**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №8**

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

**на тему «Решение задачи о коммивояжере с помощью муравьиного алгоритма»**

**Студент гр. 23Б16-пу**

**Пушкарев Н.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#__RefHeading___Toc8220_1245817469)
2. [Описание задачи 3](#__RefHeading___Toc9343_1245817469)
3. [Теоретическая часть 4](#__RefHeading___Toc8222_1245817469)
4. [Основные шаги программы 5](#__RefHeading___Toc2061_3961991740)
5. [Блок схема программы 6](#__RefHeading___Toc2063_3961991740)
6. [Описание программы 8](#__RefHeading___Toc2065_3961991740)
7. [Рекомендации пользователя 11](#__RefHeading___Toc2067_3961991740)
8. [Рекомендации программиста 12](#__RefHeading___Toc2069_3961991740)
9. [Исходный код программы 12](#__RefHeading___Toc2071_3961991740)
10. [Контрольный пример 13](#__RefHeading___Toc2073_3961991740)
11. [Анализ работы муравьиного алгоритма с «элитными» муравьями 15](#__RefHeading___Toc2075_3961991740)
12. [Вывод 18](#__RefHeading___Toc2077_3961991740)
13. [Источники 19](#__RefHeading___Toc2079_3961991740)

# Цель работы

Цель данной лабораторной работы заключается в изучении муравьиного алгоритма для решения задачи коммивояжера и оценке эффективности его модифицированной версии с использованием "элитных" муравьев. В рамках работы планируется разработать программную реализацию муравьиного алгоритма, включая функцию визуализации найденного маршрута.

# Описание задачи

1. **Изучение задачи коммивояжёра и муравьиного алгоритма**  
    Анализ алгоритма Ant Colony Optimization (ACO) и его модификаций, включая метод "элитных" муравьев, который усиливает влияние лучших найденных решений на процесс поиска. Исследование влияния параметров алгоритма на сходимость и качество решения.
2. **Разработка программы для поиска кратчайшего гамильтонова цикла**  
    Реализация базового муравьиного алгоритма и его модификации с "элитными" муравьями:

* Обновление феромонов с учетом только лучшего маршрута ("элитные" муравьи)
* Комбинирование с локальными методами оптимизации (2-opt)

1. **Тестирование программы на взвешенных графах**  
    Проверка корректности работы алгоритма на контрольных примерах. Сравнение длины найденного пути с известным оптимумом или результатами других методов (например, метода ветвей и границ).
2. **Анализ результатов**  
    Оценка точности и скорости работы алгоритма в стандартной и модифицированной версиях. Определение оптимальных параметров (коэффициент испарения феромонов, влияние "элитных" муравьев, количество итераций).

# Теоретическая часть

**Муравьиный алгоритм** Муравьиный алгоритм (Ant Colony Optimization) — это метаэвристический метод оптимизации, вдохновленный поведением муравьев в природе. В данной работе используется модификация алгоритма с применением "элитных" муравьев, которая усиливает влияние лучших решений на процесс обновления феромонов, что способствует более быстрому нахождению оптимального маршрута.

**Параметры алгоритма и их роль**

1. **Количество муравьев**:  
   Определяет число потенциальных решений, исследуемых на каждой итерации.
2. **Коэффициент испарения феромонов**:  
   Контролирует скорость уменьшения концентрации феромонов на ребрах графа.
3. **Влияние "элитных" муравьев**:  
   Усиливает отложение феромонов на ребрах лучшего найденного маршрута.
4. **Критерий остановки**:  
   Определяет условия завершения работы алгоритма (достижение максимального числа итераций или стабилизация решения).

**Принцип работы**

1. Инициализация феромонов на всех ребрах графа.
2. На каждой итерации каждый муравей строит маршрут, выбирая ребра на основе концентрации феромонов и эвристической информации.
3. Обновление феромонов с учетом всех маршрутов, с усиленным влиянием "элитного" маршрута.
4. Повторение процесса до выполнения критерия остановки.

Муравьиный алгоритм с модификацией "элитных" муравьев демонстрирует улучшенную сходимость и качество решений, однако требует тщательного подбора параметров для достижения наилучших результатов.

# Основные шаги программы

1. **Запуск программы:**
2. **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.
3. **Поиск решения:**
   * Генерация начального маршрута.
   * Итеративное улучшение маршрута методом муравьиного алгоритма.
   * Обновление феромонов.
   * Испарение феромонов.
   * Повторение процесса.
4. **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»
5. **Дополнительные возможности:**
   * Возможность удаления ребер через таблицу.
   * Возможность загрузить граф в формате json и не рисовать его вручную.

# Блок схема программы

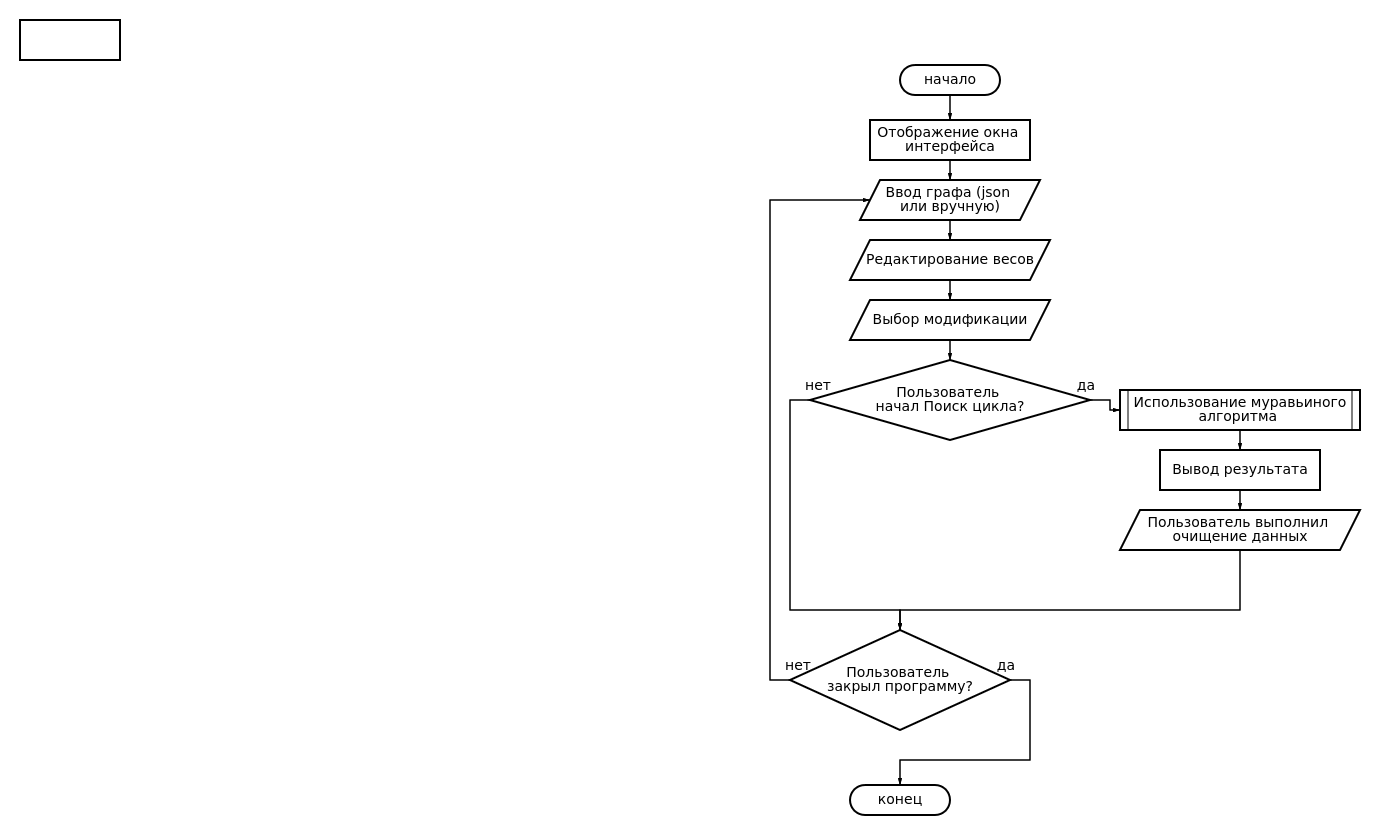
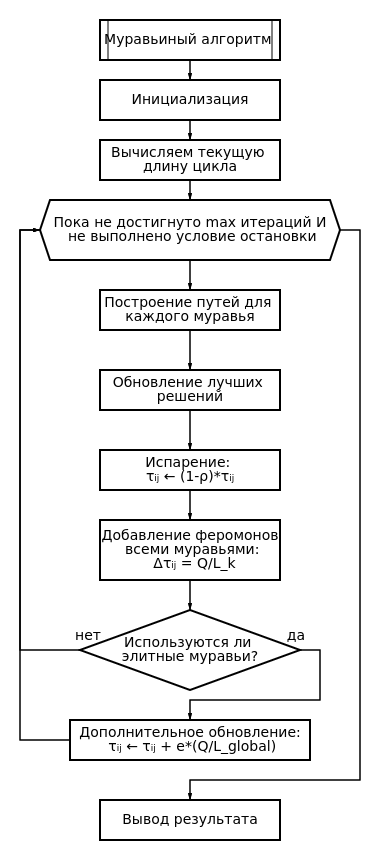


Рис 1. Блок-схема программы

Рис. Блок-схема алгоритма

# 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\_with\_ant\_colony | Основной метод муравьиного алгоритма с возможностью использования элитных муравьев | use\_elitism - флаг использования элитных муравьев (bool), параметры из GUI: количество муравьев, итераций, α, β, ρ, Q | Кортеж (best\_path, best\_length) - лучший найденный путь и его длина, либо None если решение не найдено |
| \_construct\_ant\_path | Строит путь для одного муравья | pheromone - словарь феромонов, alpha - влияние феромонов, beta - влияние эвристики | path - построенный путь (list) или None если путь неполный |
| \_select\_next\_node | Выбирает следующую вершину для муравья по вероятностному правилу | current - текущая вершина, unvisited - непосещенные вершины, pheromone - феромоны, alpha, beta - параметры | node - выбранная следующая вершина |
| \_calculate\_path\_length | Вычисляет длину гамильтонова цикла | path - список вершин пути | length - длина цикла (float) или inf если путь недопустим |
| \_has\_possible\_path | Проверяет существование хотя бы одного возможного пути между вершинами | - | bool - True если граф связный, False если есть изолированные вершины |
| \_update\_pheromones | Обновляет феромоны на ребрах | pheromone - текущие феромоны, paths - найденные пути, rho - коэффициент испарения, q - количество феромона, weight - вес обновления | - |
| 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».

* + Настройка параметров:

Настройте параметры муравьиного алгоритма.

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.
   * В поле **«Время выполнения»** - 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\_8

# 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).

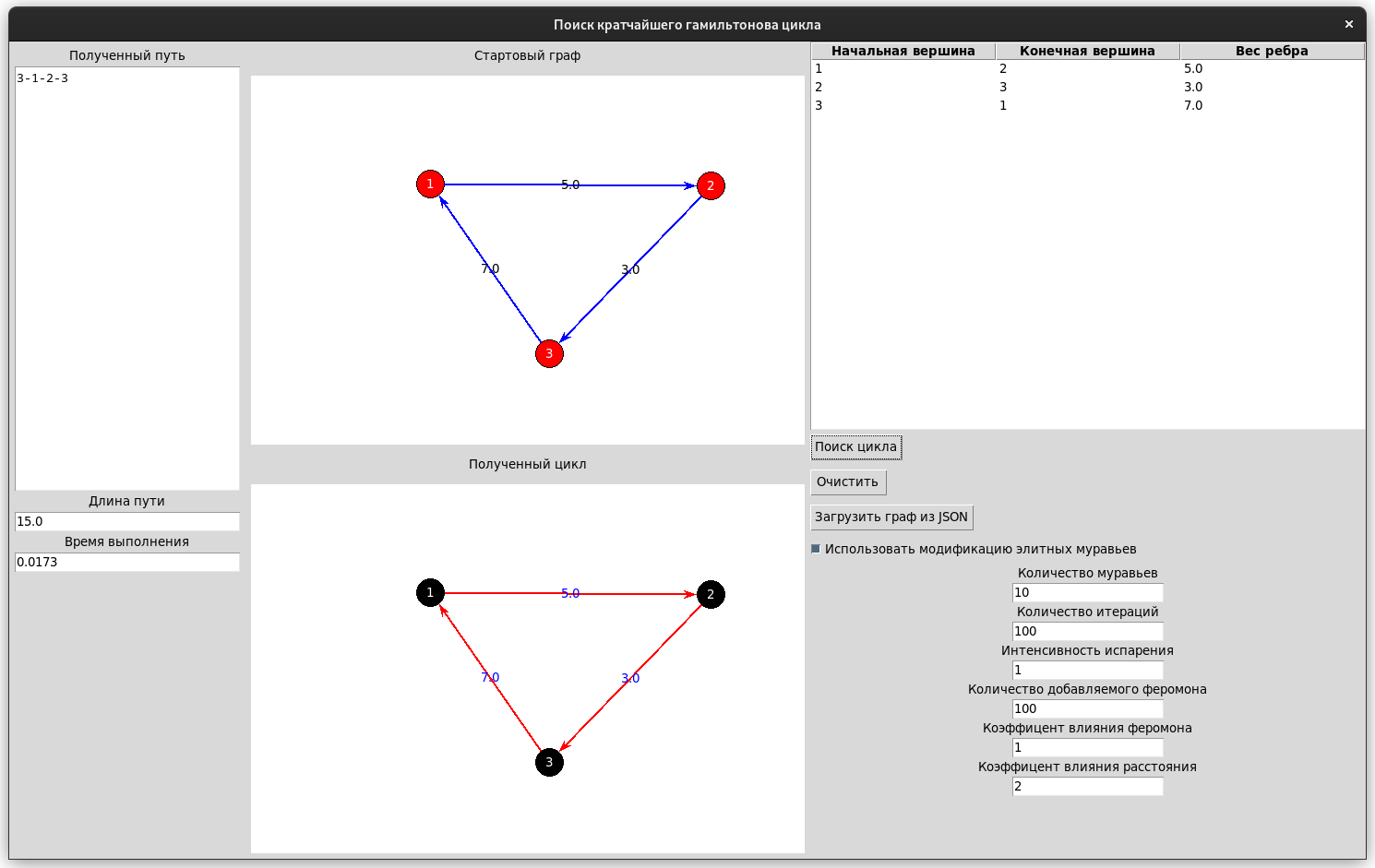


Рис 2. Пример окна программы

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

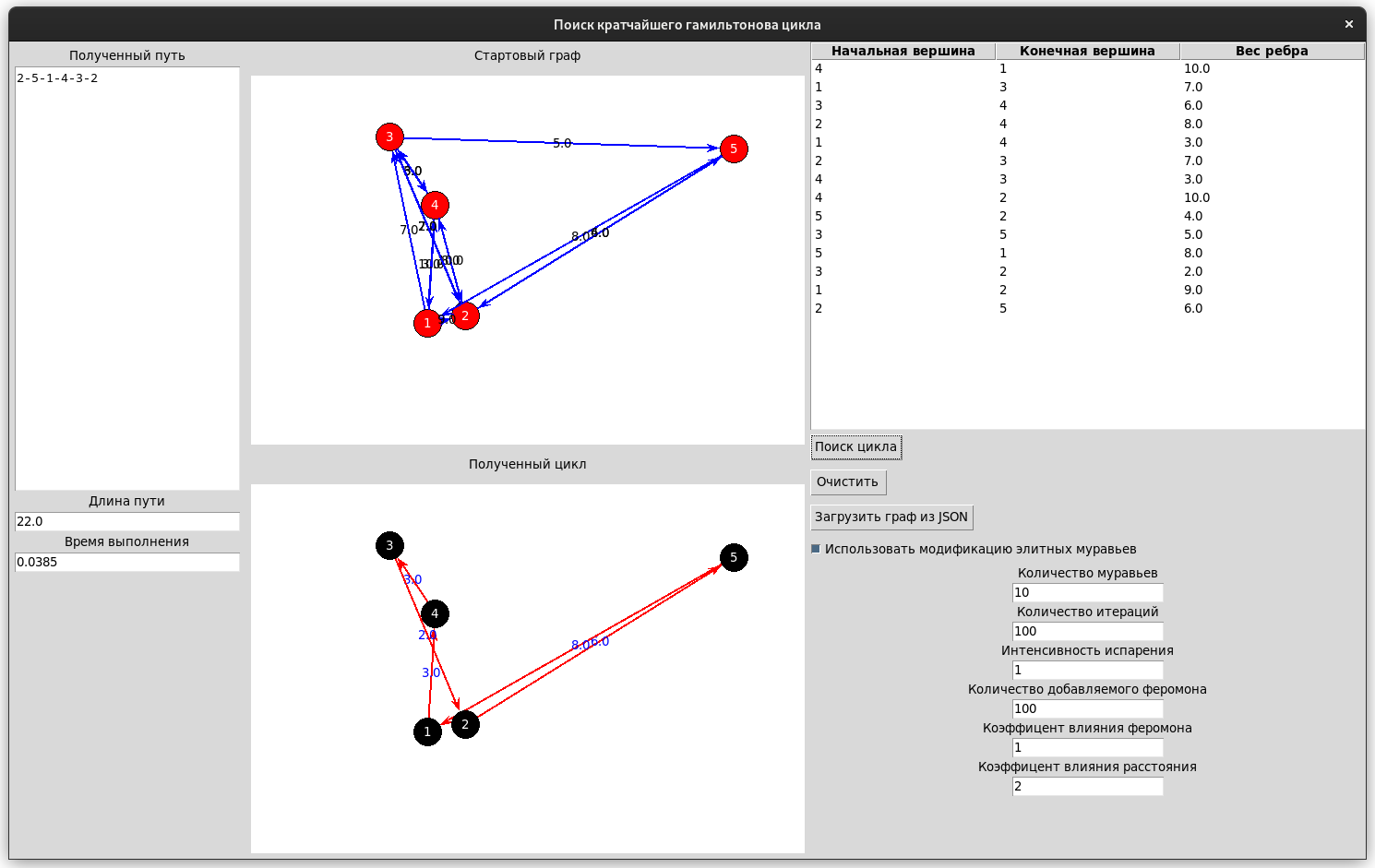


Рис 3. Пример результатов программы

Просмотр результатов

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

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

# Анализ работы муравьиного алгоритма с «элитными» муравьями

Анализ работы муравьиного алгоритма показывает, что он обладает хорошей скоростью выполнения и большой надежностью. На рис. 3 представлено, что метод успешно нашел минимальный путь в графе из 5 вершин при использовании модификации.

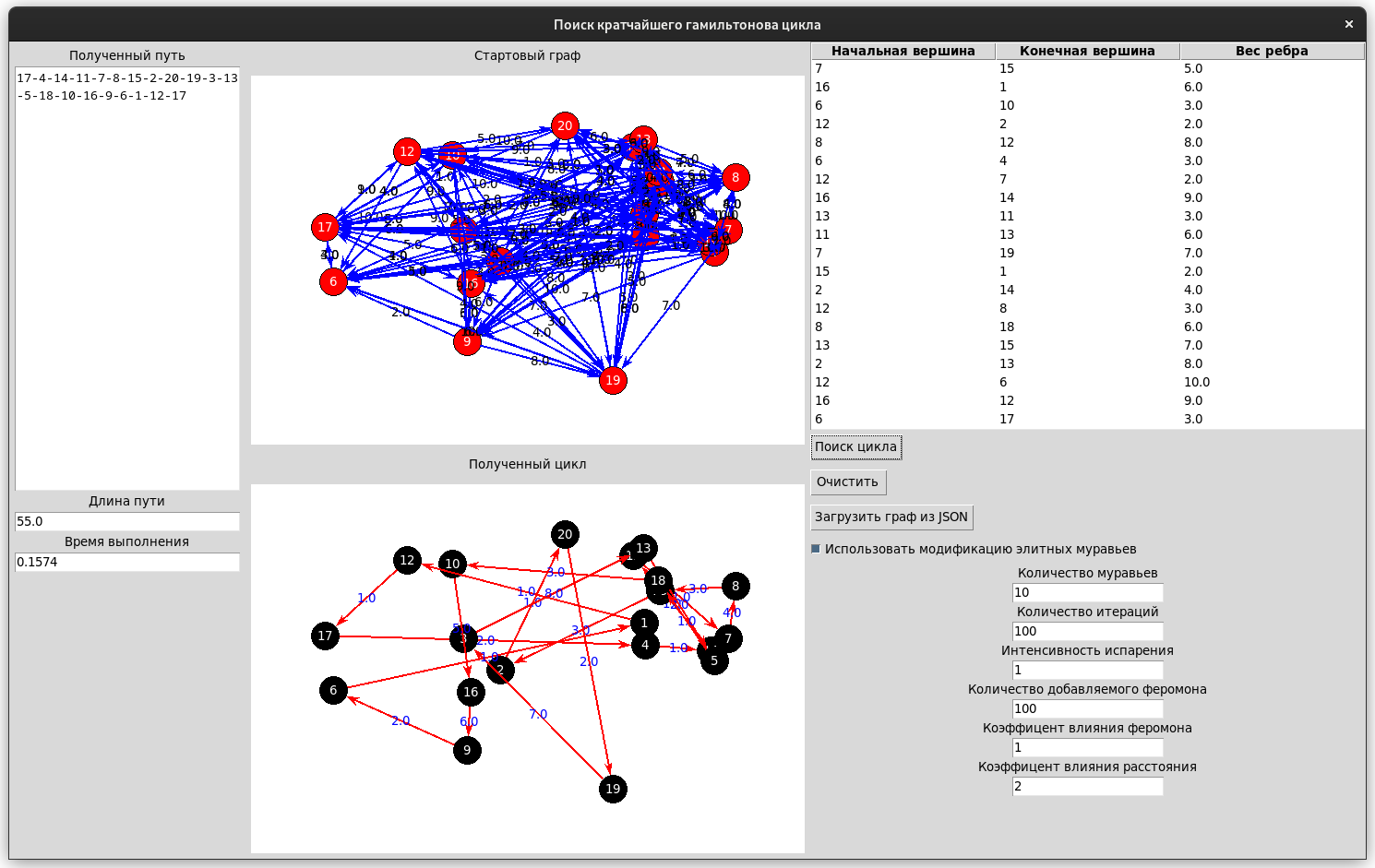


Рис 4. Пример результатов программы

На рис. 4 видно, что алгоритм смог найти путь в примере на 20 вершин с большим количеством ребер.

**Сравнительный анализ с алгоритмом ближайшего соседа**

Как показал анализ алгоритма ближайшего соседа (NN), его основными преимуществами являются:

* Высокая скорость выполнения (3-5 мс для графов из 20 вершин)
* Простота реализации
* Детерминированность решений

Однако эти преимущества достигаются ценой:

* Низкой точности решений (в среднем на 16-20% хуже оптимальных)
* Высокой вероятности (40%) попадания в тупиковые ситуации
* Сильной зависимости от выбора начальной вершины

**Результаты тестирования алгоритма имитации отжига с модификацией Коши**

Для объективного сравнения был проведен аналогичный тест на тех же графах из 20 вершин:

Таблица 2. Результаты муравьиного алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № **графа** | **Длина пути (Муравьиный)** | **Время (Муравьиный), мс** | | **Длина пути (Муравьиный + элита)** | **Время (Муравьиный + элита), мс** |
| 1 | 106 | 420 | | 103 | 580 |
| 2 | 115 | | 410 | 111 | 560 |
| 3 | 120 | 450 | | 117 | 620 |
| 4 | 112 | | 430 | 109 | 590 |
| 5 | 102 | | 400 | 98 | 550 |

Таблица 3. Результаты алгоритма ближайшего соседа

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

Таблица 4. Результаты алгоритма имитации отжига

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № **графа** | **Длина пути (SA)** | **Время (SA), мс** | | **Длина пути (SA-Коши)** | **Время (SA-Коши), мс** |
| 1 | 116 | 180 | | 112 | 245 |
| 2 | 102.3 | | 122 | 124 | 119 |
| 3 | 127 | 190 | | 125 | 263 |
| 4 | 99.2 | | 119 | 121 | 117 |
| 5 | 97.8 | | 113 | 115 | 108 |

Таблица 5. Сравнение средних значений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Алгоритм** | **Улучшение против NN** | | **Улучшение против MS-NN** | **Время работы** |
| NN | — | | — | ~3 мс |
| MS-NN | | ~16-17% | — | ~100 мс |
| SA (обычный) | ~18% | | ~1.5-2% | ~180 мс |
| SA (Коши) | | ~21-22% | ~5-7% | ~250 мс |
| Муравьиный | | ~24-25% | ~8-10% | ~400 мс |

**Ключевые наблюдения:**

* Если нужно **быстрое улучшение** — **MS-NN**.
* Если нужен **баланс скорости и качества** — **SA (обычный)**.
* Если важно **максимальное качество** — **SA-Коши** или **муравьиный алгоритм**.
* Обычный SA лучше, чем жадные алгоритмы (NN, MS-NN), но уступает SA-Коши и муравьиным методам.

# Вывод

В ходе работы был реализован **муравьиный алгоритм** для решения задачи коммивояжёра, включая его **модификацию с "элитными" муравьями**, а также разработан **графический интерфейс** для визуализации графа и результатов.

**Экспериментальные результаты** показали, что:

1. **Муравьиный алгоритм** демонстрирует **более высокую точность** по сравнению с методом ближайшего соседа (NN) и его модификацией (MS-NN), а также **превосходит алгоритм имитации отжига (SA)** по качеству решения.
2. **Скорость работы** муравьиного алгоритма **приемлема**, особенно в сравнении с SA при схожем уровне точности.
3. **Модификация с "элитными" муравьями** ускоряет сходимость, слегка снижая точность, но сохраняя преимущество перед жадными методами и SA.

# 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. Муравьиный алгоритм // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Муравьиный\_алгоритм (дата обращения: (04.03.2025).