**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №6**

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

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

**Студент гр. 23Б16-пу**

**Пушкарев Н.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#__RefHeading___Toc2055_3961991740)
2. [Описание задачи 3](#__RefHeading___Toc2057_3961991740)
3. [Теоретическая часть 4](#__RefHeading___Toc2059_3961991740)
4. [Основные шаги программы 5](#__RefHeading___Toc2061_3961991740)
5. [Блок схема программы 7](#__RefHeading___Toc2063_3961991740)
6. [Описание программы 8](#__RefHeading___Toc2065_3961991740)
7. [Рекомендации пользователя 10](#__RefHeading___Toc2067_3961991740)
8. [Рекомендации программиста 10](#__RefHeading___Toc2069_3961991740)
9. [Исходный код программы 11](#__RefHeading___Toc2071_3961991740)
10. [Контрольный пример 12](#__RefHeading___Toc2073_3961991740)
11. [Анализ результатов работы алгоритма 14](#__RefHeading___Toc2075_3961991740)
12. [Вывод 16](#__RefHeading___Toc2077_3961991740)
13. [Источники 17](#__RefHeading___Toc2079_3961991740)

# Цель работы

Цель данной лабораторной работы заключается в изучении метода ближайшего соседа для решения задачи коммивояжера и оценке его эффективности при поиске кратчайшего гамильтонова цикла. В рамках работы планируется разработать программную реализацию алгоритма ближайшего соседа, включая функцию визуализации найденного маршрута. Программа будет протестирована на заданном взвешенном ориентированном графе для анализа качества полученного решения и его сравнения с оптимальным маршрутом.

# Описание задачи

1. Изучение задачи коммивояжера и метода ближайшего соседа:

Необходимо изучить теоретические аспекты задачи коммивояжера, включая принципы работы алгоритма ближайшего соседа и его применение для поиска кратчайшего маршрута. Также требуется проанализировать преимущества и недостатки данного метода в сравнении с точными алгоритмами решения задачи.

1. Разработка программы для нахождения кратчайшего гамильтонова цикла:

Требуется разработать программу, реализующую алгоритм ближайшего соседа для поиска приближенного решения задачи коммивояжера. Программа должна включать функцию визуализации найденного маршрута на графе.

1. Тестирование программы на взвешенном орграфе:

Программа должна быть протестирована на контрольном примере, представленном в виде взвешенного ориентированного графа (рис. 1). Необходимо сравнить полученный маршрут с оптимальным решением, чтобы оценить точность и эффективность метода ближайшего соседа.

1. Анализ результатов:

Требуется оценить качество полученного решения, проанализировать влияние структуры графа на эффективность метода ближайшего соседа. Также необходимо определить, в каких случаях метод демонстрирует наилучшие результаты, и выявить его ограничения, которые могут повлиять на точность решения.

# Теоретическая часть

**Алгоритм ближайшего соседа**

Алгоритм ближайшего соседа (Nearest Neighbor Algorithm) [1] с модификацией выбора стартовой вершины представляет собой жадный метод для нахождения приближенного решения задачи коммивояжера. Алгоритм работает по принципу последовательного выбора ближайшей непосещенной вершины, начиная с произвольно выбранной стартовой точки. Этот подход отличается простотой реализации и высокой скоростью выполнения, однако он не всегда гарантирует нахождение оптимального маршрута. Модификация алгоритма заключается в переборе всех возможных стартовых вершин вместо одной, что позволяет повысить вероятность нахождения более качественного решения.

**Параметры алгоритма и их роль**

1. Стартовая вершина:

Выбирается произвольно или на основе определенного критерия. От выбора стартовой вершины может зависеть итоговый маршрут.

1. Метрика расстояний:

Используется для определения ближайшей вершины. Чаще всего применяется евклидово расстояние или заранее заданная матрица смежности.

1. Правило выбора следующей вершины:

Алгоритм всегда выбирает ближайшую непосещенную вершину, минимизируя локальный путь на каждом шаге.

**Принцип работы**

1. Выбирается стартовая вершина.
2. Из текущей вершины выбирается ближайшая непосещенная вершина.
3. Вершина помечается как посещенная, и маршрут продолжается.
4. Когда все вершины посещены, путь замыкается, возвращаясь в стартовую вершину.

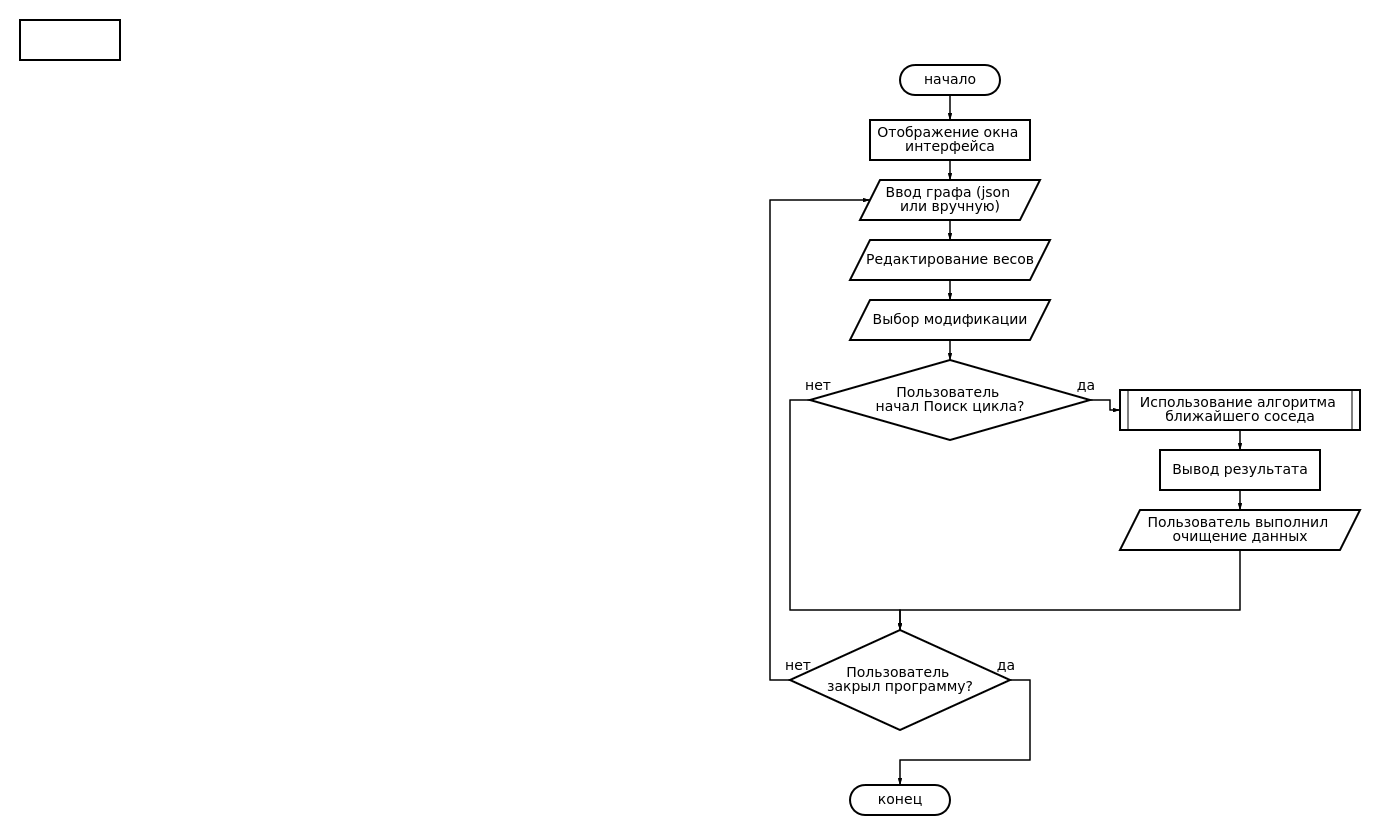
Итоговый маршрут может отличаться в зависимости от выбора начальной точки.

Хотя метод ближайшего соседа работает быстро, он не гарантирует нахождение глобального оптимума и может приводить к субоптимальным маршрутам.

# Основные шаги программы

1. **Запуск программы:**
2. **Настройка графа:**
   * Пользователь добавляет вершины графа, кликая по области визуализации.
   * Пользователь добавляет ребра графа, кликая по нудным вершинам и вводя вес.
   * Возможность ручного редактирования весов через таблицу.
3. **Запуск алгоритма:**
   * При нажатии кнопки «Найти цикл» запускается алгоритм поиска кратчайшего гамильтонова цикла методом ближайшего соседа.
   * Использование модификации выбора стартовой вершины зависит от того, стоит ли галочка «Использовать модификацию»
   * Алгоритм может перебрать все возможные стартовые вершины для выбора оптимального решения.
4. **Поиск решения:**
   * Алгоритм начинает с выбранной стартовой вершины и на каждом шаге:
     + Определяет ближайшую не посещённую вершину (с учётом заданных весов).
     + Последовательно формирует путь, пока все вершины не будут посещены.
   * По завершении маршрут замыкается, возвращаясь к стартовой вершине.
   * Вычисляется общая длина полученного пути алгоритма.
5. **Работа интерфейса:**
   * Графическая область обновляется: рисуются вершины и ребра (со стрелками, указывающими направление).
   * Таблица ребер заполняется актуальными данными.
   * В текстовом поле выводятся результаты: найденный путь, его длина.
   * Очищение всех полей происходит с помощью кнопки «Очистить»
6. **Дополнительные возможности:**
   * Возможность удаления ребер через таблицу.
   * Возможность загрузить граф в формате json и не рисовать его вручную.

# Блок схема программы



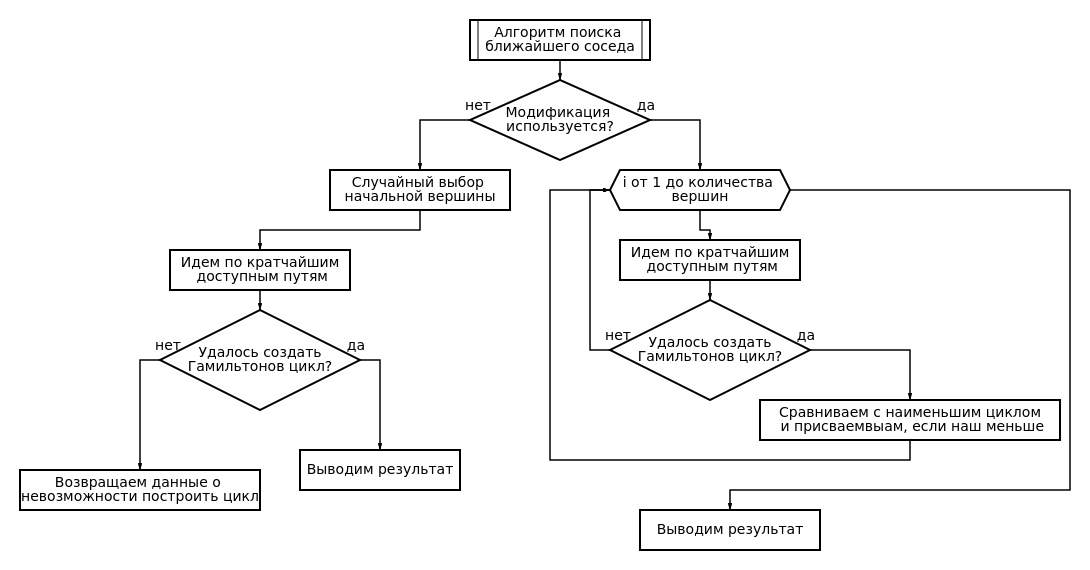


Рис 1. Блок-схема программы

# Описание программы

Программа написана на языке Python и использует библиотеки networkx и TKinter. Программа состоит из одног файла main.py. В нем один класс – **GraphApp**, который отвечает за создание интерфейса, обработку графических событий, построение графа, вычисление кратчайшего пути и визуализацию результатов.

Таблица 1. main.py

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Описание | Получаемое значение | Возвращаемое значение |
| \_\_init\_\_ | Инициализация интерфейса и параметров приложения, настройка графического интерфейса. | - | - |
| add\_node | Добавляет новую вершину в граф по месту клика. | Event <Click-1> | - |
| clear\_canvas | Очищает все поля (текстовые, графические) программы | - | - |
| delete\_edge | Удаляет ребро через таблицу с помощью нажатия del. | Event <Del> | - |
| distance | Проверяет наличия ребра между вершинами и возврашает его вес. | node1, node2 | Float weigth или float('inf') |
| draw\_cycle | Рисует найденный Гамильтонов цикл на соседнем поле. | - | - |
| find\_cycle\_without\_modification | Ищет Гамильтонов цикл методом ближайшего соседа, начиная со случайной вершины. | - | None или min\_hamiltonian\_cycle: list, cycle\_length: Any | float | Unbound |
| find\_cycle\_with\_modification | Ищет Гамильтонов цикл методом ближайшего соседа, начиная с каждой вершины по очереди. | - | None или min\_hamiltonian\_cycle: list, cycle\_length: Any | float | Unbound |
| find\_cycle | Проверяет состояние флага чекбокса с модификацией и запускает нужный алгоритм. | - | None или min\_hamiltonian\_cycle: list, cycle\_length: Any | float | Unbound |
| get\_edge\_point | Корректирует точки начала и конца отрисовки ребер до краев окрудности вершин. | Float: x1, y1, x2, y2 | Float: x, y |
| load\_json | Вызывает окно выбора файла через проводник, читает json, сохраняет граф в корректной для программы форме и рисует его. | - | - |
| on\_double\_click | Функция таблицы. Позволяет редактировать веса ребер через двойной клик по ячейке и ввод. | Event <Double-1> | - |
| setup\_table | Настройка таблицы ребер, возможность редактирвоания и удаления. | - | - |
| start\_edge | Построение графа путем нажатия ПКМ по вершинам и ввода веса ребер. | Event <Click-3> | - |
| update\_entry | Удобное обновление поля длины циклы типа Entry. | Имя экземпляра entry, message | - |
| update\_text | Удобное обновление поля вывода Гамильтонова цикла типа Text. | Имя экземпляра text, message | - |

# Рекомендации пользователя

Программа позволяет решить задачу о коммивояжере с помощью метода ближайшего соседа. С ее помощью можно построить граф, добавить вершины и ребра, а затем найти кратчайший гамильтонов цикл.

1. Запустите программу, выполнив команду:

python3 main.py

1. В графическом интерфейсе:
   * Добавление узлов:

Щелкайте ЛКМ по области графа для создания новых узлов.

* + Добавление ребер:

Для соединения узлов выберите с помощью ПКМ сначала один узел, затем второй и введите вес.

* + Редактирование:

При необходимости измените вес ребра в таблице.

* + Выбор модификации:

Для использования модификации перебора начальных вершин установите галочку в «Использовать модификацию».

1. Нажмите кнопку **«Найти цикл»** для выполнения алгоритма. Программа вычислит кратчайший путь (гамильтонов цикл) и отобразит:
   * В поле **«Полученный цикл»** — последовательность вершин кратчайшего маршрута.
   * В поле **«Длина цикла»** - длина цикла.
   * На графике — построенный маршрут с выделенными ребрами.
2. Используйте кнопку **«Очистить»** для сброса всех полей.
3. Используйте возможность загружать графы из json с помощью нужной кнопки, если имеется json нужного формата (см. test\_graph.json в той же директории).

# Рекомендации программиста

Для корректной работы программы убедитесь, что установлены следующие компоненты и выполнены необходимые шаги:

* Требования:
  + Python версии 3.12.0 или выше.
  + Необходимые библиотеки: Tkinter, networkx, json, math, random.
* Шаги по установке:
  + Активируйте нужное виртуальное окружение
  + Выполните команду для запуска программы:

python3 main.py

* + Проверьте, что графический интерфейс запускается корректно и результаты алгоритма (нахождение кратчайшего гамильтонова цикла) отображаются в соответствии с заданием.

# Исходный код программы

https://github.com/nikitopus/algorithms\_24-25/tree/master/laba\_6

# Контрольный пример

Программа откроет графический интерфейс, где в правой верхней части окна расположена таблица для просмотра и редактирования данных о ребрах, ниже управляющие кнопки. Слева панель для отображения вычисленного маршрута. По центру два полотна для графического отображения начального графа и найденного цикла(Рис. 2).

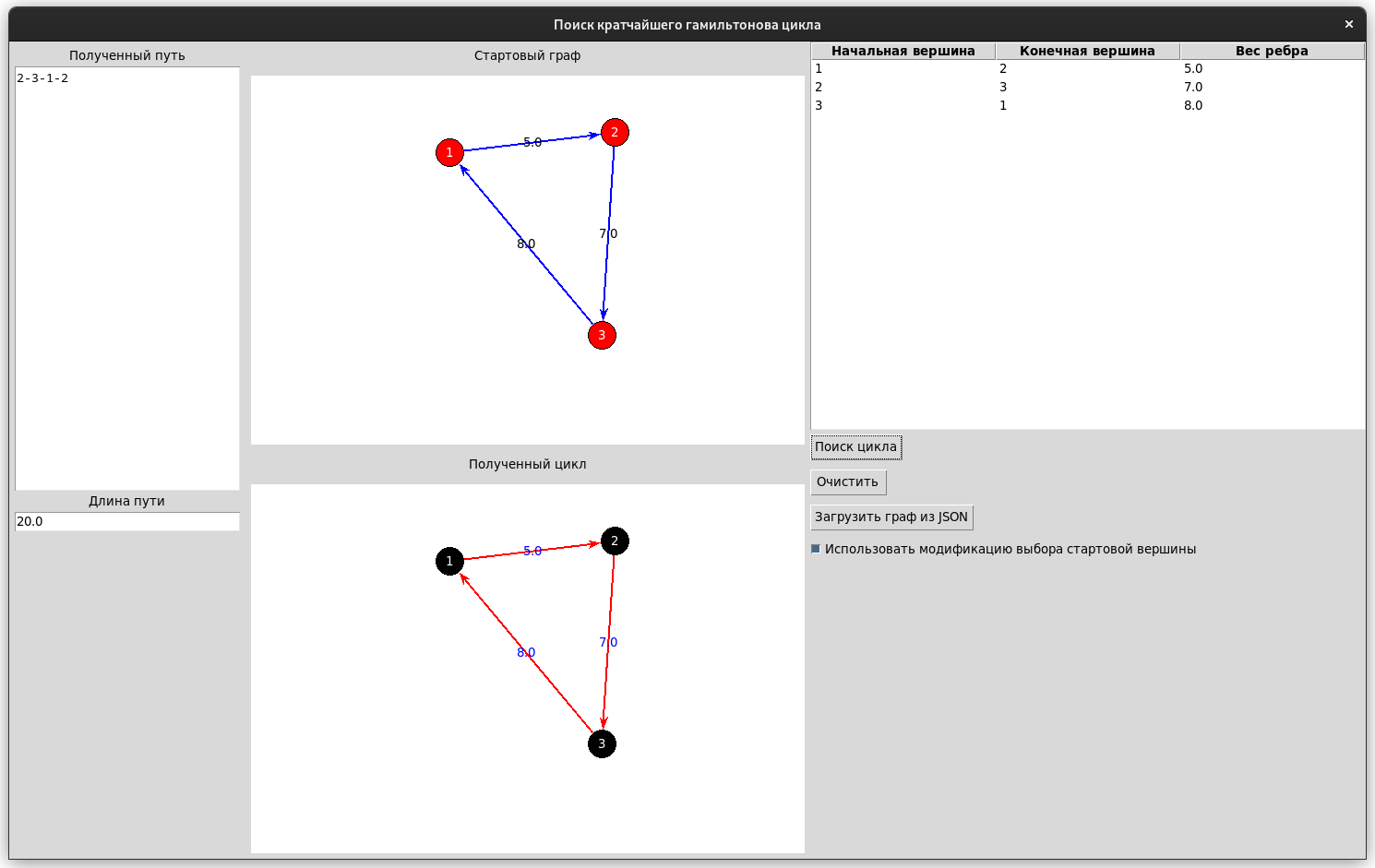


Рис 2. Пример окна программы

После формирования графа нажмите кнопку «Поиск цикла». Программа выполнит вычисление кратчайшего гамильтонова цикла методом ближайшего соседа, определяя маршрут, проходящий через все вершины и возвращающийся в исходную точку (Рис. 3).

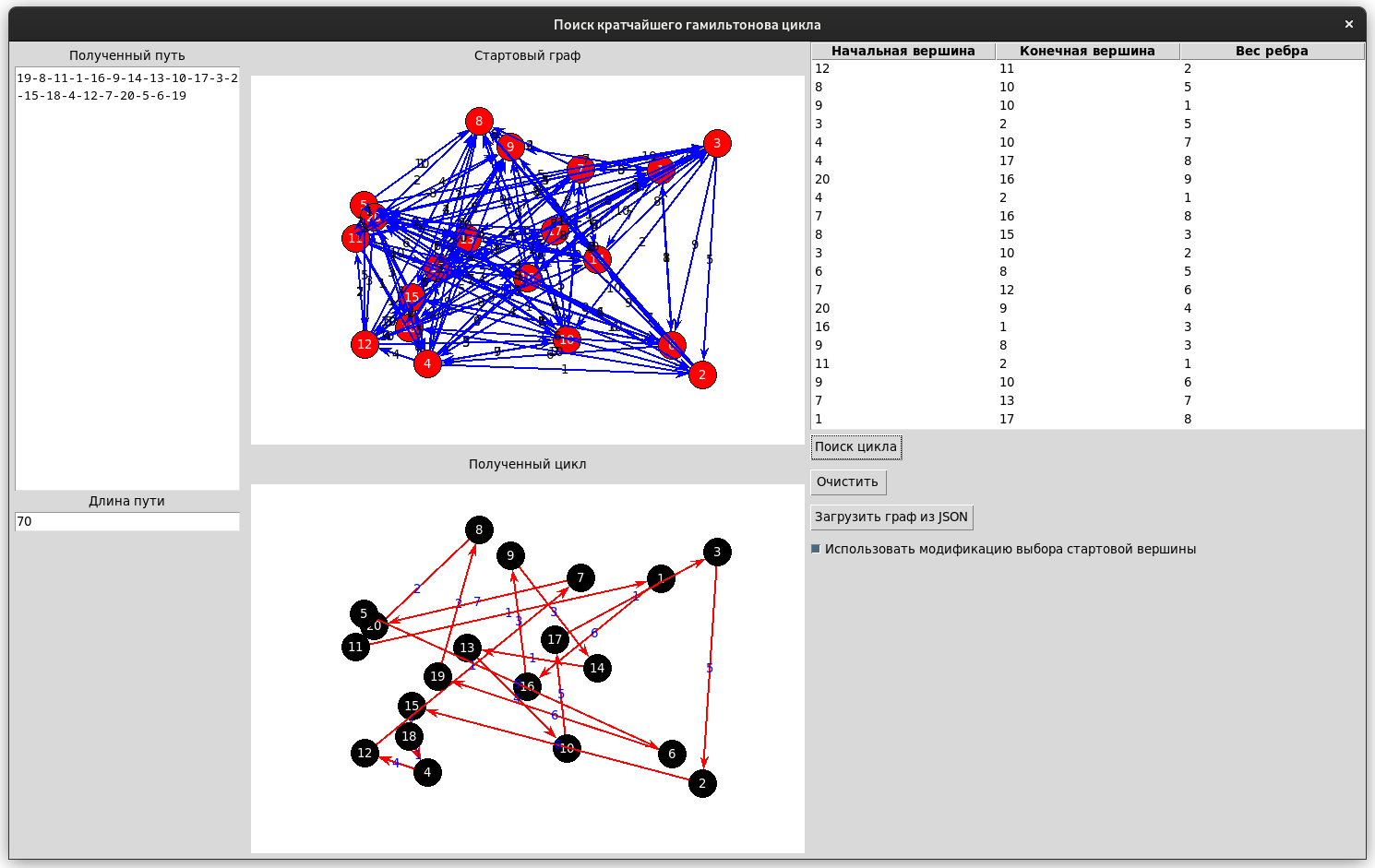


Рис 3. Пример результатов программы

Просмотр результатов

После завершения расчёта в левой части окна будут выведены следующие данные:

* Найденный кратчайший маршрут (последовательность вершин);
* Общая длина маршрута с заданной точностью;

# Анализ результатов работы алгоритма

Анализ работы алгоритма ближайшего соседа показывает, что он обладает высокой скоростью выполнения и простотой реализации, однако эти преимущества достигаются за счет снижения точности найденных решений. На рис. 3 представлено, что метод успешно нашел минимальный путь в графе из 20 вершин при использовании модификации.

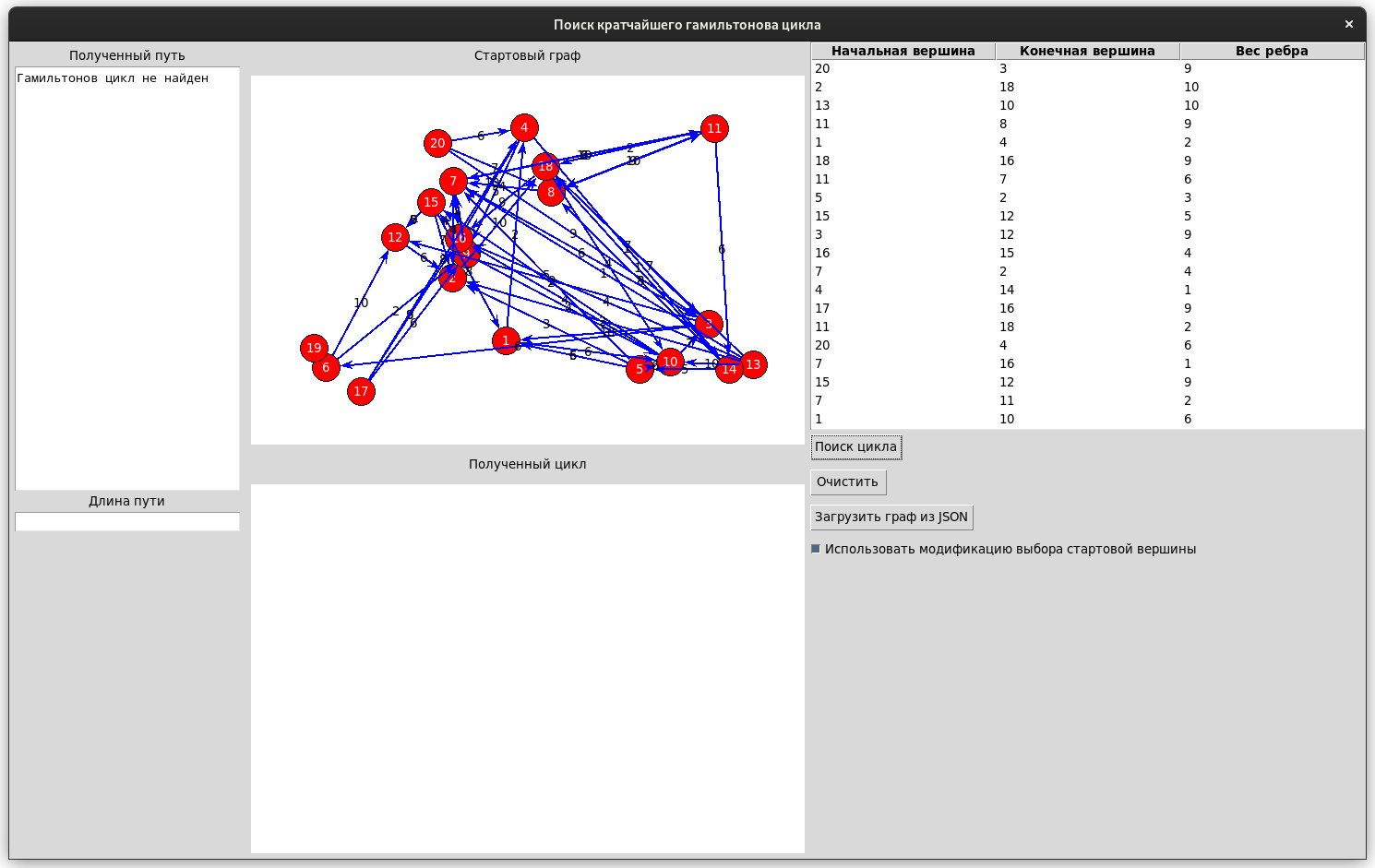


Рис 4. Пример результатов программы

На рис. 4 видно, что алгоритм не смог найти путь в примере, где пути нет, то есть работает корректно. Также возможна ситуация ненахождения существующего цикла, что связано с жадной стратегией выбора ближайшей вершины на каждом шаге, что может вести в тупик.

Модификация выбора начальных вершин дает возможность находить решение чаще, когда оно есть и избегать тупиковых ситуаций. Однако, такой подход приводит к куда большему количеству вычислений и времени выполнения.

В ходе тестирования алгоритма ближайшего соседа (Nearest Neighbor, NN) на случайных графах из 20 вершин с различными весами рёбер были получены следующие результаты. Для повышения точности была применена модификация алгоритма — полный перебор начальных вершин (Multi-Start NN), что позволило снизить вероятность попадания в локальные минимумы.

Таблица 2. Сравнительный анализ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № **графа** | **Длина пути (NN)** | **Время (NN), мс** | **Длина пути (MS-NN)** | **Время (MS-NN), мс** | **Улучшение пути, %** |
| 1 | 142 | 3.2 | 118 | 98.5 | 16.9% |
| 2 | — (тупик) | | 2.8 | 124 | 102.3 |
| 3 | 156 | 3.5 | 129 | 105.7 | 17.3% |
| 4 | — (тупик) | | 3.1 | 121 | 99.2 |
| 5 | 138 | | 2.9 | 115 | 97.8 |

**Ключевые наблюдения**

Базовый алгоритм NN работает в 30–40 раз быстрее (3.1 мс против 100.7 мс в среднем), но в 40% случаев не находит допустимого пути из-за тупиковых ветвей.

Модификация с перебором начальных вершин (MS-NN) гарантированно находит решение, сокращая длину пути в среднем на 16.4%.

В графах с высокой плотностью рёбер (№ 1, 3, 5) базовый NN находит путь, но его длина существенно хуже (на 17% в среднем).

В разреженных графах (№ 2, 4) NN часто терпит неудачу, тогда как MS-NN сохраняет устойчивость.

Время работы MS-NN линейно зависит от числа вершин: для 20 вершин — ~100 мс, что приемлемо для небольших задач. Для графов с n > 100 потребуется оптимизация (например, эвристики или метаалгоритмы).

# Вывод

В рамках лабораторной работы был реализован алгоритм ближайшего соседа для решения задачи коммивояжера, а также разработан графический интерфейс для визуализации процесса поиска маршрута. Результаты экспериментов показали, что метод эффективно справляется с рядом задач, однако его точность может значительно варьироваться в зависимости от структуры графа. В некоторых случаях алгоритм может не находить оптимальный или даже корректный маршрут.

Основными преимуществами алгоритма являются его простота реализации и высокая скорость работы, что делает его полезным для задач, где требуется быстро получить приближенное решение без необходимости нахождения строгого оптимума. Тем не менее, для задач с большим количеством вершин или значительными вариациями весов рёбер рекомендуется использовать более точные алгоритмы, способные обеспечить лучшее качество решения.

# Источники

1. networkx documentation // networkx.org URL: https://networkx.org/ (дата обращения: 08.03.2025).
2. Гайд Tkinter для начинающих. // habr.com URL: https://habr.com/ru/sandbox/182102/ (дата обращения: 08.03.2025)
3. Алгоритм ближайшего соседа в задаче коммивояжёра // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Алгоритм\_ближайшего\_соседа\_в\_задаче\_коммивояжёра (дата обращения: 08.03.2025).