**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №1**

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

**на тему «Решение задачи о коммивояжере с помощью метода ближайшего соседа».**

**Студент гр. 23Б15-пу**

**Беляева А.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

**Оглавление**

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

Краткое описание алгоритма метода ближайшего соседа...............................................................................................................................3

Схема пошагового выполнения алгоритма........................................................................................................................4

Формализация задачи....................................................................................................4

Описание программы....................................................................................................4

Рекомендации пользователя........................................................................................7

Рекомендации для программиста................................................................................8

Исходный код программы......................................................................................…..8

Спецификация программы...........................................................................................8

Контрольный пример....................................................................................................9

Анализ работы алгоритма.......................................................................................................................11

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

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

**Цель работы**

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

**Краткое описание алгоритма метода ближайшего соседа**

Метод ближайшего соседа[3] — это жадный алгоритм, который строит маршрут, начиная с произвольной вершины и на каждом шаге добавляя ближайшую непосещённую вершину. Алгоритм работает следующим образом:

1. Выбирается начальная вершина.
2. На каждом шаге из текущей вершины выбирается ближайшая непосещённая вершина.
3. Процесс повторяется, пока все вершины не будут посещены.
4. После посещения всех вершин алгоритм возвращается в начальную вершину, замыкая цикл.

**Схема пошагового выполнения алгоритма**

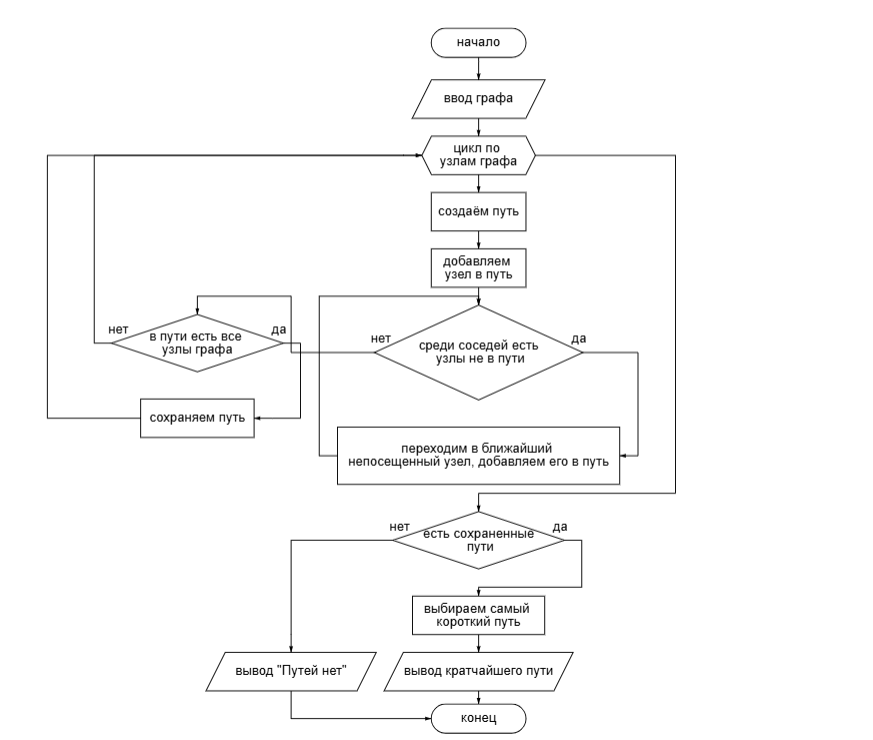


Рис 1. Блок-схема программы

**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 |
| two\_opt | Модификация | \_\_len\_\_, \_\_getitem\_\_ |
| calculate\_path\_distance | Вычисление дистанции | float|int |
| draw\_result | Отрисовка итогового графа | None |
| clear\_graph | Очистка графа | None |

# **Рекомендации для пользователя**

# Запустите программу, выполнив команду:

# python nearest\_neighbor.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.

# Модификация RNN.

**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:

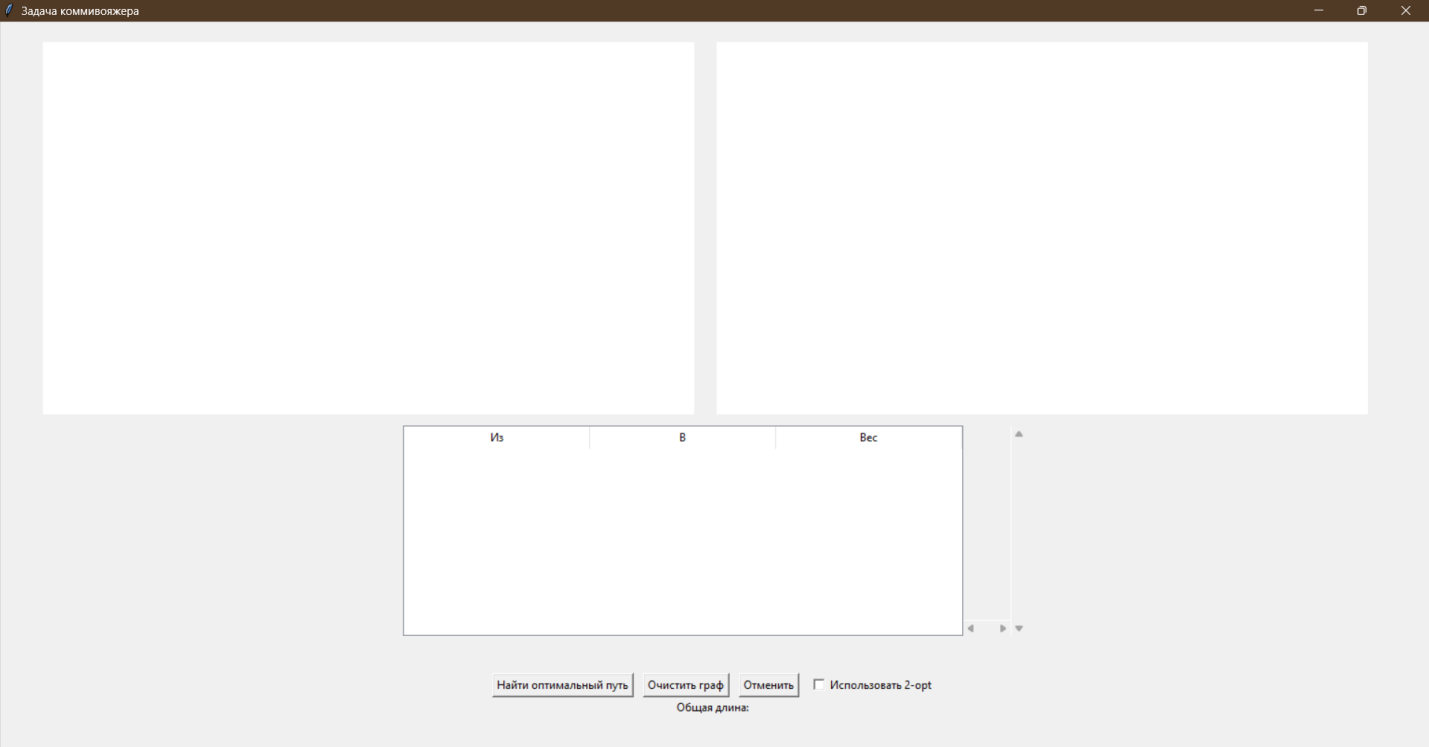


Рис. 2. Начальное окно

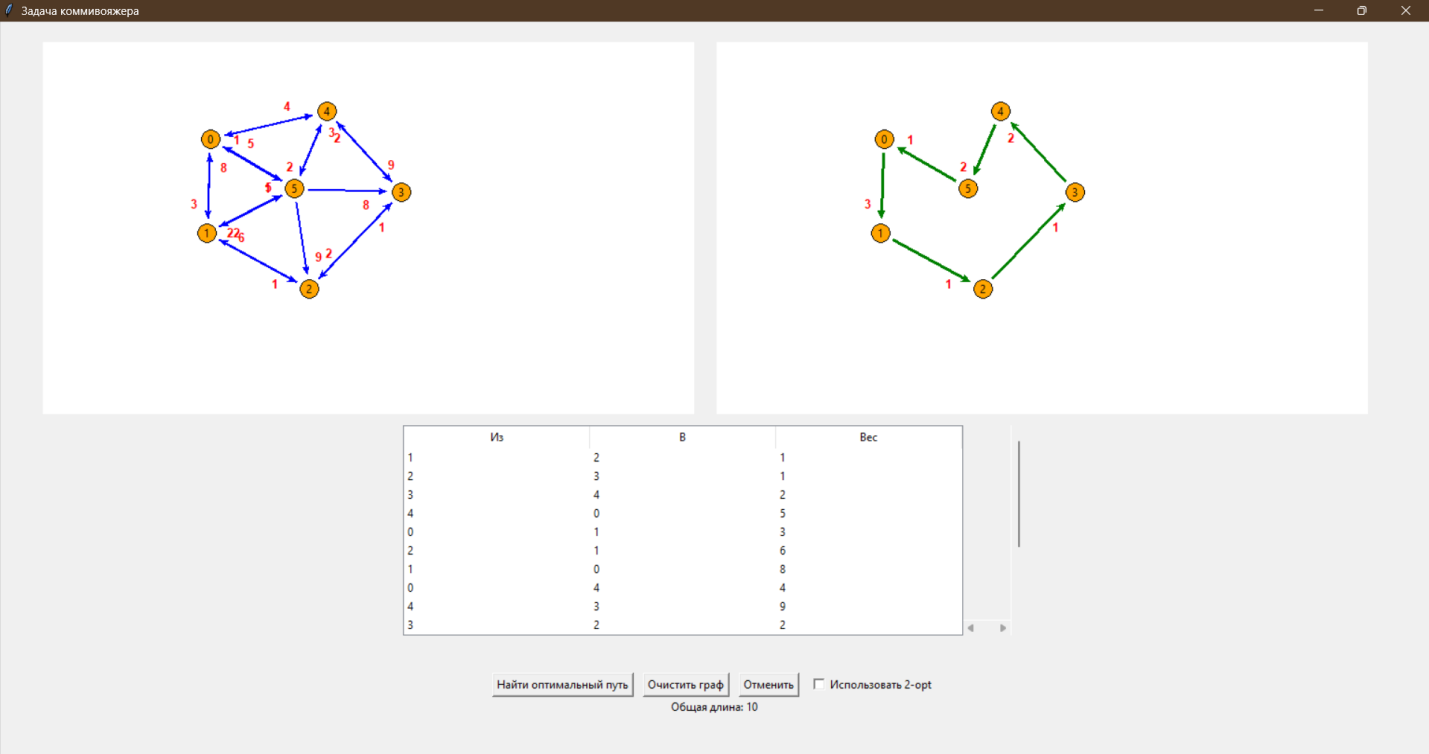
1. Рисуем граф и запускаем алгоритм:  
   

Рис. 3. Отображение графов

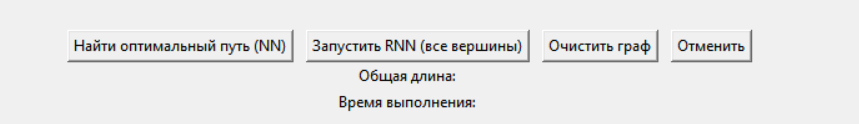
1. Можем использовать модификацию  
   

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 без модификации** | **Время без модификации** | **Расстояние с модификацией** | **Время с модификацией** |
| 6 | 6 | 0.0001 | 6 | 0.0006 |
| 17 | 20 | 0.0002 | 18 | 0,0020 |
| 30 | Пути не найдены | | 44 | 0,0031 |

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.

# Вывод

# В ходе выполнения лабораторной работы была реализована программа для решения задачи коммивояжёра с использованием метода ближайшего соседа. Программа позволяет находить кратчайший гамильтонов цикл в графе и визуализировать результат. Также была реализована модификация, которая находит решение лучше, чем оригинальный алгоритм.

# Источники

1. tkinter - <https://metanit.com/python/tkinter/> (дата обращения 12.03.25)
2. Networkx - <https://networkx.org/documentation/stable/tutorial.html> (дата обращения 12.03.25)
3. Метод ближайшего соседа - <https://habr.com/ru/articles/329604/> (дата обращения 10.03.25)
4. Деревья и графы Python - <https://sky.pro/wiki/python/derevya-i-grafy-v-python-osnovy-i-primery/> (дата обращения 10.03.25)