**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Решение задачи о коммивояжере с помощью алгоритма имитации отжига».**

**Студент гр. 23Б15-пу**

**Беляева А.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

**Оглавление**

Цель работы....................................................................................................................3

Краткое описание алгоритма имитации отжига..............................................................................................................................3

Схема пошагового выполнения алгоритма........................................................................................................................5

Формализация задачи....................................................................................................6

Описание программы....................................................................................................6

Рекомендации пользователя.........................................................................................8

Рекомендации для программиста.................................................................................9

Исходный код программы......................................................................................…...9

Спецификация программы............................................................................................9

Контрольный пример....................................................................................................10

Анализ работы алгоритма.......................................................................................................................12

Вывод..............................................................................................................................13

Источники......................................................................................................................13

**Цель работы**

Исследование особенностей решения задачи о коммивояжёре с помощью алгоритма имитации отжига

**Краткое описание алгоритма имитации отжига.**

Алгоритм имитации отжига (англ. simulated annealing)[3] — эвристический алгоритм глобальной оптимизации, особенно эффективный при решении дискретных и комбинаторных задач.

Алгоритм вдохновлён процессом [отжига](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%B6%D0%B8%D0%B3) в металлургии — техники, заключающейся в нагревании и контролируемом охлаждении металла, чтобы увеличить его кристаллизованность и уменьшить дефекты. Симулированние отжига в переборных задачах может быть использовано для приближённого нахождения глобального минимума функций с большим количеством свободных переменных.

Алгоритм вероятностный и не даёт почти никаких гарантий сходимости, однако хорошо работает на практике при решении NP-полных задач.

Для примера будем рассматривать задачу коммивояжёра:

*Eсть n городов, соединённых между собой дорогами. Необходимо проложить между ними кратчайший замкнутый маршрут, проходящий через каждый город только один раз.*

Пусть имеется некоторая функция f(x) от состояния x, которую мы хотим минимизировать. В данном случае x это перестановка вершин (городов) в том порядке, в котором мы будем их посещать, а f(x) это длина соответствующего пути.

Возьмём в качестве базового решения какое-то состояние x0 (например, случайную перестановку) и будем пытаться его улучшать.[1]

На входе: минимальная температура IMG_256, начальная температура IMG_257

1. Задаём произвольное первое состояние IMG_258 IMG_259
2. Пока IMG_260
3. IMG_261
4. IMG_262
5. Если IMG_263, тогда IMG_264
6. Если IMG_265 переход осуществляется с вероятностью IMG_266
7. Понижаем температуру IMG_267
8. Возвращаем последнее состояние s

**Схема пошагового выполнения алгоритма**

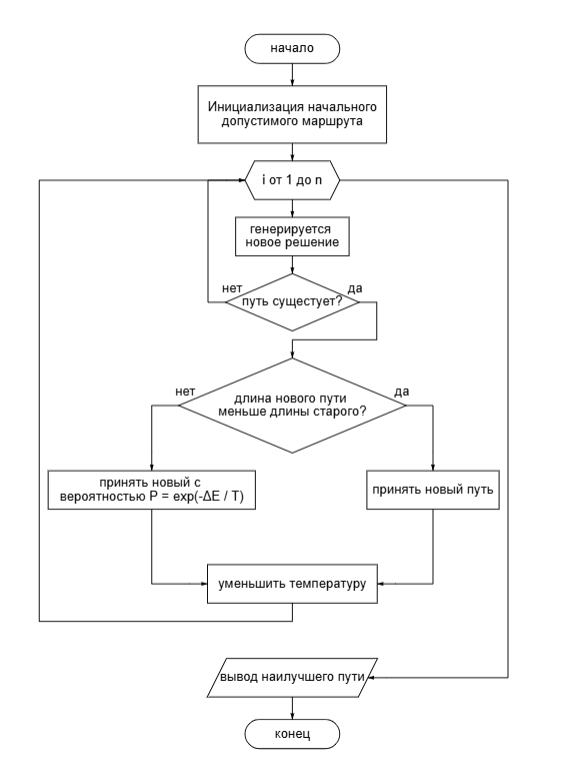


Рис 1. Блок-схема программы

**Формализация задачи**

Задача коммивояжёра заключается в нахождении кратчайшего гамильтонова цикла в графе. Граф задаётся как набор вершин и рёбер, где каждое ребро имеет вес (расстояние между вершинами)[4].

**Описание программы**

Программа реализована на языке Python с использованием библиотек:

1. Networkx[2] — для работы с графами.
2. Tkinter[1] — для создания графического интерфейса.

**Основные компоненты программы:**

Таблица 1. GraphApp.

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Тип возвращаемых данных** |
| \_\_init\_\_ | Создание интерфейса | None |
| handle\_click | Обработка клика | None |
| add\_node | Добавление узла | None |
| add\_edge | Добавление ребра | None |
| update\_table | Обновление таблицы | None |
| draw\_graph | Отрисовка графа | None |
| cencelator | Отмена последнего действия при построении графа | None |
| remove\_from\_table | Удаление ребра из таблицы | None |
| main\_procces | Запуск алгоритма | None |
| nearest\_neighbor | Алгоритм ближайшего соседа | None |
| run\_rnn | Модификация rnn | \_\_len\_\_, \_\_getitem\_\_ |
| calculate\_path\_distance | Вычисление дистанции | float|int |
| draw\_result | Отрисовка итогового графа | None |
| clear\_graph | Очистка графа | None |
| simulated\_annealing | Имитация отжига | list |
| run\_annealing | Запуск имитации отжига с привязкой к интерфейсу | None |
| boltzmann\_annealing | Модификация Больцмановкий отжиг | list |
| run\_boltzmann\_annealing | Запуск имитации модификации отжига с привязкой к интерфейсу | None |

# **Рекомендации для пользователя**

# Запустите программу, выполнив команду:

# python graph\_analysis.py

# 2. Интерфейс программы

# После запуска программы откроется окно с двумя областями:

# Левая область: для построения графа.

# Правая область: для отображения результата (оптимального маршрута).

# Внизу окна находится таблица, отображающая все рёбра графа и их веса.

# 3. Построение графа

# Добавление вершин:

# Щёлкните левой кнопкой мыши в левой области, чтобы добавить вершину. Вершины будут пронумерованы автоматически.

# Добавление рёбер:

# Щёлкните на первую вершину, затем на вторую. Появится окно для ввода веса ребра. Введите вес и нажмите "ОК".

# Удаление рёбер:

# Нажмите кнопку "Отменить", чтобы удалить последнее добавленное ребро.

# 4. Запуск алгоритма

# После построения графа нажмите кнопку "Найти оптимальный путь". Программа найдёт кратчайший маршрут и отобразит его в правой области.

# Если вы хотите улучшить результат, поставьте галочку "Использовать 2-opt" перед запуском алгоритма.

# 5. Очистка графа

# Чтобы начать заново, нажмите кнопку "Очистить граф". Это удалит все вершины и рёбра.

# 6. Сохранение результата

# Результат (оптимальный маршрут и его длина) отображается в нижней части окна. Вы можете вручную записать его или сделать скриншот.

# **Рекомендации для программиста**

# Запуск программы Убедитесь, что на вашем компьютере установлен Python (версии 3.7 или выше).

# Установите необходимые библиотеки: networkx, tkinter

# Структура программы

# Программа состоит из следующих основных компонентов:

# Графический интерфейс: реализован с помощью библиотеки tkinter.

# Логика работы с графом: используется библиотека networkx.

**Исходный код программы**

**[Git](https://github.com/hysterria/nearest_neighbor.git)**

**Спецификация программы**

*1. Входные данные:*

Программа принимает на вход ориентированный взвешенный граф, который пользователь вводит вручную через графический интерфейс. Граф задаётся в виде множества вершин и рёбер:

* Вершины вводятся кликом на поле графа.
* Рёбра задаются путем соединения вершин, при этом пользователь указывает вес каждого ребра.

*2. Выходные данные:*

Программа вычисляет и отображает кратчайший гамильтонов цикл, найденный методом ближайшего соседа. Выходные данные включают:

* Графическое представление маршрута на холсте. Оптимальный путь выделяется зелёными линиями, а веса рёбер подписываются красным цветом.
* Числовое значение длины пути, выводится в интерфейсе в виде строки «Общая длина: X».
* Сообщения об ошибках, если путь невозможно построить (например, если граф несвязный или невозможно замкнуть цикл).

**Контрольный пример**

1. Запуск программы:

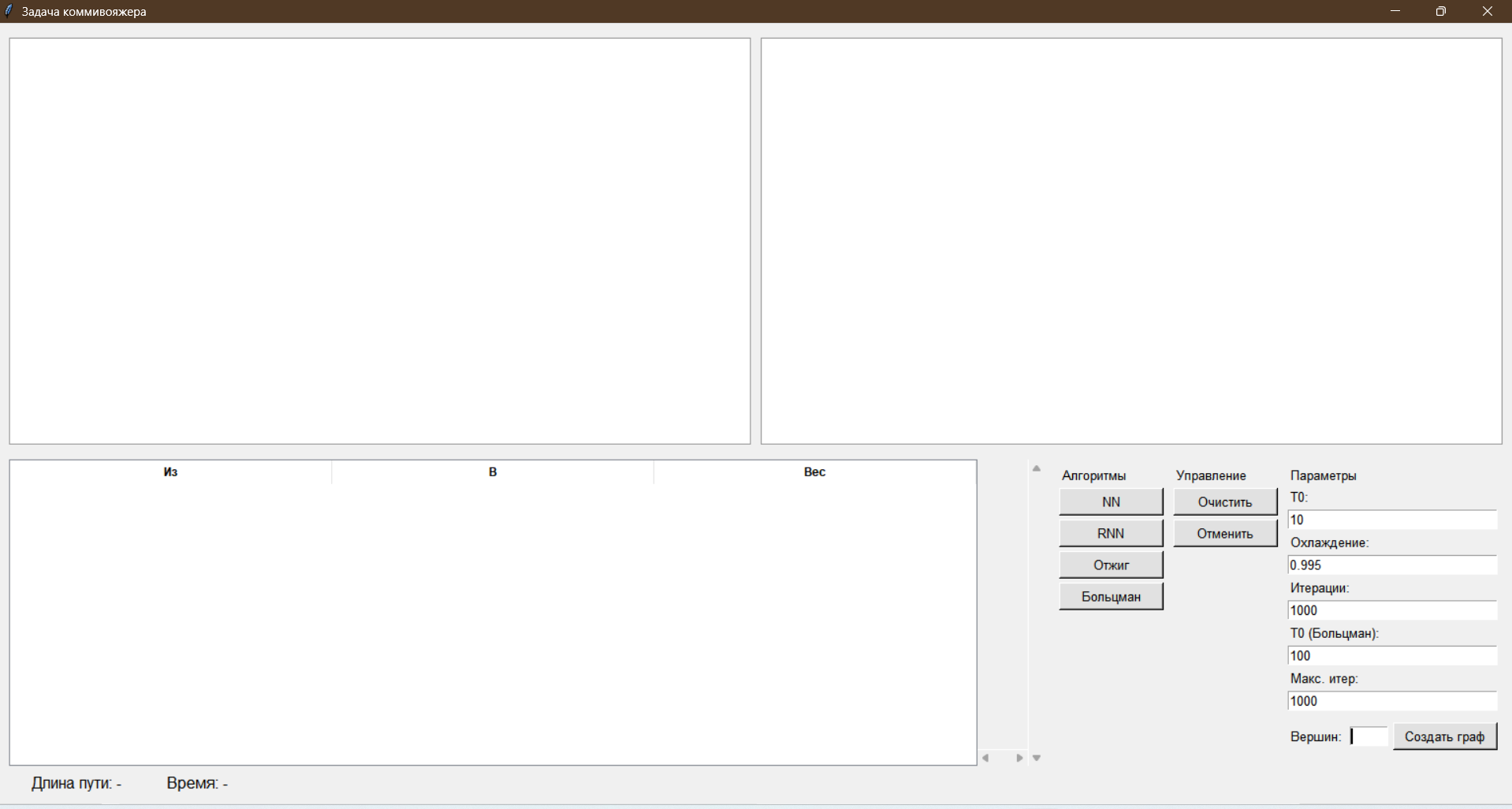


Рис. 2. Страт программы.

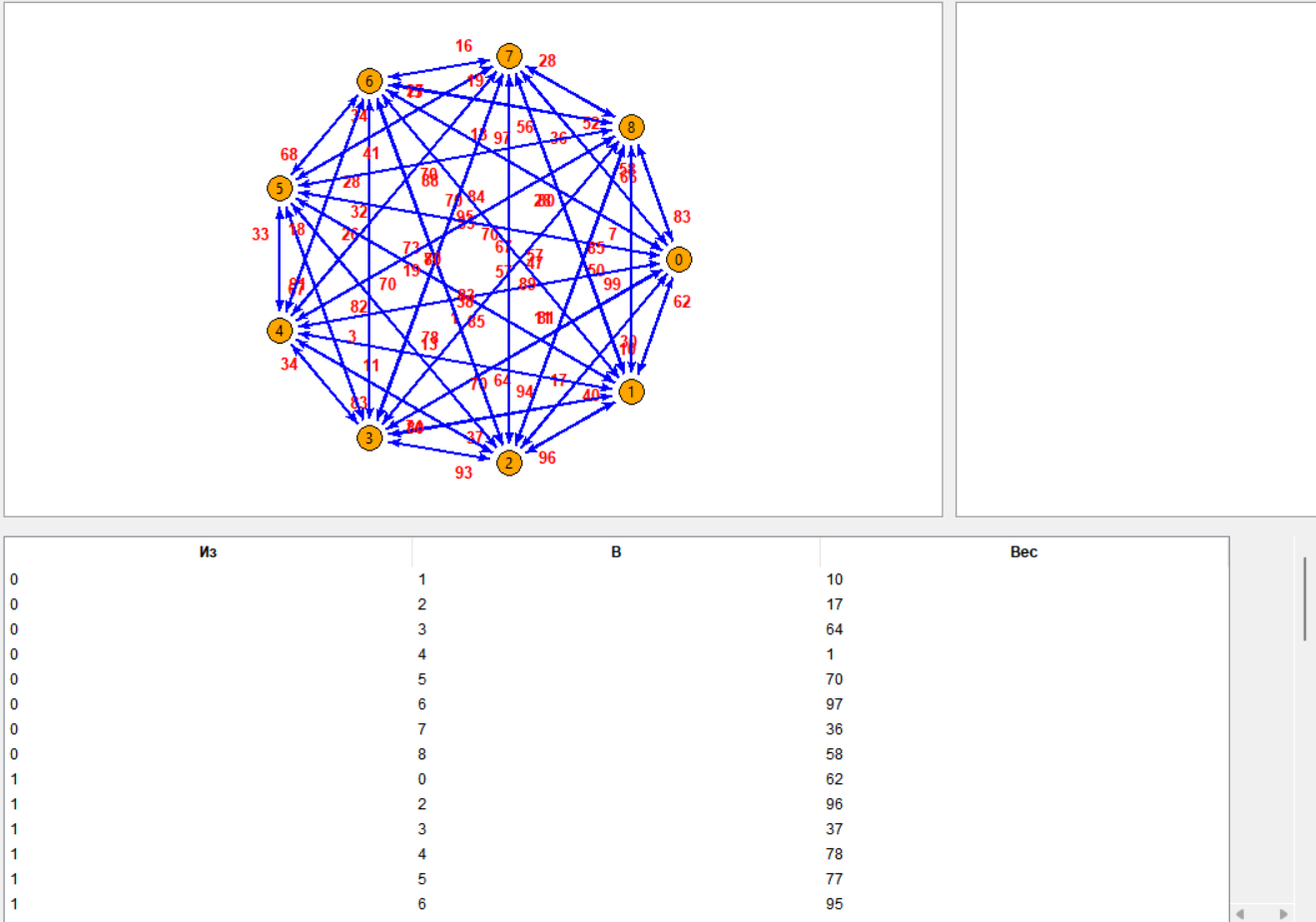
1. Рисуем граф и запускаем алгоритм:  
   ,

Рис. 3. Поле ввода графа.

1. Результат работы отжига.

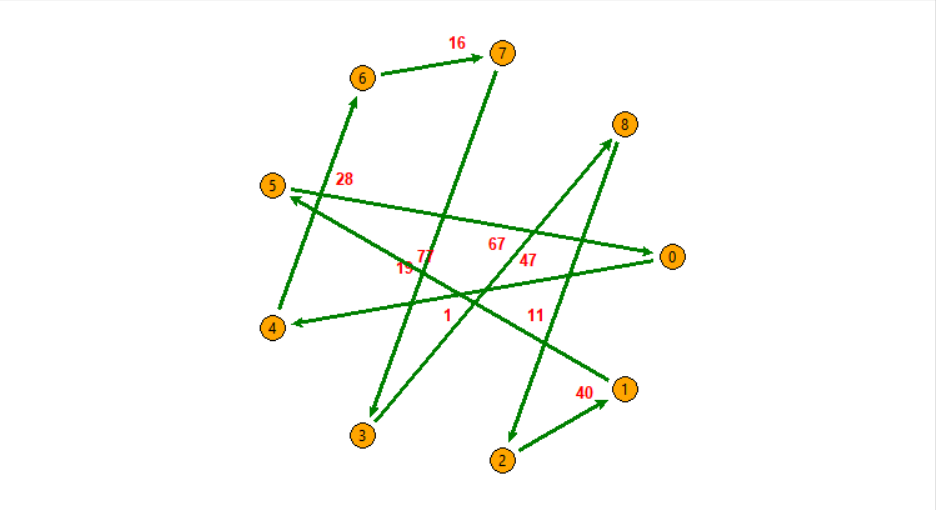


Рис. 4. Отображение результата.

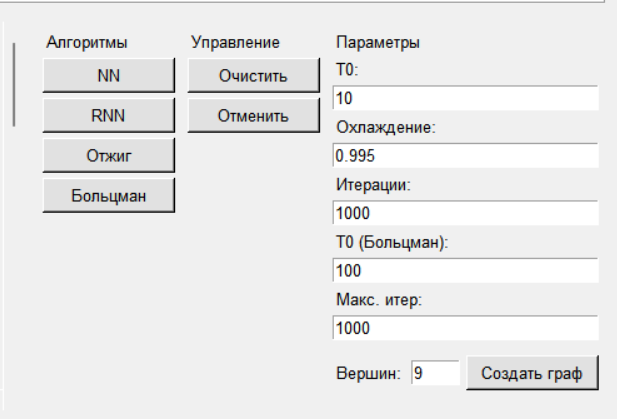
1. Поле управления  
   

Рис. 5. Управление.

# Анализ работы алгоритма

Для анализа работы алгоритма измерим время работы с модификацией и без на графах с разным количеством вершин.

Таблица 2. Замеры времени.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество вершин** | **Расстояние NN** | **Время NN** | **Расстояние RNN** | **Время RNN** |
| 10 | 230 | 0,000142 | 158 | 0,000728 |
| 30 | 344 | 0,000743 | 244 | 0,011605 |
| 60 | 421 | 0,002691 | 304 | 0,053689 |
| **Количество вершин** | **Расстояние Отжиг** | **Время Отжиг** | **Расстояние Больцмановский** | **Время Больцмановский** |
| 10 | 230 | 0,008572 | 190 | 0,0074 |
| 30 | 287 | 0014079 | 336 | 0,091061 |
| 60 | 347 | 0,167997 | 326 | 0,161265 |

Анализ показал, что оригинальный алгоритм с случайным выбором начальной вершины не всегда находит оптимальный путь, а иногда может застревать в более сложных графах, это связано с тем что при проходе от одной вершины алгоритм может застревать в локальных минимумах и не находить путь.

При использовании модификации с началом от каждой вершины мы рассматриваем больше вариантов путей, что даёт более точный результат, но за большее время чем оригинальный, это ожидаемое увеличение времени из за большего количества итераций. Несмотря на улучшенные результаты, работа алгоритма с модификацией тоже может застревать и не находить гамильтонов цикл.

По результатам работы отжига можно увидеть, что оригинальный алгоритм отжига работает лучше лучше чем оригинальный алгоритм ближайшего соседа, но за большее время, однако не лучше чем модификация ближайшего соседа. Больмановский отжиг показывает более хорошие результаты при маленькой температуре на небольший графах, иногда работает лучше чем модификация алгоритма ближайшего соседа.

# Вывод

# В ходе выполнения лабораторной работы была реализована программа для решения задачи коммивояжёра с использованием метода ближайшего соседа, имитации отжига и их модификаций. Программа позволяет находить кратчайший гамильтонов цикл в графе и визуализировать результат. По результатам работы наилучшим образом себя показывает RNN.

# Источники

1. tkinter - <https://metanit.com/python/tkinter/> (дата обращения 05.04.25)
2. Networkx - <https://networkx.org/documentation/stable/tutorial.html> (дата обращения 05.04.25)
3. Алгоритм имитации отжига - <https://habr.com/ru/articles/209610/> (дата обращения 04.04.25)
4. Деревья и графы Python - <https://sky.pro/wiki/python/derevya-i-grafy-v-python-osnovy-i-primery/> (дата обращения 04.04.25)