**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЁТ**

**по лабораторной работе №3**

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

**Студент гр. 23Б15-пу**

**Беляева А.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

**Оглавление**

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

Описание задачи............................................................................................................3

Теоретическая часть......................................................................................................4

Основные шаги программы...........................................................................................5

Блок-схема программы..................................................................................................6

Описание программы.....................................................................................................8

Рекомендации пользователя........................................................................................9

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

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

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

Вывод.............................................................................................................................11

Источники......................................................................................................................11

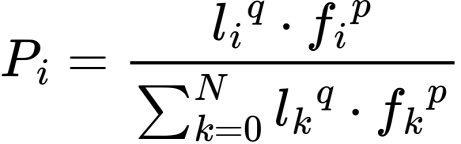
**Цель работы**

Формализация задачи коммивояжера с помощью муравьиного алгоритма. Написание алгоритма, реализующего контрольный пример на орграфе и находящего кратчайший гамильтонов цикл.

**Краткое описание муравьиного алгоритма.**

**Муравьиный алгоритм** (алгоритм оптимизации подражанием муравьиной колонии, англ. ant colony optimization, ACO) — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию. Первая версия алгоритма, предложенная доктором наук Марко Дориго в 1992 году, была направлена на поиск оптимального пути в графе.

В основе алгоритма лежит поведение муравьиной колонии — маркировка более удачных путей большим количеством феромона. Работа начинается с размещения муравьёв в вершинах графа (городах), затем начинается движение муравьёв — направление определяется вероятностным методом, на основании формулы вида:

,

где:

wps — вероятность перехода по пути i

wps — величина, обратная весу (длине) wps-го перехода,

wps — количество феромона на wps-м переходе,

q — величина, определяющая «жадность» алгоритма,

p— величина, определяющая «стадность» алгоритма.

Решение не является точным и даже может быть одним из худших, однако, в силу удачно подобранных эвристик, итерационное применение алгоритма обычно даёт результат, близкий к оптимальному.

**Краткое изложение**

В реальном мире муравьи (первоначально) ходят в случайном порядке и после нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым, и, как следствие, плотность феромонов остаётся высокой. Испарение феромонов также имеет функцию избежания стремления к локально-оптимальному решению. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.[3]

**Схема пошагового выполнения алгоритма**

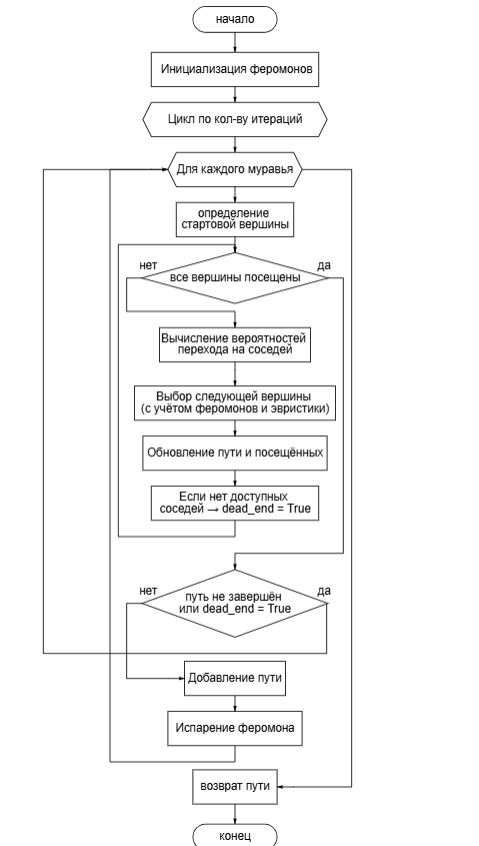


Рис 1. Блок-схема программы

**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 |

**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:

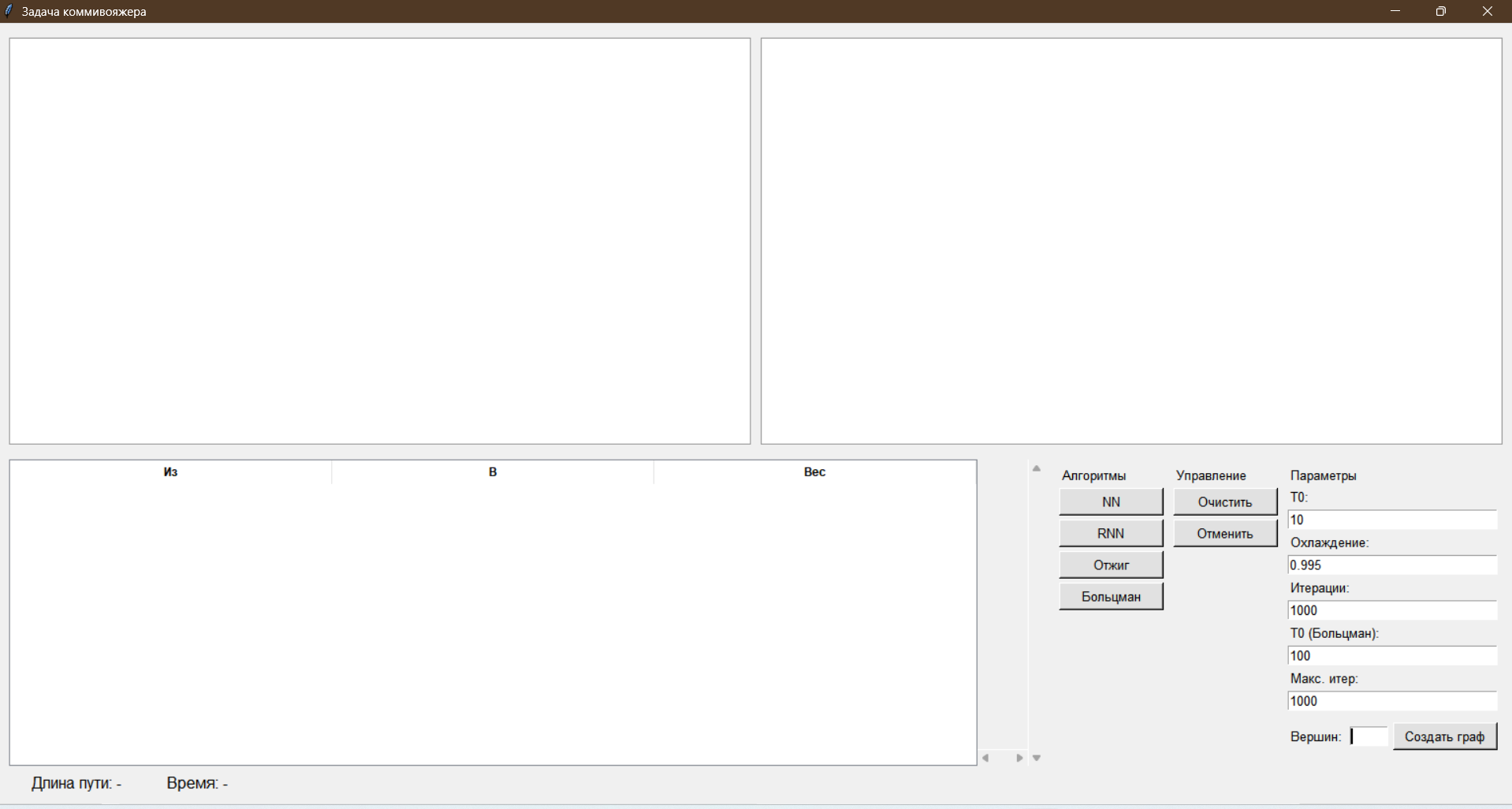


Рис. 2. Страт программы.

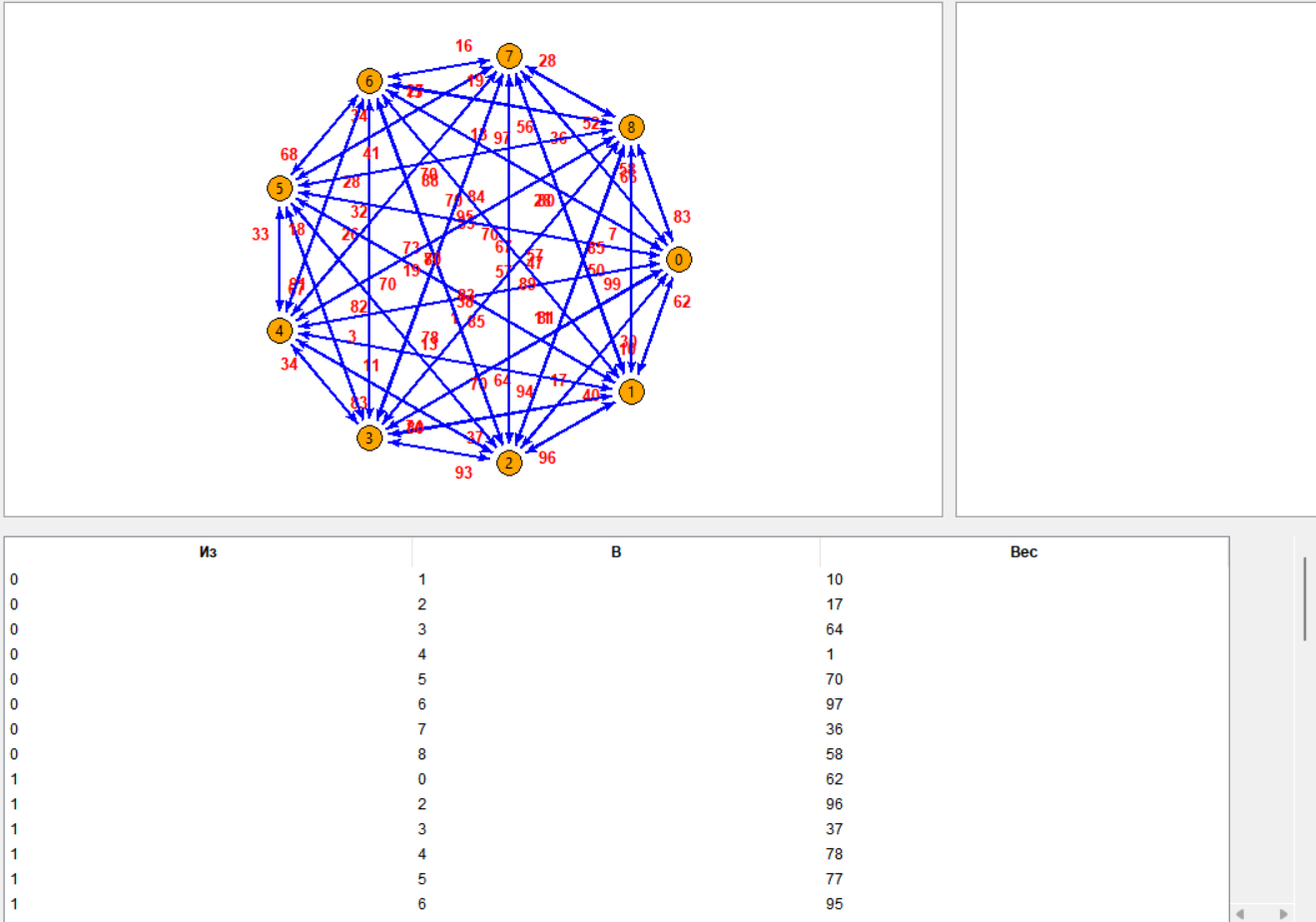
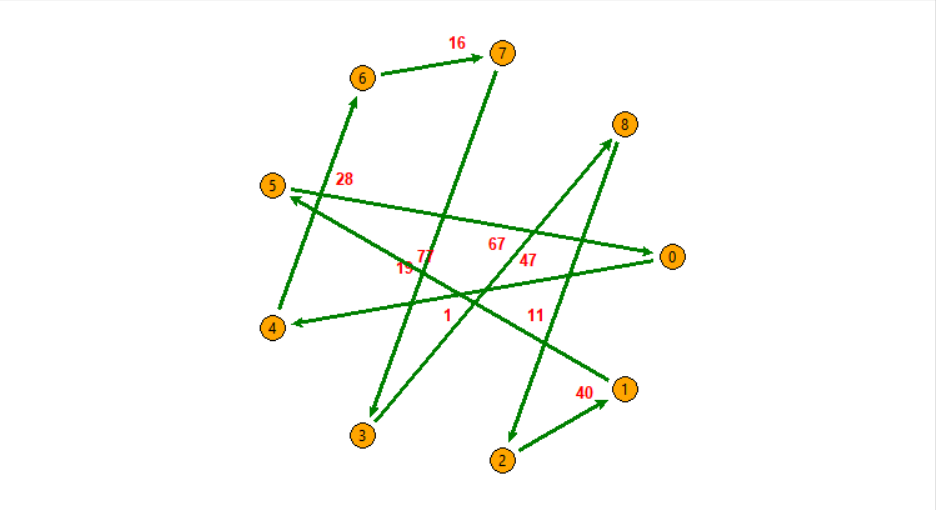
1. Рисуем граф и запускаем алгоритм:  
   ,

Рис. 3. Поле ввода графа.

1. Результат работы отжига.



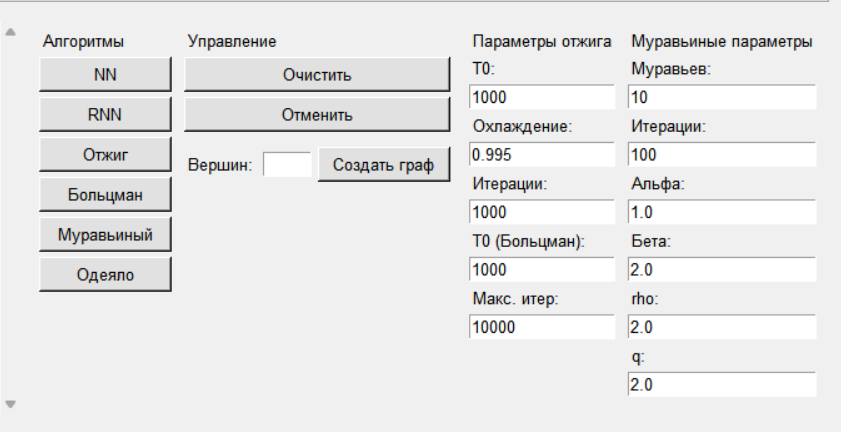
1. Поле управления  
   

Рис. 4. 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 | 228 | 0,000057 | 139 | 0,001242 |
| 30 | 271 | 0,000627 | 225 | 0,016252 |
| 60 | 423 | 0,002691 | 227 | 0,053689 |
| **Количество вершин** | **Расстояние Отжиг** | **Время Отжиг** | **Расстояние Больцмановский** | **Время Больцмановский** |
| 10 | 162 | 0,011324 | 149 | 0.05347 |
| 30 | 258 | 0,024817 | 259 | 0,106779 |
| 60 | 227 | 0,167997 | 344 | 0,161265 |
| **Количество вершин** | **Расстояние Муравьиный** | **Время Муравьиный** | **Расстояние Одеяло** | **Время Одеяло** |
| 10 | 131 | 0,065401 | 131 | 0,069712 |
| 30 | 195 | 1,4393 | 204 | 1,4293 |
| 60 | 217 | 105 | 224 | 104 |

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.

# **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.

# Алгоритм ближайшего соседа: реализован в методе nearest\_neighbor.

# Вывод

# В ходе выполнения лабораторной работы была реализована программа для решения задачи коммивояжёра с использованием метода ближайшего соседа, имитации отжига и муравьиного алгоритма их модификаций. Программа позволяет находить кратчайший гамильтонов цикл в графе и визуализировать результат.

# Источники

1. tkinter - <https://metanit.com/python/tkinter/> (дата обращения 20.04.25)
2. Networkx - <https://networkx.org/documentation/stable/tutorial.html> (дата обращения 20.04.25)
3. Муравьиный алгоритм - <https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%80%D0%B0%D0%B2%D1%8C%D0%B8%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC> (дата обращения 20.04.25)