**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. 23Б15-пу**

**Бек В.А.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc194102718)
2. [Описание задачи (формализация задачи) 4](#_Toc194102719)
3. [Теоретическая часть 5](#_Toc194102720)

[**Задача коммивояжера:** 5](#_Toc194102721)

[**Алгоритм имитации отжига:** 5](#_Toc194102722)

[**Модификация (Отжиг Коши):** 6](#_Toc194102723)

1. [Основные шаги программы 7](#_Toc194102724)
2. [Блок схема программы 9](#_Toc194102725)
3. [Описание программы 11](#_Toc194102726)
4. [Рекомендации пользователя 13](#_Toc194102727)
5. [Рекомендации программиста 15](#_Toc194102728)
6. [Контрольный пример 16](#_Toc194102729)
7. [Анализ работы алгоритма с модификацией и без 18](#_Toc194102730)
8. [Вывод 20](#_Toc194102731)
9. [Источники 21](#_Toc194102732)

# Цель работы

Цель данной работы — изучение и реализация решения задачи о коммивояжере с использованием метода имитации отжига. Задача заключается в нахождении кратчайшего гамильтонова цикла во взвешенном орграфе. Разработанная программа будет протестирована на заданном взвешенном орграфе.

# 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/(1+ iterations)

# Теоретическая часть

### **Задача коммивояжера:**

Задача коммивояжёра (**Traveling Salesman Problem** / **TSP**) является одной из классических задач комбинаторной оптимизации. Она формулируется следующим образом: имеется множество городов и известны расстояния между каждой парой городов. Требуется найти кратчайший маршрут, который проходит через каждый город ровно один раз и возвращается в начальную точку.

Задача коммивояжёра относится к классу NP-трудных задач, и при увеличении числа городов найти точное решение становится практически невозможно за разумное время. Поэтому на практике используют специальные приближённые методы и эвристики, которые позволяют быстро находить хорошие, но не обязательно идеальные маршруты.

### **Алгоритм имитации отжига:**

Имитация отжига вдохновлена процессом физического отжига, где металл или другой материал нагревается, а затем медленно охлаждается, чтобы достичь наиболее стабильного состояния. В контексте задачи коммивояжёра, алгоритм использует случайные изменения текущего маршрута, оценивает их и решает, стоит ли принять новое состояние в зависимости от температуры системы, которая постепенно уменьшается.

* **Инициализация**: Генерация случайного плохого маршрута при помощи dfs и вычисление его длины.
* **Основной цикл**:
  + Случайным образом меняются два города в маршруте.
  + Если новый маршрут короче, он принимается безусловно.
  + Если новый маршрут длиннее, его принимают с переходной вероятностью.
* **Охлаждение**: Температура уменьшается, что уменьшает вероятность принятия худших решений.
* **Завершение**: Алгоритм заканчивается, когда температура достигает минимального значения.

### **Модификация (Отжиг Коши):**

Модификация алгоритма отличается лишь тем, что для переходной вероятности и изменения температуры используются другие формулы:

Где P – переходная вероятность, δ– разница между новым и старым решением, T – температура.

Где T – обновленная температура, T0 – начальная температура, k – число прошедших итераций.

# 

# Основные шаги программы

1. **Создание графического интерфейса**
   * Используется библиотека **Tkinter**.
   * Основное окно разделено на два блока: левый (для графического ввода/вывода) и правый (для отображения рёбер и управления).
   * В левом блоке два холста: для входного графа и для вывода решения.
   * В правом блоке — таблица рёбер, кнопки управления и область вывода результата.
2. **Добавление вершин**
   * Левый клик по холсту добавляет вершину, если рядом нет других.
3. **Добавление рёбер**
   * Правый клик по вершине выбирает её как начальную.
   * Следующий правый клик на другой вершине создаёт между ними ребро с заданным пользователем весом.
   * Если ребро уже существует, можно изменить его вес повторно кликая на начальную и конечную вершины
4. **Решение задачи коммивояжера**
   * На кнопку “Рассчитать” запускается процесс решения задачи коммивояжера
   * Создаётся матрица смежности на основе рёбер.
   * Для каждого возможного начального города ищется приближённый путь алгоритмом имитации отжига.
5. **Отображение результата**
   * Если путь найден, он выводится в текстовом виде и рисуется на выходном холсте.
   * Если решения нет, отображается соответствующее сообщение.
6. **Дополнительные кнопки**

* **Кнопка "Отменить"** отменяет последнее действие, удаляя последнюю добавленную вершину или ребро.
* **Кнопка "Очистить"** очищает все данные и визуальные элементы, сбрасывая все вершины, рёбра и результаты.
* **Кнопка "Модификация"** позволяет выбрать алгоритм будет стартовать только из первой вершины или попробует из всех.
* **Кнопка "Сохранить граф"** открывает проводник и позволяет сохранить уже введённый граф в xlsx файл.
* **Кнопка "Загрузить граф"** открывает проводник и позволяет выбрать xlsx файл в котором уже записан граф.

# Блок схема программы

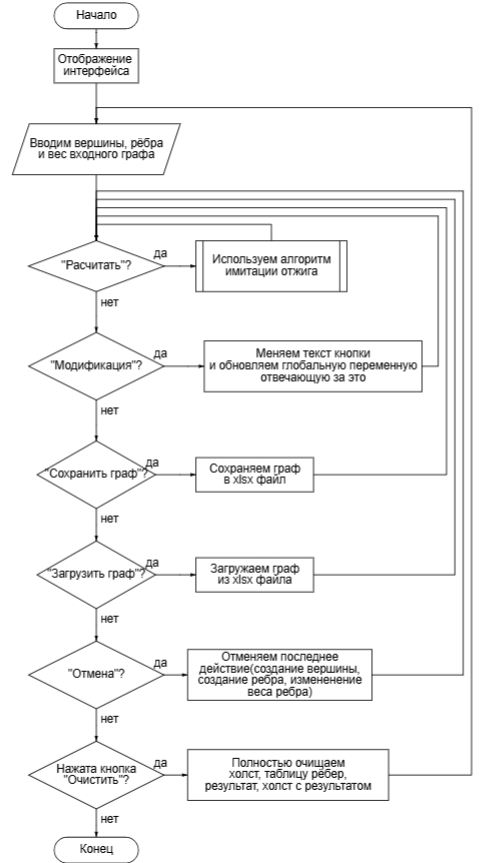


Рис. 1 Блок-схема основной программы

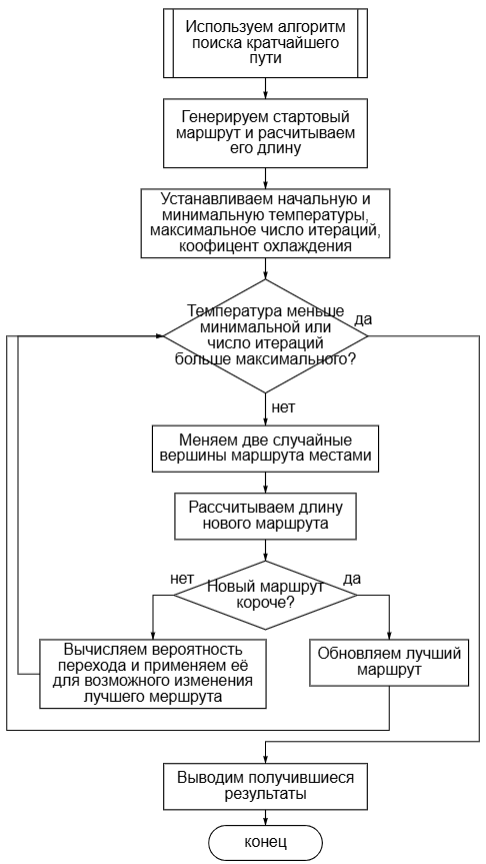


Рис. 2 Блок-схема подпрограммы

# Описание программы

Программная реализация задачи коммивояжёра написана на языке **Python 3.12.7** с использованием библиотек **tkinter, ttk, math, openpyxl**. Программа представляет собой графический интерфейс, который позволяет создавать и редактировать граф, состоящий из вершин и рёбер. На основе введённых данных программа решает задачу коммивояжёра, находя кратчайший путь, который проходит через все вершины. В графическом интерфейсе пользователи могут добавлять вершины, соединять их рёбрами и указывать веса рёбер. Также предусмотрены возможности для отмены последнего действия и очистки всех данных, сохранения и загрузки графа. Боле того предусмотрено отображает решение задачи коммивояжёра, выводя оптимальный маршрут и его длину.

Таблица 1. simulated\_annealing\_algorithm.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| toggle\_mode() | Меняет модификацию(Вкл/Выкл) | None |
| add\_vertex(event) | Добавляет вершину на канвас в месте клика мышью. | None |
| select\_vertex\_for\_edge(event) | Выбирает вершину для соединения с другой или изменяет вес существующего рёбра. | None |
| draw\_directed\_edge(start\_vertex, end\_vertex) | |  | | --- | | Рисует направленное ребро между двумя вершинами. | | ID нарисованного ребра |
| draw\_shortest\_path(path, adj\_matrix) | |  | | --- | | Рисует кратчайший путь, найденный для задачи коммивояжёра. |  |  | | --- | |  | | None |
| calculate\_tsp() | |  | | --- | | Рассчитывает решение задачи коммивояжёра, используя алгоритм имитации отжига. | | None |
| simulated\_annealing(adj\_matrix) | Применяет алгоритм имитации отжига или модификацию – Отжиг Коши | Кратчайший путь, его размер, кол-во итераций |
| undo\_last\_action() | Отменяет последнее действие пользователя (удаление вершины или рёбра). | None |
| clear\_all() | |  | | --- | | Очищает все данные (все вершины и рёбра), сбрасывает интерфейс. | | None |

# 

# Рекомендации пользователя

Для корректного выполнения программы следуйте приведённым шагам:

* **Добавление вершин**:

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

* **Сохранение графа**:

Для сохранения введённого графа используйте кнопку "Сохранить граф". Откроется проводник, в котором вы сможете ввести название сохраняемого файла и его расположение.

* **Загрузка графа**:

Для загрузки введённого ранее графа используйте кнопку "Загрузить граф". Откроется проводник, в котором вы сможете выбрать файл, в который ранее сохраняли граф.

* **Отмена последнего действия**:

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

* **Очистка графа**:

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

* **Модификация**:

Нажатие на кнопку: “Модификация: Вкл.” позволит поменять режим работы алгоритма. Модификация Вкл. – алгоритм работает дольше, но точнее, Модификация Выкл. – алгоритм работает гораздо быстрее, но достаточно неточно.

* **Расчёт оптимального пути**:

После добавления всех вершин и рёбер, нажмите кнопку "Рассчитать". Программа найдет оптимальный путь для задачи коммивояжера и отобразит результат на поле "Выходной граф". На поле также будет показан весь маршрута, а результаты расчета будут отображены в блоке "Результаты". Здесь будет показан оптимальный путь, общий вес маршрута и время выполнения алгоритма.

# 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 **3.1** 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/Kliooo/Algorithms-and-data-structures>**

# Контрольный пример

* **Запуск программы:** Для запуска программы используйте файл simulated\_annealing\_algorithm.py. Программа откроет графический интерфейс (Рис. 4).

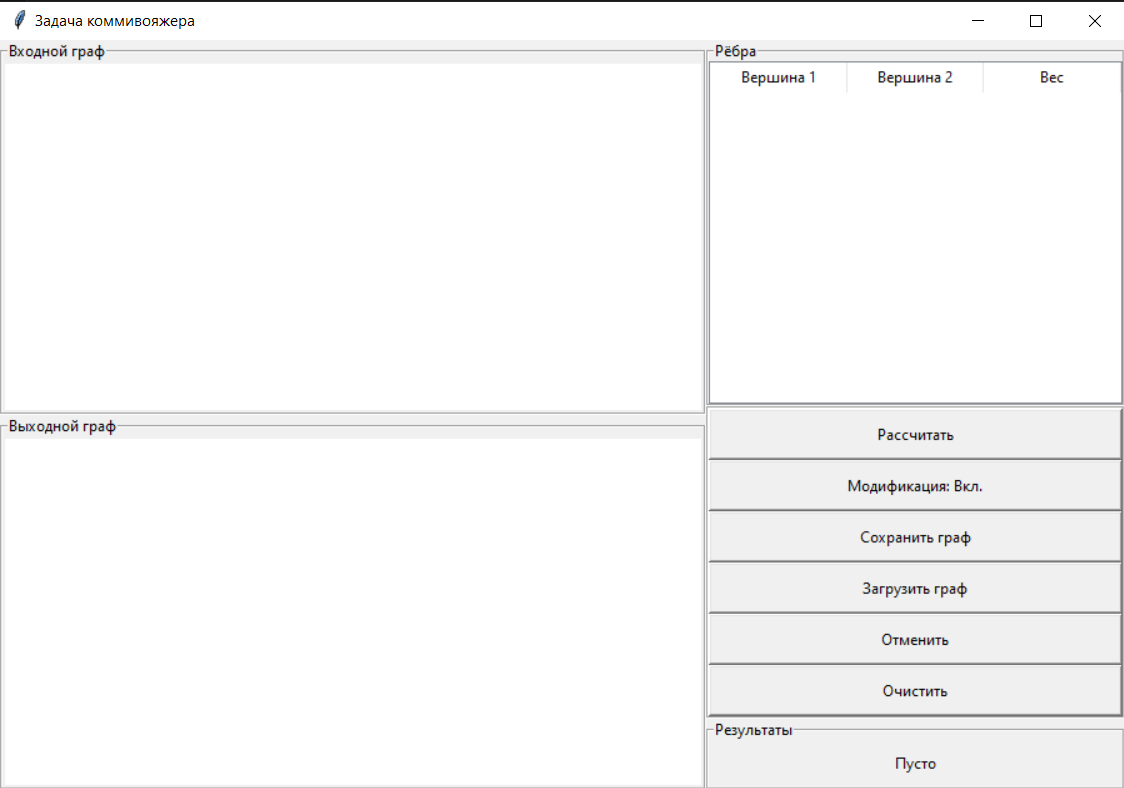


Рис. 3 Графический интерфейс программы

* **Инициализация частиц:** В фрейме “Входной граф” необходимо при помощи нажатий на правую и левую кнопки мыши изобразить нужный граф. (Рис. 4)

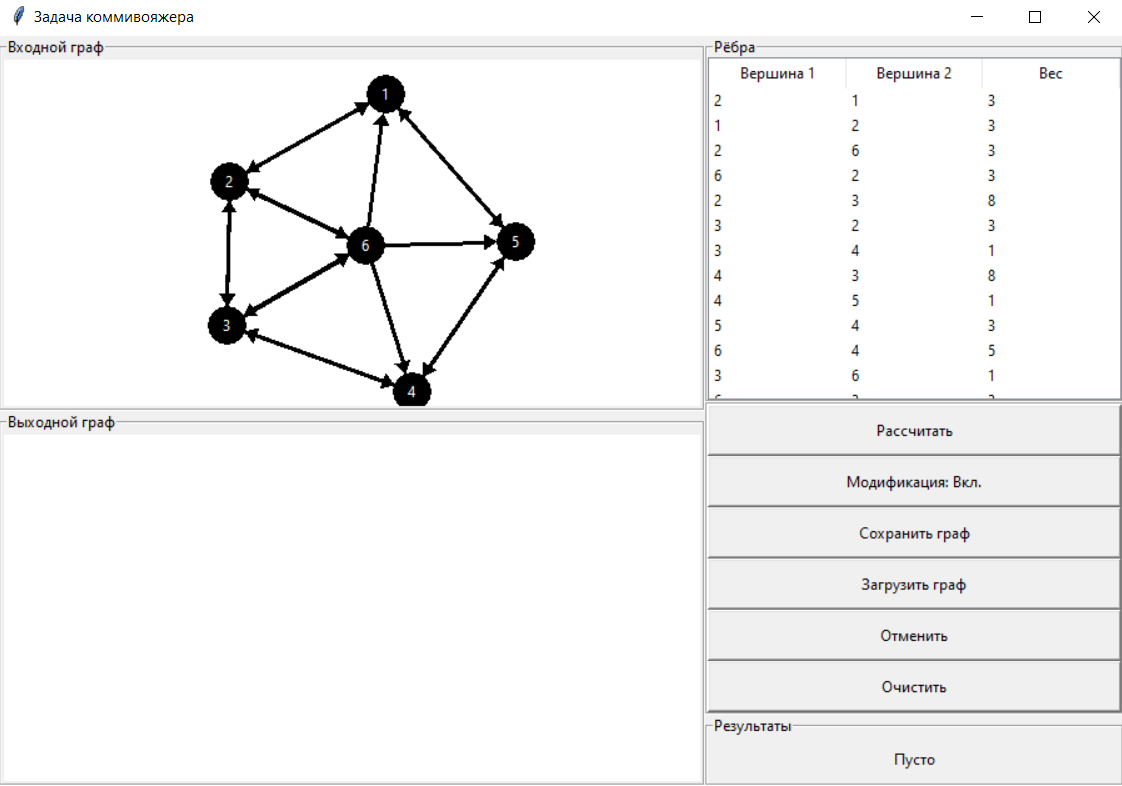


Рис. 4 Ввод графа

* **Запуск алгоритма ближайшего соседа с модификацией:** При нажатии на кнопку “Рассчитать” произойдёт запуск алгоритма, и мы получим результат в фрейме “Результаты” и его изображение в фрейме “Выходной граф”. (Рис. 5 / Рис. 6)

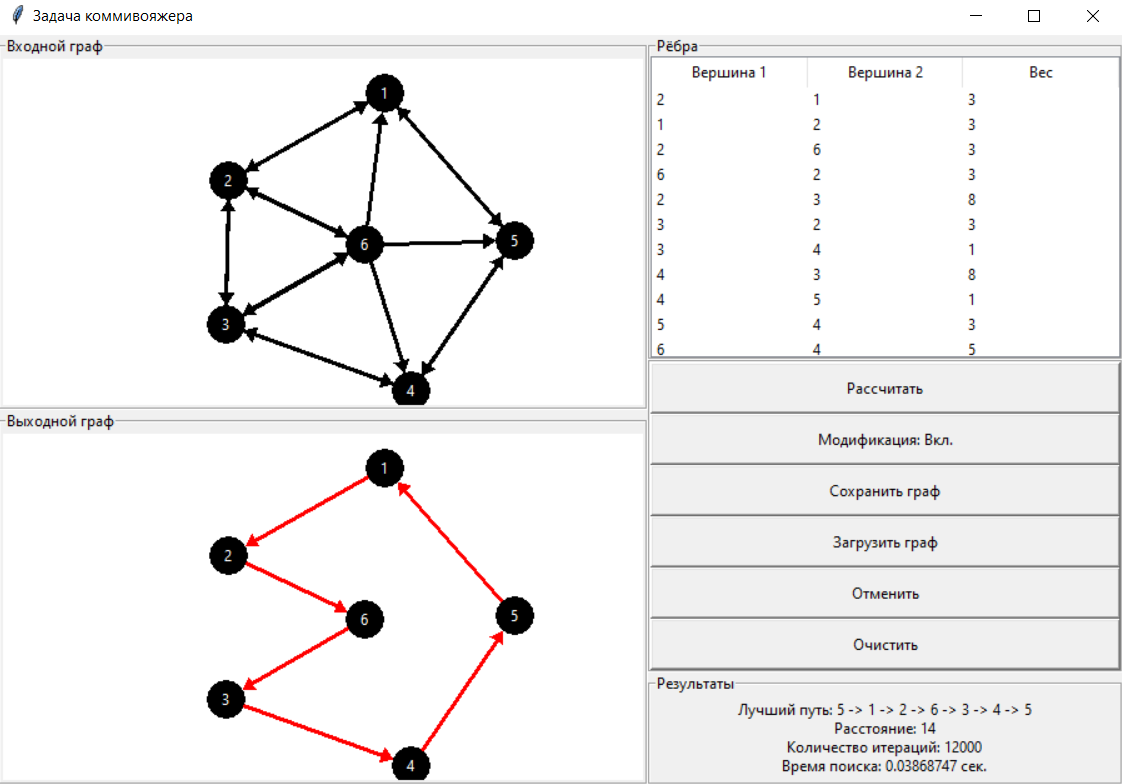


Рис. 5 Результат с модификацией

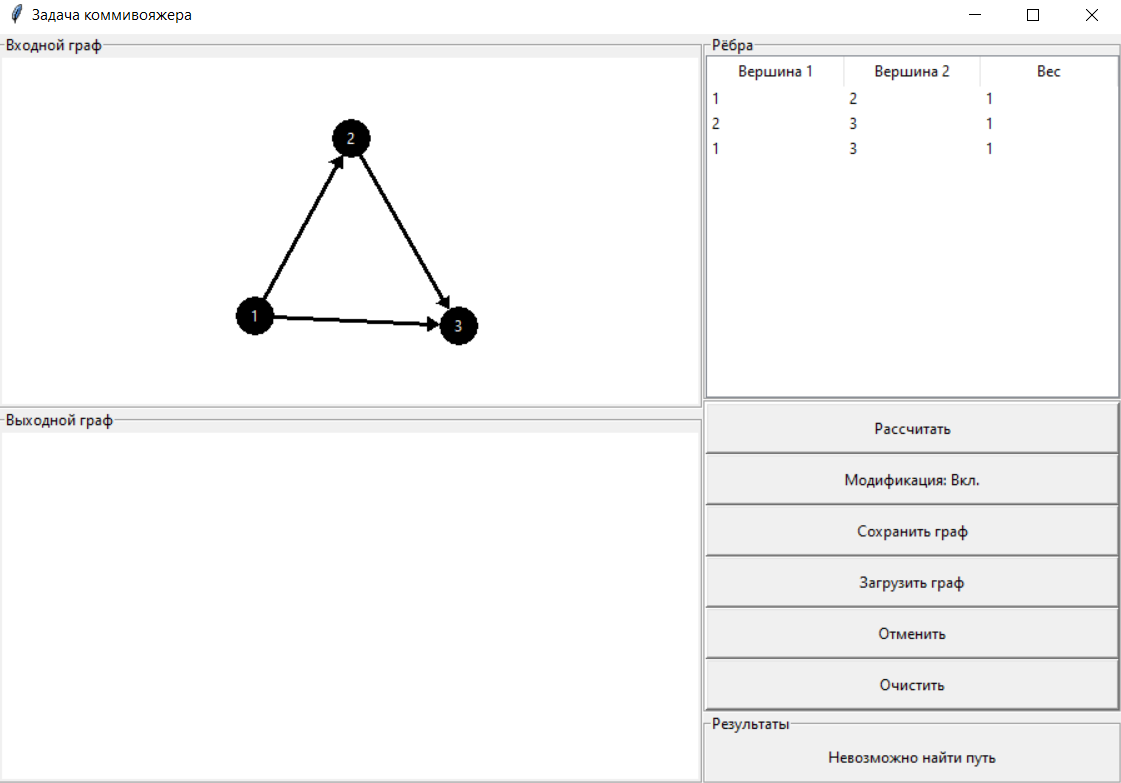


Рис. 6 Неудачный результат

# Анализ работы алгоритма с модификацией и без

В качестве модификации использовался **отжиг Коши**, который включает в себя новые способы вычисления температуры и вероятности перехода

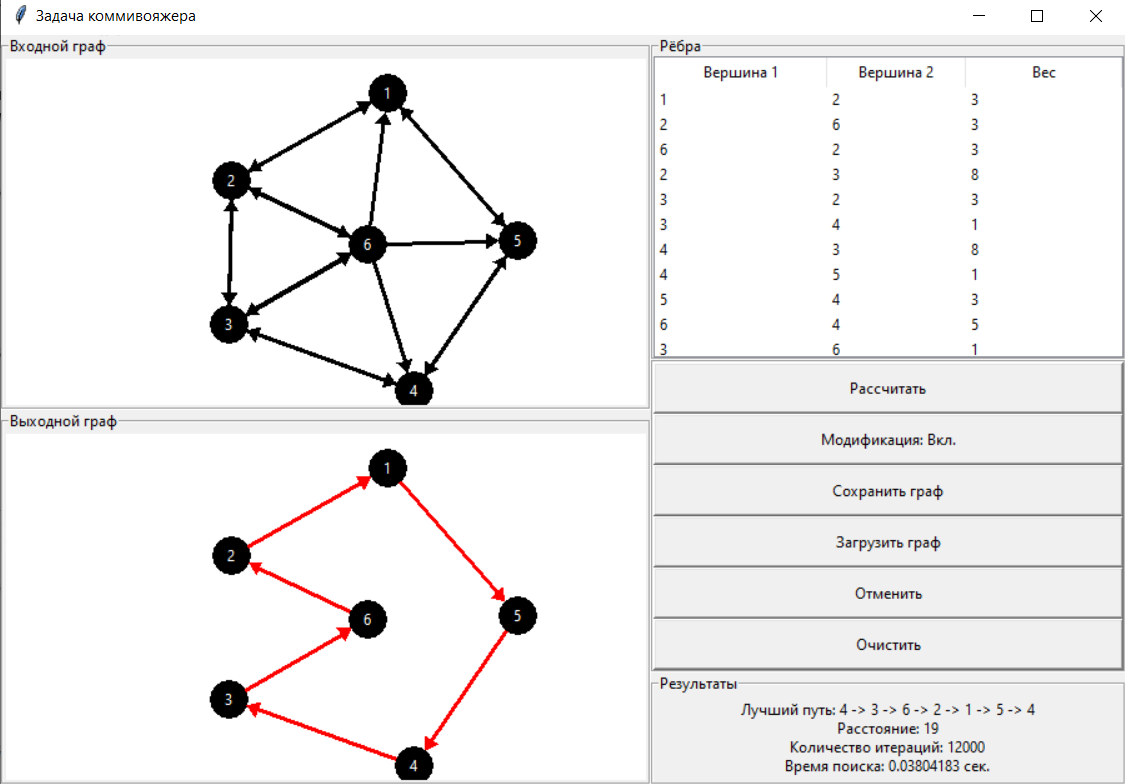


Рис. 7 Результат работы с модификацией

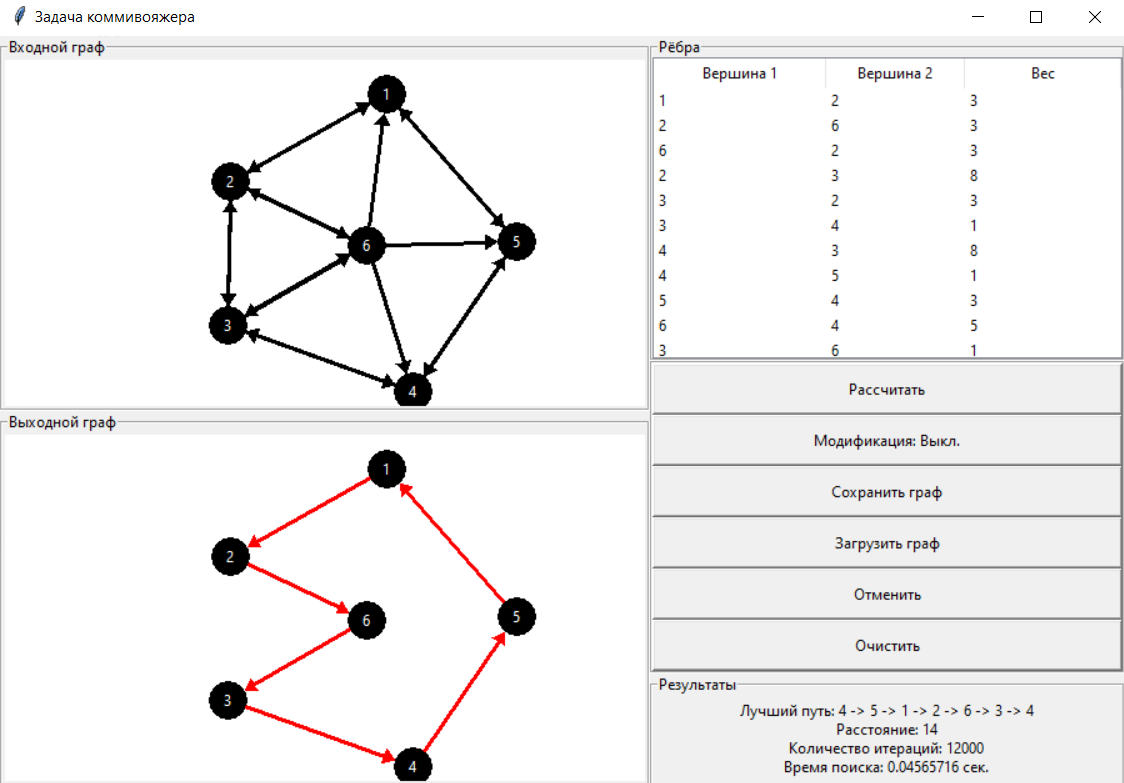


Рис. 8 Результат работы без модификации

Важно сказать: несмотря на то, что алгоритм с модификацией и без удачно справляются с предложенным для проверки графом, они иногда из-за вероятности перехода всё-таки попадают в неверный результат 19.

Таблица 2. Сравнение работы алгоритма с модификацией и без для ближайшего соседа

|  |  |  |  |
| --- | --- | --- | --- |
| Модификация | Время | Длина пути | Кол-во вершин |
| С модификацией | 0.00171… | 17 | 15 |
| Без модификации | 0.00000… | 20 | 15 |
| С модификацией | 0.00204… | 32 | 30 |
| Без модификации | 0.00000… | 35 | 30 |

Таблица 3. Сравнение работы алгоритма с модификацией и без для имитации отжига

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модификация | Время | Длина пути | Кол-во вершин | Кол-во итераций |
| С модификацией | 0.080… | 40-400 | 15 | 30000 |
| Без модификации | 0.060… | 30-400 | 15 | 18412 |
| С модификацией | 0.195… | +-1000 | 30 | 60000 |
| Без модификации | 0.080… | +-1000 | 30 | 18412 |

Таким образом можно сделать самый очевидный вывод – алгоритм имитации отжига по всем параметрам проигрывает алгоритму ближайшего соседа. Осмелюсь предположить, что мог создать графы не лучшим образом (недостаточно полные и с слишком большим разбросом от лучшего решения).

Также важно отметить для алгоритма имитации отжига: вариант без модификации хоть и получает такие же ответы, а иногда и лучше работает быстрее за счет меньшего количества итераций.

Вероятно алгоритм имитации отжига может хорошо показать себя на более полных или лучше равномерно взвешенных графах.

# Вывод

В ходе работы был реализован и протестирован алгоритм имитации отжига для решения задачи коммивояжёра, а также его модифицированный вариант. Результаты экспериментов показали, что данный метод, несмотря на его теоретическую способность находить приближённые решения, проигрывает жадному алгоритму ближайшего соседа во всех смыслах. Это может быть связано как со сложностью самого метода, требующего значительного количества итераций, так и с его склонностью к поиску решений, которые не всегда являются оптимальными.

Анализ работы алгоритма имитации отжига продемонстрировал, что модифицированный вариант не всегда даёт преимущество. В частности, в графах из 15 вершин версия без модификации чаще находила решение, тогда как модифицированный вариант справлялся с задачей реже. Кроме того, для графов из 30 вершин оба варианта алгоритма не смогли найти корректное решение.

Таким образом, алгоритм имитации отжига и его модифицированный вариант (Отжиг Коши) в данной ситуации оказались менее эффективны, чем алгоритм ближайшего соседа.

# Источники

* Редактор блок-схем.

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

*дата обращения: (29.03.2025)*

* 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: (29.03.2025)*

* 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: (29.03.2025)*

* openpyxl — Библиотека для записи и чтения файлов Excel.

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

*дата обращения: (29.03.2025)*