.

Сравнение работы алгоритма ближайшего соседа (БС) и имитации отжига (ИО):

Таблица 3. работы алгоритма ближайшего соседа

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

1. **Время выполнения:**

* ИО демонстрирует схожее время работы для версий с модификацией и без (0.97–96.06 мс), что указывает на незначительное влияние модификации на скорость.
* БС без модификации работает значительно быстрее (0.04–0.23 мс), но с модификацией время резко возрастает (до 8.03 мс для 40 вершин). Например, для 40 вершин БС с модификацией в 35 раз медленнее базовой версии.

1. **Качество решения (длина пути):**

* ИО с модификацией показывает улучшение результатов на больших графах: для 20 вершин путь сократился с 410 до 317, для 40 вершин — улучшение не зафиксировано (583 против 517 без модификации, возможна аномалия).
* БС с модификацией стабильно даёт более короткие пути: для 40 вершин путь уменьшился с 530 до 336. На малых графах (6 вершин) оба алгоритма равны.

1. **Масштабируемость:**

* ИО сохраняет стабильность времени при росте вершин (0.97 мс для 6 вершин → 96 мс для 40), но качество решения может ухудшаться без модификации.
* БС без модификации крайне быстр на малых данных (0.04 мс для 6 вершин), но его модификация становится ресурсозатратной на крупных графах (8.03 мс для 40 вершин).

**Итог:**

* ИО предпочтителен для средних и больших графов, где критично качество решения, а время выполнения допустимо, но уступает во всех аспектах БС.
* БС без модификации оптимален для малых графов, где скорость приоритетна. Модификация БС оправдана, если требуется сократить длину пути на крупных данных ценой увеличения времени.

# Вывод

В ходе лабораторной работы была успешно реализована программа для решения задачи коммивояжёра с использованием алгоритма имитации отжига (ИО) и его модификации — отжига Коши. Эксперименты показали, что модификация алгоритма ИО позволяет улучшить качество решения для графов среднего размера (например, сокращение длины пути на 22.7% для 20 вершин), однако время выполнения остаётся сопоставимым с базовой версией. Для крупных графов (40 вершин) геометрическое охлаждение оказалось эффективнее, что подчёркивает важность адаптации параметров алгоритма к масштабу задачи.

Сравнение с алгоритмом ближайшего соседа (БС) выявило ключевые компромиссы: БС без модификации демонстрирует высокую скорость на малых данных (0.04–0.23 мс), но его модификация значительно увеличивает время выполнения (до 8.03 мс для 40 вершин). При этом модифицированный БС обеспечивает более короткие маршруты на больших графах, что делает его использование предпочтительней чем алгоритм ИО.

Таким образом, выбор алгоритма зависит от требований задачи. Для средних и крупных графов, где критично качество решения, рекомендуется использовать имитацию отжига с адаптивными параметрами. Для малых данных, где скорость приоритетна, оптимален базовый алгоритм ближайшего соседа. Модификации обоих методов оправданы при необходимости баланса между точностью и ресурсозатратностью.

# Источники

* Редактор блок-схем.

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

*дата обращения: (08.03.2025)*

* 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: (08.03.2025)*

* 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: (08.03.2025)*

* openpyxl — Библиотека для записи и чтения файлов Excel.

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

*дата обращения: (16.03.2025)*