**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Решение задачи о коммивояжере с помощью метода ближайшего соседа»**

**Студент гр. 22Б16-пу \_\_\_\_\_\_\_\_\_\_\_\_\_ Шарабарин М.С.**

**Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

# Оглавление

[Оглавление 2](#_Toc1709610656)

[Цель работы 3](#_Toc124072108)

[Описание задачи (формализация задачи) 3](#_Toc259608225)

[Теоретическая часть 3](#_Toc917857674)

[Основные шаги программы 4](#_Toc281540400)

[Сравнительный анализ: 4](#_Toc1124058329)

[Ограничения: 4](#_Toc494639595)

[Основные шаги программы 5](#_Toc1755642168)

[Блок схема программы 6](#_Toc968683058)

[Описание программы 7](#_Toc107777993)

[Рекомендации для пользователя 7](#_Toc672695833)

[Рекомендации для программиста 7](#_Toc1271854660)

[Исходный код программы 8](#_Toc1991648769)

[Контрольный пример 8](#_Toc554080984)

[Вывод 10](#_Toc165422677)

[Источники 10](#_Toc1695077998)

# Формулировка проблемы

**Цель работы** – изучить и реализовать на практике **метод ближайшего соседа** для решения задачи о коммивояжёре, создать **графический интерфейс** для интерактивного построения графа (вершин и рёбер), а также обеспечить **визуализацию матрицы смежности** и демонстрацию работы алгоритма в виде пошагового поиска пути.

.

# Метод ближайшего соседа

1. **Используемые структуры**:  
   Для хранения весов рёбер используется **матрица смежности**, представляющая собой таблицу размером n на n (где n — число вершин). В каждой ячейке таблицы хранится вес ребра между двумя вершинами. Если ребра нет, в ячейку записывается символ бесконечности.
2. **Стартовая вершина**:  
   Выбирается произвольная вершина (обычно это вершина с индексом 0). Назовём её v0.
3. **Инициализация**:  
   Вершина v0 помечается как посещённая, а текущая вершина устанавливается равной v0.
4. **Жадный выбор**:  
   На каждом шаге из текущей вершины ищется ближайшая непосещённая вершина (то есть вершина с наименьшим расстоянием от текущей). Переход совершается к этой ближайшей вершине, она добавляется в маршрут и помечается как посещённая.
5. **Продолжение**:  
   Шаг жадного выбора повторяется, пока остаются непосещённые вершины.
6. **Замыкание цикла**:  
   Если в конце алгоритма есть возможность вернуться из последней посещённой вершины обратно в стартовую (v0), маршрут замыкается и образует гамильтонов цикл. Если вернуться нельзя, маршрут остаётся незамкнутым, и алгоритм сообщает, что гамильтонов цикл не найден.

Таким образом, метод ближайшего соседа поэтапно добавляет в маршрут ближайшую непосещённую вершину, пока все вершины не будут посещены или пока не появится тупик. Результатом может быть полный цикл (если удалось вернуться к начальной вершине) или частичный путь.

# Описание задачи (формализация задачи)

Задача состоит в создании системы, которая:

1. Позволяет **интерактивно** строить граф (добавлять вершины и рёбра, задавать их веса, указывать ориентацию рёбер).
2. Предоставляет **графический интерфейс** для наглядного отображения построенного графа, включая отрисовку вершин и рёбер (со стрелками для ориентированного случая).
3. Реализует **метод ближайшего соседа** для решения задачи о коммивояжёре, показывая найденный маршрут и вычисленную длину пути.
4. Обеспечивает **визуализацию матрицы смежности** (в отдельном окне), где отражаются веса рёбер или символ «∞» при отсутствии ребра.
5. Позволяет **управлять** режимами построения графа (добавление рёбер, удаление вершин/рёбер, очистка всего графа) и **настройками** (ориентированный/неориентированный, взвешенный/невзвешенный граф).

Таким образом, система должна предоставить удобные средства для создания и редактирования графа, вычисления решения TSP методом ближайшего соседа и визуального анализа полученных данных.

# Спецификация программы

**Входные данные**

1. **Граф**, который пользователь формирует в интерфейсе:
   1. Набор **вершин**, добавляемых щелчком левой кнопки мыши на свободном месте Canvas (если «Режим добавления рёбер» выключен).
   2. Набор **рёбер**, добавляемых, когда «Режим добавления рёбер» включён: первый клик по вершине выбирает начальную вершину, второй клик по другой вершине формирует ребро.
2. **Ориентированность** (флажок «Ориентированный граф»):
3. Если флажок установлен, каждое добавленное ребро имеет направление только из i в j.
4. Если не установлен, добавляется и обратное ребро j в i.
5. **Взвешенность** (флажок «Взвешанный граф»):
6. Если флажок установлен, вес рёбер берётся из поля «Вес для нового ребра»; при ошибке чтения используется Евклидово расстояние.
7. Если флажок не установлен, всегда используется Евклидово расстояние между вершинами.

**Выходные данные**

1. **Отрисованный граф** (вершины и рёбра), отображаемый на Canvas.
2. **Результат решения TSP** (методом ближайшего соседа):
3. Порядок обхода вершин.
4. Суммарная длина пути.
5. Предупреждение, если не удалось обойти все вершины и замкнуть цикл.
6. **Матрица смежности** (при нажатии соответствующей кнопки), показывающая расстояния между всеми парами вершин или «∞», если ребра нет.

**Основные переменные**

1. vertices: список вершин. Каждая вершина — словарь с координатами и именем, например {"x": 100, "y": 80, "name": "0"}.
2. edges: список рёбер. Каждое ребро описывается кортежем (i, j, weight), где i и j — индексы вершин, а weight — числовой вес.
3. var\_directed: булев флажок, указывающий, будет ли граф ориентированным.
4. var\_weighted: булев флажок, указывающий, будет ли граф взвешенным.
5. entry\_weight: поле ввода текста, из которого берётся вес нового ребра (при включённом «Взвешанном»).
6. dist\_matrix: матрица смежности (список списков), заполняемая при решении TSP.

**Основные методы**

1. GraphCanvas.clear\_graph(): очищает списки вершин и рёбер, перерисовывает Canvas пустым.
2. GraphCanvas.add\_edge(i, j): добавляет ребро между вершинами i и j, вычисляет вес в зависимости от настроек (ориентированный/взвешенный).
3. GraphCanvas.redraw(): отрисовывает на Canvas все вершины и рёбра (включая стрелки и подписи весов).
4. TSPApp.solve\_tsp(): строит матрицу смежности и запускает метод ближайшего соседа, выводит результат в текстовое поле. Если не удалось обойти все вершины и замкнуть цикл, показывает предупреждение.
5. TSPApp.show\_adjacency\_matrix(): открывает новое окно с таблицей матрицы смежности.
6. TSPApp.load\_control\_example(): загружает контрольный пример (шесть заранее заданных вершин с координатами), формирует между ними рёбра и отображает на Canvas.

**Описание контрольного примера**

1. **Координаты вершин**:
   1. (100, 100), (200, 80), (300, 120), (250, 200), (150, 220), (100, 180).
   2. Каждой вершине присваивается индекс от 0 до 5 и имя (строковый индекс).
2. **Рёбра**: при загрузке примера программа автоматически соединяет каждую пару вершин (i < j) и рассчитывает вес рёбер как Евклидово расстояние (если граф неориентированный, добавляется и обратное ребро).
3. Это позволяет быстро протестировать алгоритм ближайшего соседа на заранее известном расположении вершин. Нажав «Решить (ближ. сосед)», пользователь видит порядок обхода и длину пути.

# Анализ результатов работы алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Граф | Длина пути | Время | Без модиф | С модиф |
| 1 | 58.02; 14.93 | 0.000007; 0.000003 сек | [0, 2, 5, 3, 4, 1, 0] | [4, 3, 2, 0] |
| 2 | 19.19; 31.17, | 0.000005; 0.000003 сек | [0, 1, 2, 3, 4, 0] | [3, 4, 1, 2, 0, 5, 3] |
| 3 | 24.03, 16.03 | 0.000008;  0.000005 сек | [0, 3, 1, 2, 0] , | [5, 0, 3, 1, 2] |
| 4 | 21.01; 25.02 | 0.000005; 0.000004 сек | [0, 1, 4, 3, 0] | [3, 0, 1, 4, 5] |
| 5 | 37.25; 10.69 | 0.000005; 0.000002 сек | [0, 3, 4, 5, 2, 1, 0] | [4, 5, 3, 4] |

**Результаты на контрольном примере**

* При загрузке шести заранее заданных вершин (координаты (100,100)(100,100)(100,100), (200,80)(200,80)(200,80), (300,120)(300,120)(300,120), (250,200)(250,200)(250,200), (150,220)(150,220)(150,220), (100,180)(100,180)(100,180)) и запуске метода ближайшего соседа алгоритм находит путь за 0.000002 сек.
* Если граф полностью связный (каждая вершина соединена с каждой), алгоритм без проблем посещает все вершины. Если в конце существует ребро обратно в стартовую вершину, формируется гамильтонов цикл.
* В случае, когда какой-либо вершине не удаётся найти путь к непосещённым вершинам или нет дороги обратно в начало, выводится предупреждение о том, что гамильтонов цикл не найден.

· **Улучшение результата алгоритма**

* Метод ближайшего соседа является жадной эвристикой, поэтому он не гарантирует нахождения оптимального (минимального) пути.
* Для получения более качественного решения можно:
  1. **Перебирать разные стартовые вершины**. Если всегда начинать с 0, иногда результат может быть хуже. Запуск алгоритма из каждой вершины в качестве старта и выбор лучшего пути улучшит итог.
  2. **Применить метод «два раза ближайший сосед»** (Two-way Nearest Neighbor), когда обход идёт не только вперёд, но и назад.
  3. **Использовать дополнительную эвристику**, например, **метод вставки** (Insertion), **метод улучшения** (2-оп, 3-оп) для локального улучшения уже найденного маршрута.
  4. **Рассмотреть другие алгоритмы** (например, муравьиный или генетический), если нужно более точное решение на большом числе вершин.

· **Процесс отладки алгоритма**

* На этапе отладки добавлялись выводы отладочной информации: порядок обхода, длины промежуточных рёбер, текущее состояние массива visited.
* Проверялась корректность работы с матрицей смежности: в частности, вывод в таблицу и совпадение значений с реальными расстояниями на Canvas.
* Проверялась реакция программы на различные сценарии:
  + Граф с одним ребром,
  + Разреженный граф (где некоторые вершины недостижимы),
  + Полный граф (где каждая вершина связана со всеми).
* В результате были устранены ошибки, при которых алгоритм не выводил предупреждение о невозможности замкнуть цикл или не учитывал правильный вес ребра из поля ввода.

· **Графики и комментарии**

* Ниже можно приводить графики (скриншоты интерфейса), иллюстрирующие:
  1. **Контрольный пример** (6 вершин), на котором видно расположение вершин и рёбер, и результат (путь).
  2. **Ситуацию, когда алгоритм не может завершить цикл** (например, убрав ребро из последней вершины к начальной). Программа выводит путь и предупреждение, что гамильтонов цикл не найден.
* На этих графиках видно, как программа выделяет вершины, рисует стрелки (при ориентированном графе) и подписи весов (при взвешенном графе).

# Блок схема программы

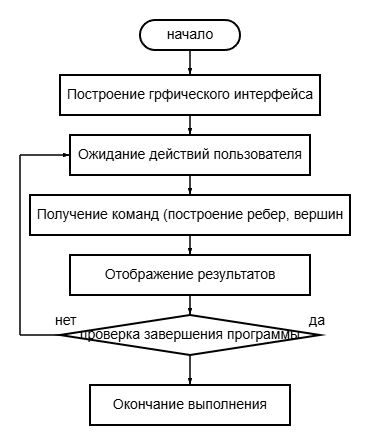
****

Рис 1. Блок-схема основной программы

# Контрольный пример

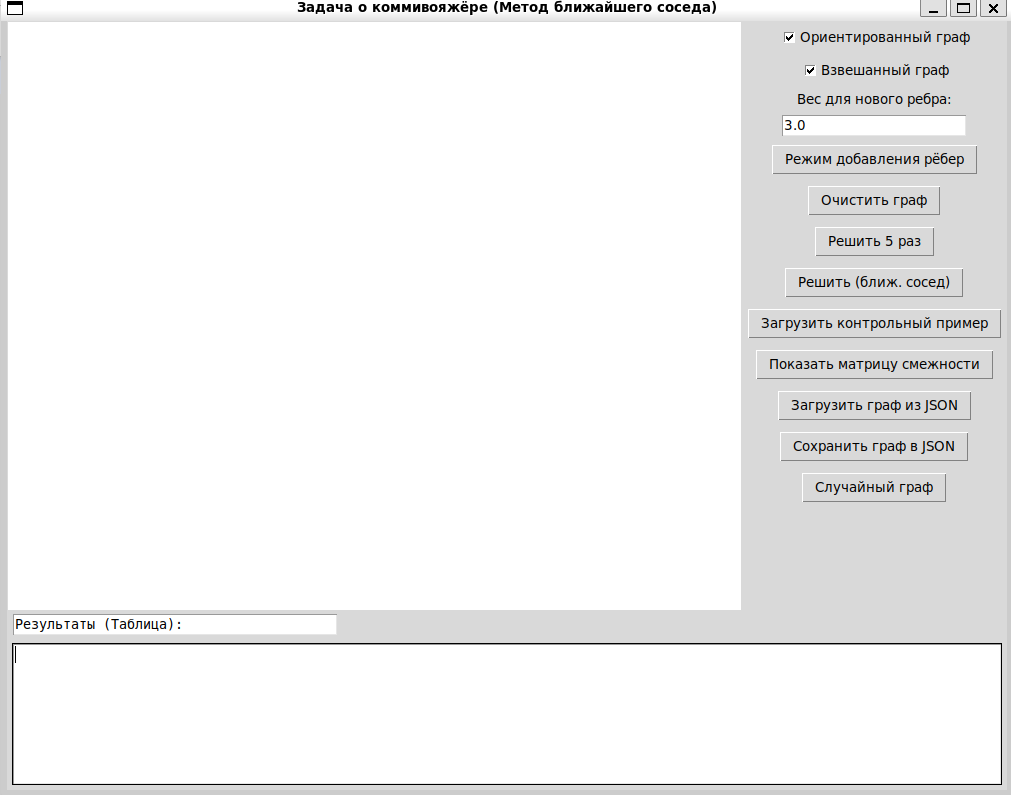


Рис 1. Интерфейс

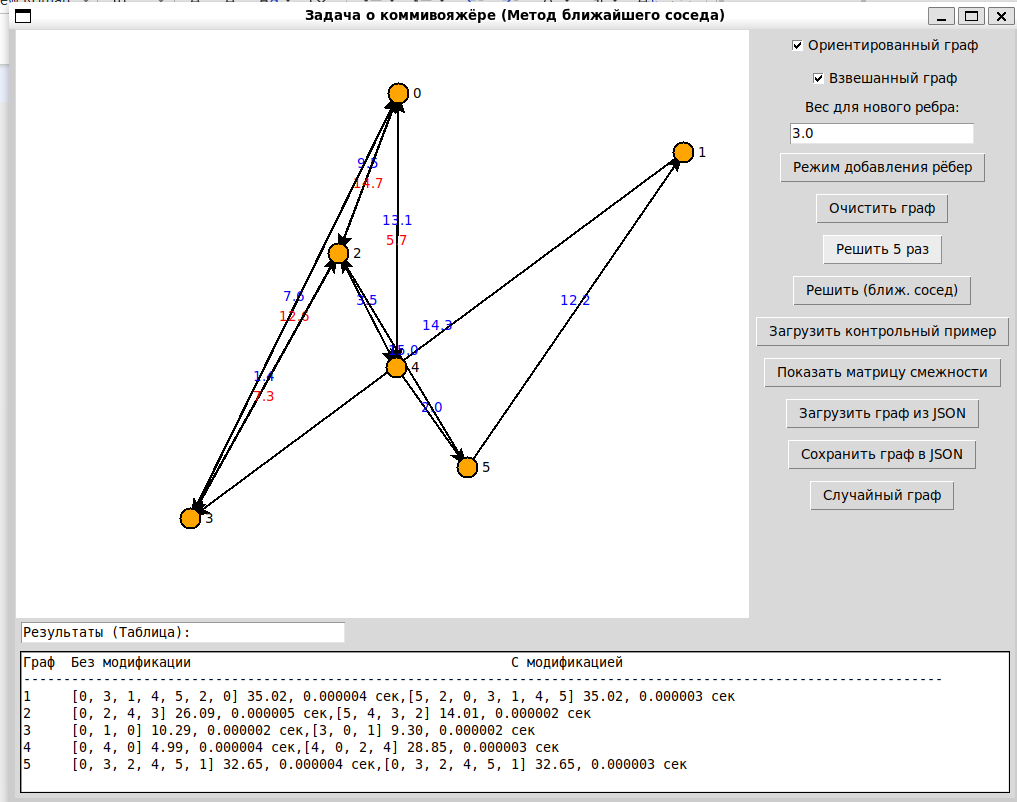


Рис 2 Пример решения графа

# Вывод

* Метод ближайшего соседа даёт быстрое приблизительное решение, но не всегда оптимальное.
* Реализация с интерактивным интерфейсом позволяет легко строить различные тестовые графы, проверять работу алгоритма и анализировать результаты.
* Для более точных решений или больших графов можно использовать улучшенные эвристики или точные методы с оптимизацией.

ы, что открывает новые возможности для анализа безопасности.

# Источники

1. Tkinter библиотека: [https://tkinter.org/](https://pandas.pydata.org/)
2. Math библиотека: https://math.org/
3. Json модуль: [https://docs.python.org/3/library/json.html](https://docs.python.org/3/library/csv.html)
4. Random модуль: [https://docs.python.org/3/library/random.html](https://docs.python.org/3/library/concurrent.futures.html)