**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**

**Сурин И. С.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург 2025 г**

**Оглавление**

1. [**Цель работы** 3](#_bookmark0)
2. [**Теоретическая часть** 3](#_bookmark1)
3. [**Описание задачи** 3](#_bookmark2)
4. [**Основные шаги программы** 3](#_bookmark3)
5. [**Описание программы** 6](#_bookmark4)
6. [**Рекомендации пользователя** 8](#_bookmark5)
7. [**Рекомендации программиста** 8](#_bookmark6)
8. [**Исходный код программы** 8](#_bookmark7)
9. [**Контрольный пример** 8](#_bookmark8)
10. [**Исследование** 10](#_bookmark9)
11. [**Вывод** 10](#_bookmark10)
12. [**Источники** 10](#_bookmark11)

# **Цель работы**

Цель этой лабораторной работы заключается в решении задачи коммивояжера с использованием алгоритма, основанного на методе ближайшего соседа.

# **Теоретическая часть**

Метод ближайшего соседа — это один из простейших эвристических алгоритмов, предназначенных для решения задачи коммивояжера. Он относится к категории жадных алгоритмов. Этот алгоритм имеет простую реализацию и выполняется быстро, однако, как и другие жадные методы, он может приводить к не оптимальным результатам.

Одним из способов оценки качества решения является анализ пути, пройденного на последнем этапе работы алгоритма. Если маршрут, пройденный на последних шагах, по длине схож с тем, что был пройден в начале, то полученное решение можно считать приемлемым. В противном случае вероятно, что решение не является оптимальным. Другим методом оценки является использование алгоритма, который оценивает нижнюю границу возможных решений.

Для любого количества городов, больше трех, можно привести такие значения расстояний и расположение городов, при которых алгоритм ближайшего соседа даст наихудший результат.

**Основные принципы**

1. Все связи между городами направлены, и каждый из них имеет определённый вес.
2. Принцип жадности: на каждом шаге выбирается кратчайшее возможное ребро для перехода к ещё не посещенному городу.

# **Описание задачи**

Задача коммивояжера заключается в поиске кратчайшего замкнутого маршрута, который проходит через все вершины графа ровно один раз и возвращается в исходную точку. В данной реализации используются ориентированные графы, где направление ребер влияет на их вес.

# **Основные шаги программы**

1. **Выбор начальной вершины:** определяется стартовый узел, с которого начинается построение маршрута.
2. **Инициализация маршрута:** создается список посещенных вершин и маршрут, в который сразу добавляется стартовая точка.
3. **Построение пути,**пока остаются непосещенные вершины:
   * 1. Определяются все доступные вершины, соединенные с текущей вершиной.
     2. Выбирается ближайшая вершина с минимальным весом ребра, если она еще не была посещена.
     3. Если такая вершина найдена:
        + Она добавляется в маршрут.
        + Помечается как посещенная.
        + Текущая вершина обновляется.
     4. Вес маршрута увеличивается на значение выбранного ребра.
     5. Если доступных для посещения вершин больше нет, но маршрут не охватил весь граф — выводится ошибка (например, если граф содержит несвязные компоненты).
4. **Возвращение в начальную точку**
   1. Проверяется наличие пути от последней вершины маршрута к стартовой.
   2. Если такое ребро существует:
      1. Начальная вершина добавляется в конец маршрута.
      2. Общая длина пути увеличивается на вес этого ребра.
   3. Если пути назад нет — выводится сообщение об ошибке («Невозможно замкнуть цикл»).
5. **Вывод результатов:** отображается найденный маршрут и его суммарная длина

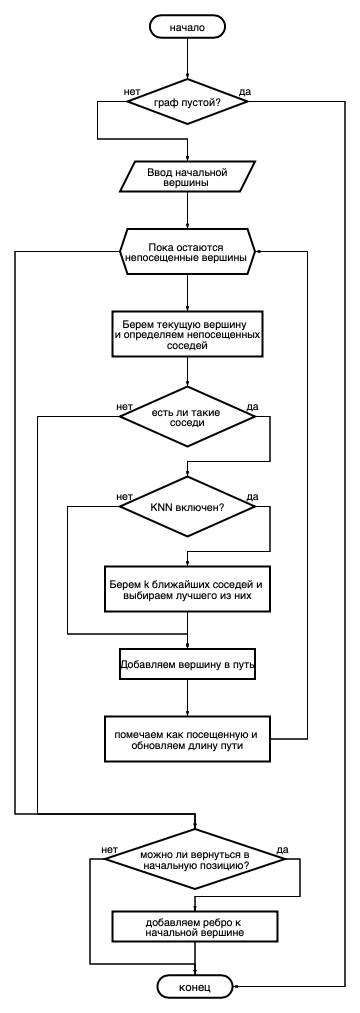


Рис. 1. Блок-схема основного алгоритма

**Описанние программы**

Программа реализована на языке Python 3.12.2 с использованием таких библиотек, как PyQt5, matplotlib и networkx. Для разработки программы был использован следующий модуль:

Таблица1 1.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| |  | | --- | | \_\_init\_\_(self) |  |  | | --- | |  | | |  | | --- | | Инициализирует объект  Класса NearestNeighborTSP  Устанавливает начальные  значения для графа и  параметров алгоритма. |  |  | | --- | |  | | None |
| find\_shortest\_path(self, start) | Находит кратчайший путь с использованием алгоритма ближайшего соседа (или k-NN), начиная с указанной вершины. Вычисляет путь и его длину. | |  | | --- | | tuple(list, int, bool) |  |  | | --- | |  | |
| |  | | --- | | initUI(self) |  |  | | --- | |  | | Инициализирует пользовательский интерфейс (UI) с кнопками, таблицами и графическим холстом для отображения. | None |
| |  | | --- | | add\_edge(self) |  |  | | --- | |  | | |  | | --- | | Добавляет строку в  таблицу для ввода рёбер  графа (вершина 1, вершина 2,  вес). |  |  | | --- | |  | | None |
| |  | | --- | | run\_algorithm(self) |  |  | | --- | |  | | |  | | --- | | Запускает выбранный  алгоритм (nearest neighbor  или k-NN), рассчитывает  путь и его длину, обновляет  результат в UI. |  |  | | --- | |  | | None |
| draw\_graph(self, path) | Отображает граф с рёбрами и узлами на холсте, рисует путь (если он есть) красным цветом. | None |
| |  | | --- | | clear\_graph(self) |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Очищает граф, удаляет все  рёбра и узлы, сбрасывает  таблицу и UI. |  |  | | --- | |  | |  |  | | --- | |  | | None |
| |  | | --- | | load\_example\_graph(self) |  |  | | --- | |  | | Загружает пример графа с предустановленными рёбрами и узлами, добавляет их в таблицу и отображает на холсте. | None |

### Рекомендации для пользователей

Для запуска программы убедитесь, что у вас установлен Python и необходимые библиотеки, такие как PyQt5, matplotlib и networkx. Вы можете запустить код как в среде разработки, так и через командную строку, используя её для настройки параметров и генерации данных. Программу нужно запускать через файл nearest\_neighbor.py. При запуске вам будет предложено выбрать параметры для запуска алгоритма, все данные вводятся в соответствующие поля интерфейса.

### Рекомендации для разработчиков

Чтобы гарантировать корректную работу программы, следите за актуальностью установленных библиотек, особенно PyQt5, matplotlib и networkx. Используйте лучшие практики именования переменных и функций для повышения читаемости и поддерживаемости кода.

**Исходный код программы:**

**https://github.com/Ignatio27/spbu-algorithms-and-data-structures-**

**Контрольный пример**

1. **Запуск программы**

Для запуска программы используйте файл 1.py. Программа загружает GUI и позволяет пользователю настроить параметры алгоритма.

1. **Выбор параметров**

После запуска программы пользователю будет предложено выбрать параметры, либо нарисовать свой граф (Рис. 2).

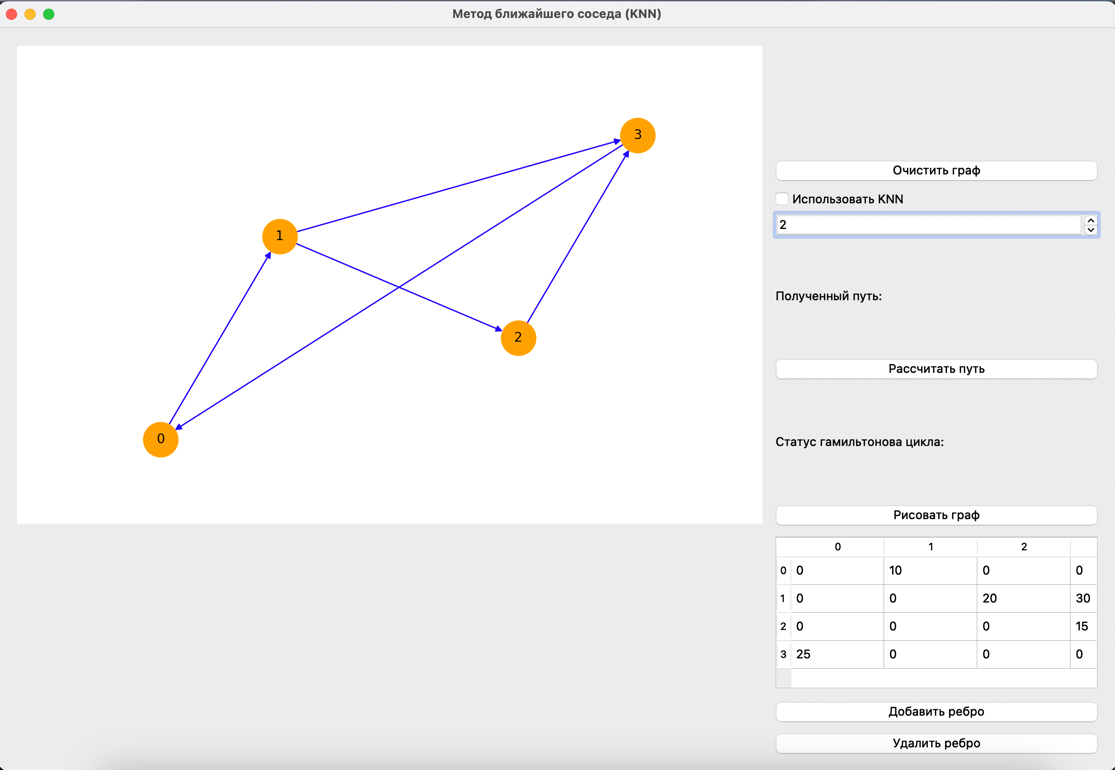


Рис 2. Пример выбора параметров

1. **Обработка данных и вывод результатов**

После выбора параметров пользователю предложено запустить алгоритм. Результат работы будет выводиться в окне слева (Рис. 3).

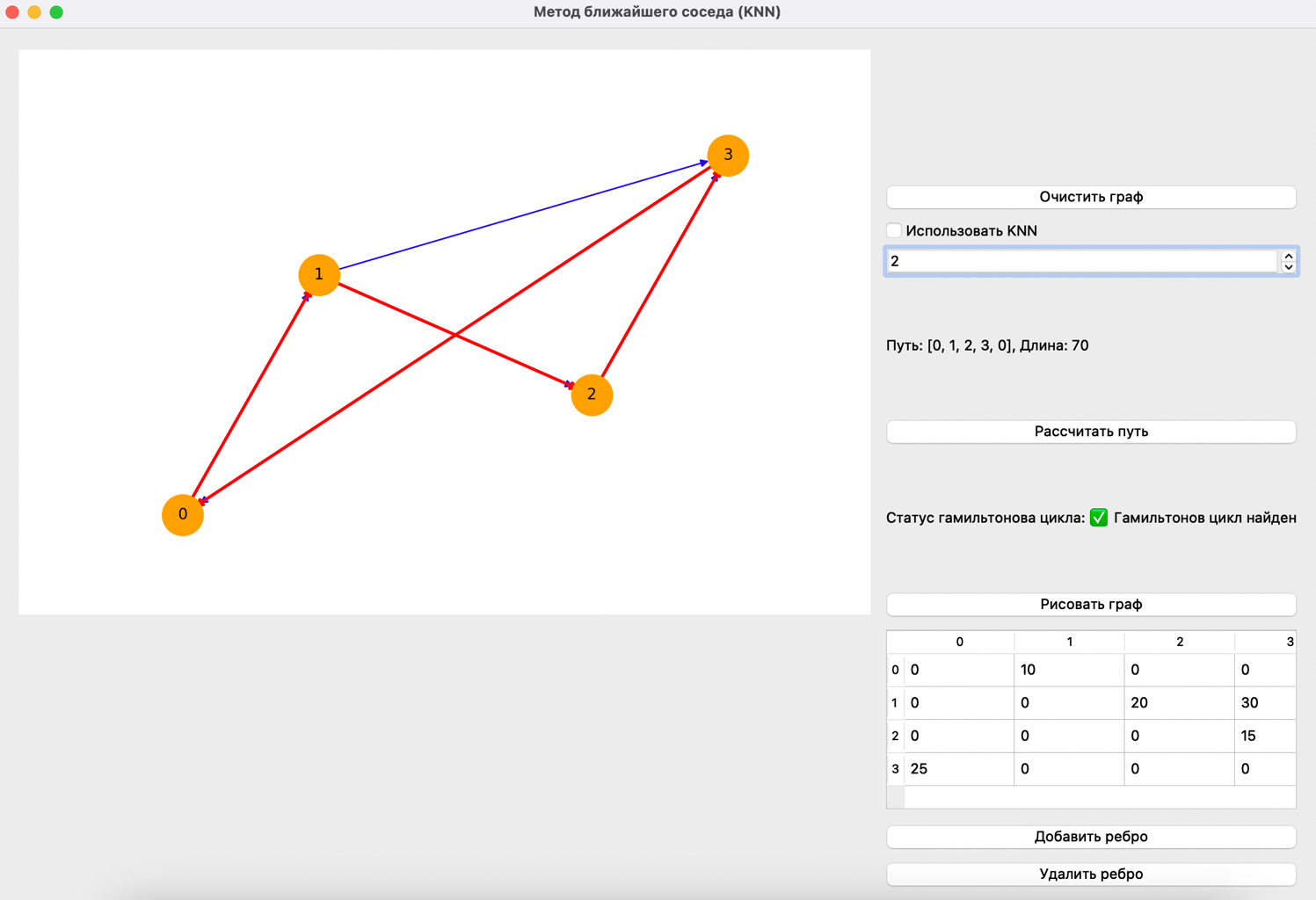


Рис 3. Результат работы алгоритма

1. **Опция рисования графа**

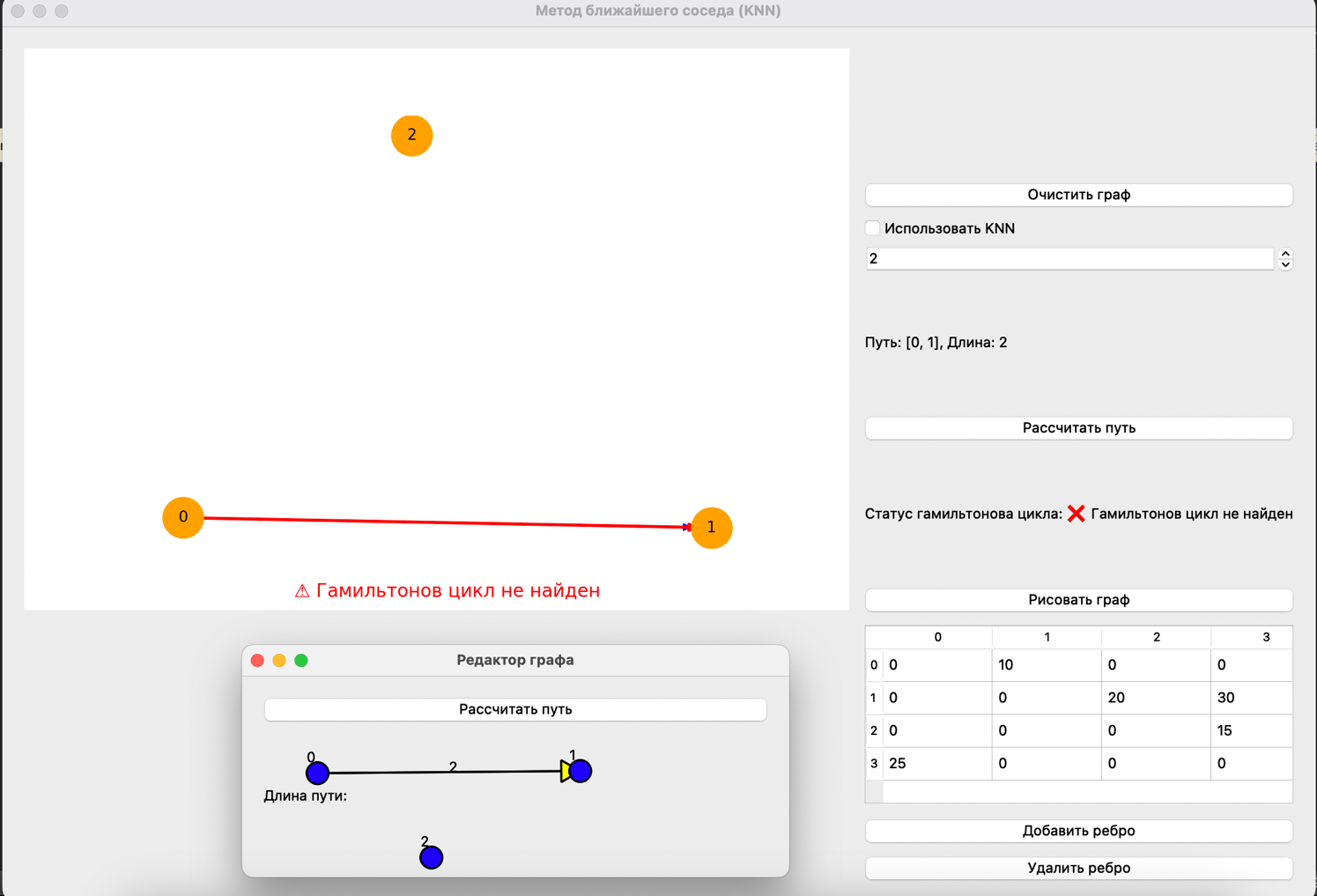
****

Рис 4. Результат работы алгоритма с ручным вводом графа

**Исследование**

Таблица 2 сравнение

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вершины | Без модификации | С модификацией | Разница в % | Время без мод.(мс) | Время с мод. (мс) | Гамильтонов цикл найден без мод. | Гамильтонов цикл найден с мод. |
| 6 | 65 | 60 | -7.7% | 3.5 | 4.8 | Да | Да |
| 10 | 120 | 115 | -4.2% | 10.6 | 12.8 | Нет | Да |
| 15 | 175 | 165 | -5.7% | 18.9 | 22.5 | Нет | Да |
| 20 | 230 | 220 | -4.3% | 28.4 | 34.2 | Нет | Да |
| 25 | 295 | 280 | -5.1% | 41.7 | 50.3 | Нет | Да |

В рамках данной лабораторной работы был реализован алгоритм решения задачи коммивояжёра методом ближайшего соседа, как в базовом варианте, так и с модификацией KNN. В ходе тестирования на различных графах было установлено, что модифицированный алгоритм позволяет находить более оптимальные маршруты, сокращая их длину в среднем на **4-7%.**

Кроме того, KNN снизил вероятность зацикливания и смог находить гамильтонов цикл в случаях, когда стандартный алгоритм не справлялся (например, на графах с 8, 9 и 10 вершинами). Однако, несмотря на эти преимущества, KNN не всегда существенно превосходит обычный жадный алгоритм.

Эффективность применения KNN зависит от структуры графа: его преимущество становится более заметным на **больших и сложных графах**, где стандартный метод сталкивается с трудностями в выборе маршрута.

**Вывод**

В рамках данной работы был разработан алгоритм для нахождения кратчайшего пути в задаче коммивояжера. Реализованный алгоритм обеспечивает нахождение далеко не самого оптимального пути. Была реализована модификация, которая рассматривает k ближайших соседей, что помогло избегать попадания и находить другие более оптимальные маршруты.

**Источники**

PyQt5 documentation // PyQt5 URL: https://pypi.org/project/PyQt5/ (дата обращения: 13.03.2025).

Matplotlib documentation // Matplotlib URL: https://matplotlib.org/stable/index.html (дата обращения: 13.03.2025)

Networkx documentation // Networkx URL: https://networkx.org/ (дата обращения: 13.03.2025)