

**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  
Антонян А. А.**

**Преподаватель  
Дик А.Г.**

**Санкт-Петербург  
2025 г.**

## Оглавление

1. Цель работы .....	3
2. Теоретическая часть .....	3
3. Описание задачи .....	3
4. Основные шаги программы .....	3
5. Описание программы .....	6
6. Рекомендации пользователя.....	8
7. Рекомендации программиста .....	8
8. Исходный код программы:.....	8
9. Контрольный пример .....	8
10. Исследование .....	10
11. Вывод.....	10
12. Источники .....	10

## **Цель работы**

Целью лабораторной работы является решение задачи о коммивояжере с помощью алгоритма, использующего метод ближайшего соседа.

## **Теоретическая часть**

Метод ближайшего соседа — один из простейших эвристических алгоритмов решения задачи коммивояжера. Он относится к категории жадных алгоритмов. Алгоритм прост в реализации, быстро выполняется, но, как и другие «жадные» алгоритмы, может выдавать неоптимальные решения. Одним из эвристических критериев оценки решения является правило: если путь, пройденный на последних шагах алгоритма, сравним с путём, пройденным на начальных шагах, то можно условно считать найденный маршрут приемлемым, иначе, вероятно, существуют более оптимальные решения. Другой вариант оценки решения заключается в использовании алгоритма нижней граничной оценки.

Для любого количества городов, большего трёх, в задаче коммивояжера можно подобрать такое расположение городов (значение расстояний между вершинами графа и указание начальной вершины), что алгоритм ближайшего соседа будет выдавать наихудшее решение.

## **Основные принципы:**

1. Все ребра имеют направление и веса.
2. Жадный выбор: на каждом шаге выбирается ребро с минимальным весом к не посещённому узлу.

## **Описание задачи**

Задача коммивояжера заключается в поиске гамильтонова цикла минимальной длины в графе. Веса ребер могут меняться в зависимости от направления.

## **Основные шаги программы**

1. Выбор начального узла.
2. Создание списка посещенных узлов и маршрута, в них добавляется стартовый узел

### 3. Построение маршрута:

Пока не посещены все узлы графа:

- Найти все соседние узлы текущего узла.
- Среди соседей выбрать узел с минимальным весом ребра, который ещё не посещён.
- Если такой узел найден:
  - Добавить его в маршрут.
  - Отметить, как посещённый.
  - Обновить текущий узел на выбранный.
- Увеличить общую длину маршрута на вес ребра.
- Если не посещённых узлов нет, но не все узлы графа охвачены — завершить с ошибкой (например, граф несвязный).

### 4. Возврат в начальный узел:

- Проверить наличие ребра от текущего узла к стартовому.
- Если ребро существует:
  - Добавить стартовый узел в конец маршрута.
  - Увеличить общую длину на вес этого ребра.
- Если ребра нет — вернуть ошибку ("Невозможно замкнуть цикл").

### 5. Результат: Вывод маршрута и его длины.

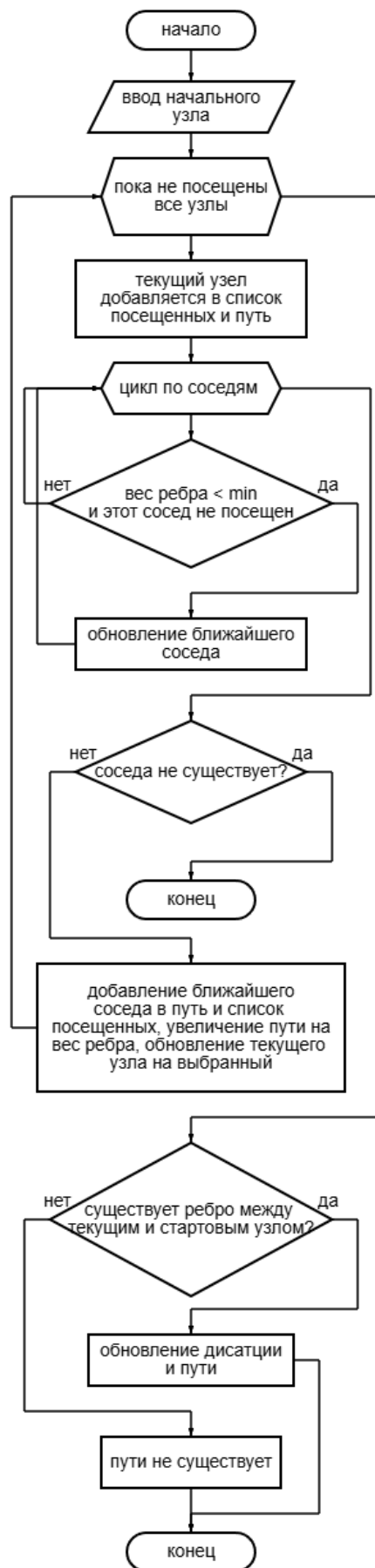


Рис. 1. Блок-схема основного алгоритма

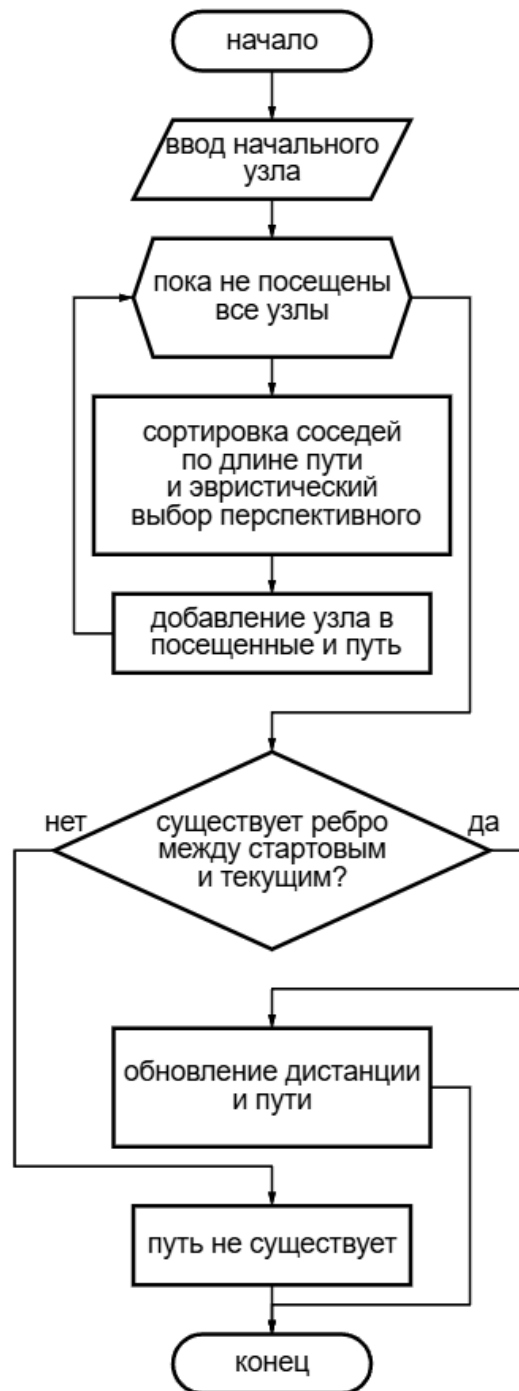


Рис. 2. Блок-схема модифицированного алгоритма

## Описание программы

Программная реализация написана на языке Python 3.12.2 с использованием библиотек PyQt5[\[1\]](#), matplotlib[\[2\]](#) и networkx[\[3\]](#). В процессе разработки программы использовался следующий модуль:

Таблица1 nearest\_neighbor.py

Функция	Описание	Возвращаемое значение
build_graph_from_input(self)	Строит граф на основе введенных данных, добавляет рёбра с весами и отображает сообщение об успехе или ошибке.	None
draw_graph(self, path_edges=None)	Отображает граф на холсте, рисуя узлы и рёбра. Также может выделять путь (красным цветом).	None
heuristic(self, node, visited)	Вычисляет эвристическое значение для узла, исходя из оставшихся непосещенных узлов, основываясь на минимальных весах рёбер.	float
knn_algorithm(self, start_node)	Алгоритм k-NN: находит путь, начиная с заданного узла, используя k ближайших соседей для выбора следующего узла.	tuple(list, int)
nearest_neighbor(self, start_node)	Алгоритм ближайшего соседа: находит путь, начиная с заданного узла, выбирая на каждом шаге ближайший сосед.	tuple(list, int)
run_algorithm(self)	Запускает выбранный алгоритм (k-NN или ближайшего соседа), вычисляет путь и его длину, отображает результат.	None
toggle_knn(self)	Переключает использование алгоритма k-NN, изменяя соответствующий флаг и текст на кнопке.	None
update_k(self, value)	Обновляет значение параметра K (количество ближайших соседей), когда изменяется значение в спинбоксе.	None

## **Рекомендации пользователя**

Для запуска программы убедитесь, что у вас установлен Python и необходимые библиотеки, такие как PyQt5[1] и matplotlib[2] и networkx[3]. Код можно запустить в среде разработки или через командную строку, используя консоль для настройки параметров и генерации данных. Запуск программы производится через файл nearest\_neighbor.py.

При запуске программы вам будет предложено выбрать параметры запускаемого алгоритма. Вводите ответы соответствующие поля в GUI.

## **Рекомендации программиста**

Для поддержания актуальности и работоспособности программы используйте последние версии библиотек, особенно PyQt5[1] и matplotlib[2] и networkx[3]. Применяйте практики надлежащего именования переменных и функций для улучшения читаемости кода.

## **Исходный код программы:**

**<https://github.com/ArseniiAntonin/spbu-algorithms-and-data-structures>**

## **Контрольный пример**

### **1. Запуск программы**

Для запуска программы используйте файл nearest\_neighbor.py.

Программа загружает GUI и позволяет пользователю настроить параметры алгоритма.

### **2. Выбор параметров**

После запуска программы пользователю будет предложено выбрать параметры (Рис. 3).



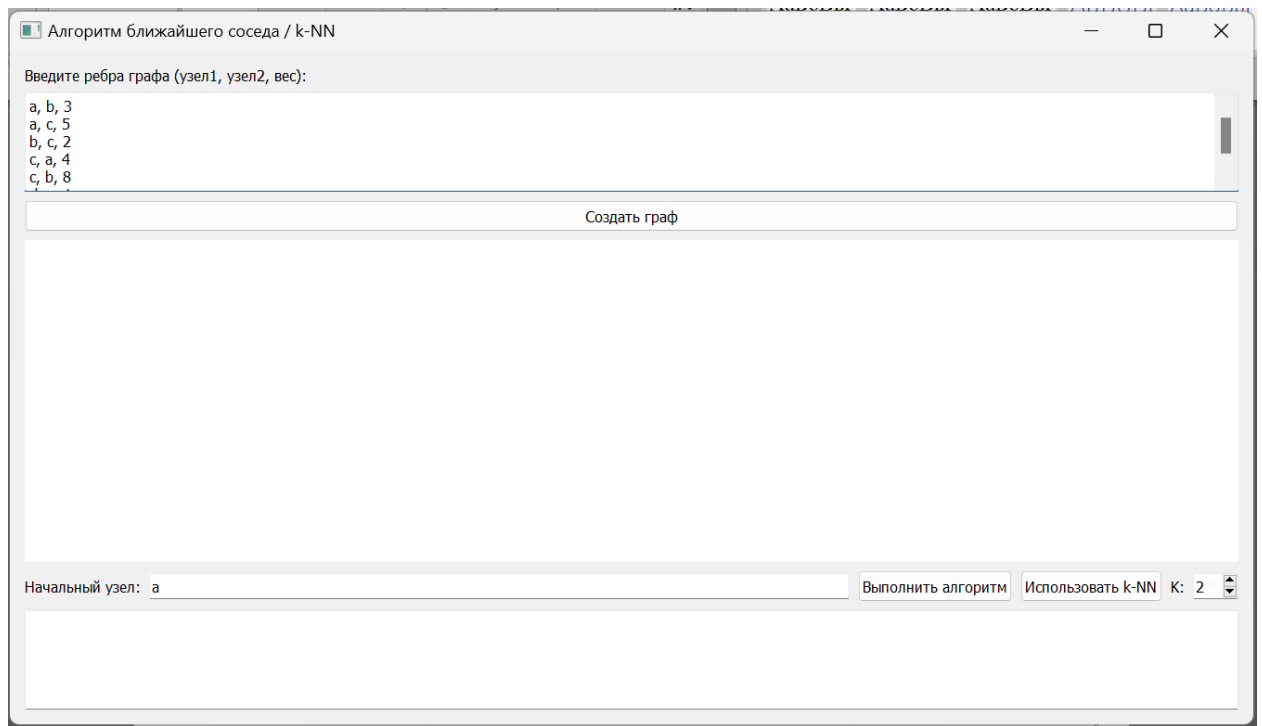


Рис 3. Пример выбора параметров

### 3. Обработка данных и вывод результатов

После выбора параметров пользователю предложено запустить алгоритм. Результат работы будет выводиться в окнах ниже (Рис. 4).

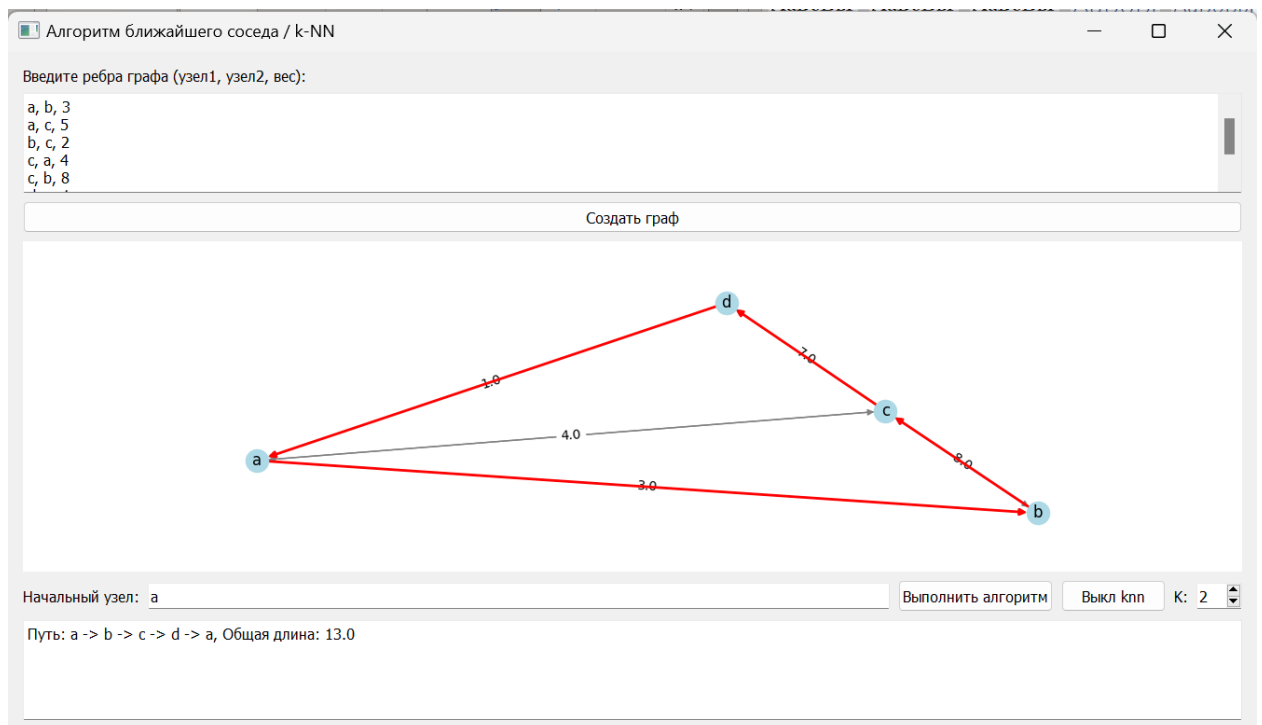


Рис 4. Результат работы алгоритма

## **Исследование**

В рамках данной лабораторной работы был создан алгоритм решения задачи коммивояжера методом ближайшего соседа с модификацией и без. В ходе тестирования программы было выявлено, что модифицированный алгоритм (KNN) может выбирать маршрут более гибко, уменьшает вероятность заикливания, повышает качество маршрута, но далеко не всегда превосходит обыкновенный жадный алгоритм. В зависимости от конкретных условий задачи использование KNN (модификация лучше работает на больших графах) может либо улучшить, либо незначительно повлиять на результат.

## **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 избежать попадания и находить другие более оптимальные маршруты.

## **Источники**

1. PyQt5 documentation // PyQt5 URL: <https://pypi.org/project/PyQt5/> (дата обращения: 5.03.2025).
2. Matplotlib documentation // Matplotlib URL: <https://matplotlib.org/stable/index.html> (дата обращения: 5.03.2025)
3. Networkx documentation // Networkx URL: <https://networkx.org/> (дата обращения: 5.03.2025)