**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**Факультет прикладной математики – процессов управления**

**отчет**

**по лабораторной работе**

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

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 23.Б16 |  | Барафанов А.А. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2025 г.**

**Оглавление**

[**1.** **Цель работы** 3](#_Toc195294793)

[**2.** **Задачи (формализация)** 3](#_Toc195294794)

[**3.** **Теоретическая часть** 3](#_Toc195294795)

[**4.** **Описание алгоритма** 5](#_Toc195294796)

[**5.** **Описание программы** 10](#_Toc195294797)

[**5.1 Описание функций** 10](#_Toc195294798)

[**6.** **Рекомендации пользователю** 11](#_Toc195294799)

[**7.** **Рекомендации программисту** 12](#_Toc195294800)

[**8.** **Контрольный пример** 13](#_Toc195294801)

[**9.** **Анализ результатов** 16](#_Toc195294802)

[**10.** **Вывод** 18](#_Toc195294803)

[**11.** **Листинг программы** 18](#_Toc195294805)

# **Цель работы**

Исследовать эффективность муравьиного алгоритма для решения задачи коммивояжёра (TSP), изучить влияние параметров (интенсивность феромона, коэффициент привлекательности маршрута, скорость испарения феромона) на точность решения и сравнить результаты разных семейств алгоритмов (жадные, стохастические, мультиагентные) для решения задачи коммивояжёра.

# **Задачи (формализация)**

* *Реализовать муравьиный алгоритм* (классический и модификация «Шаблоны»).
* *Разработать графический интерфейс*, позволяющий: вводить и визуализировать граф и настраивать параметры алгоритма.
* *Сравнить эффективность классического муравьиного алгоритма и с шаблонами* (по времени выполнения и длине пути). Сопоставить результаты с алгоритмами ближайшего соседа и имитации отжига.

# **Теоретическая часть**

* *Задача коммивояжёра:*

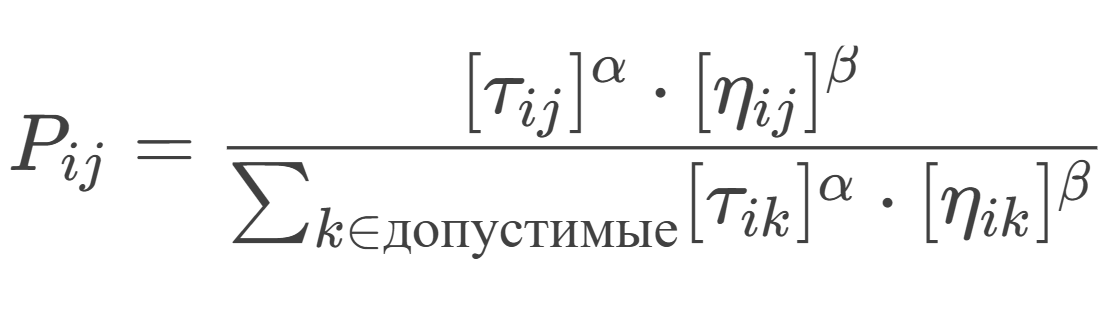
Задача коммивояжёра (TSP — Traveling Salesman Problem) заключается в поиске кратчайшего замкнутого маршрута, проходящего через все заданные города ровно один раз и возвращающегося в исходную точку. Эта задача является классической NP-сложной проблемой комбинаторной оптимизации, что делает её удобной для тестирования метаэвристических алгоритмов, включая муравьиные системы.

* *Муравьиный алгоритм (Ant Colony Optimization, ACO):*

*Муравьиный алгоритм* — это биоинспирированный метод оптимизации, основанный на поведении муравьёв, которые оставляют феромоны на пути к пище. Основные принципы:

* *Феромонная модель:* Каждый муравей строит маршрут, выбирая следующий город с вероятностью, зависящей от уровня феромона и эвристической информации (например, обратного расстояния).
* *Обновление феромонов:* После построения всех маршрутов феромоны обновляются:
* Испарение: уменьшение интенсивности феромонов для избегания застоя.
* Большее добавление феромонов на рёбра лучших маршрутов.
* *Баланс исследования и эксплуатации:* Параметры (α, β) регулируют влияние феромонов и эвристики на выбор пути.

*Вероятность перехода из города i в город j:*



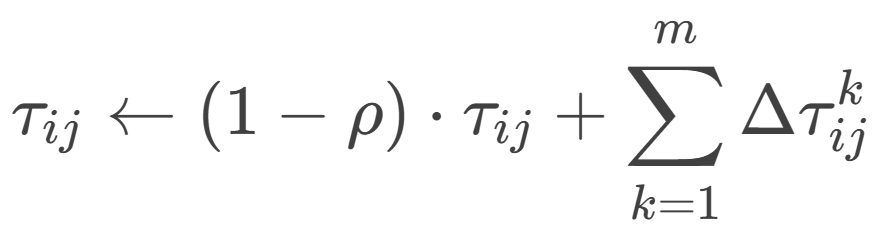
где:

— уровень феромона на ребре (i, j),

— эвристическая информация ( обратное расстояние),

— параметры, определяющие вес феромона и расстояния.

*Обновление феромонов:*



где:

— коэффициент испарения,

— количество феромона, отложенного муравьём k (обычно обратно пропорционально длине маршрута).

* *Модификация "Шаблоны":*

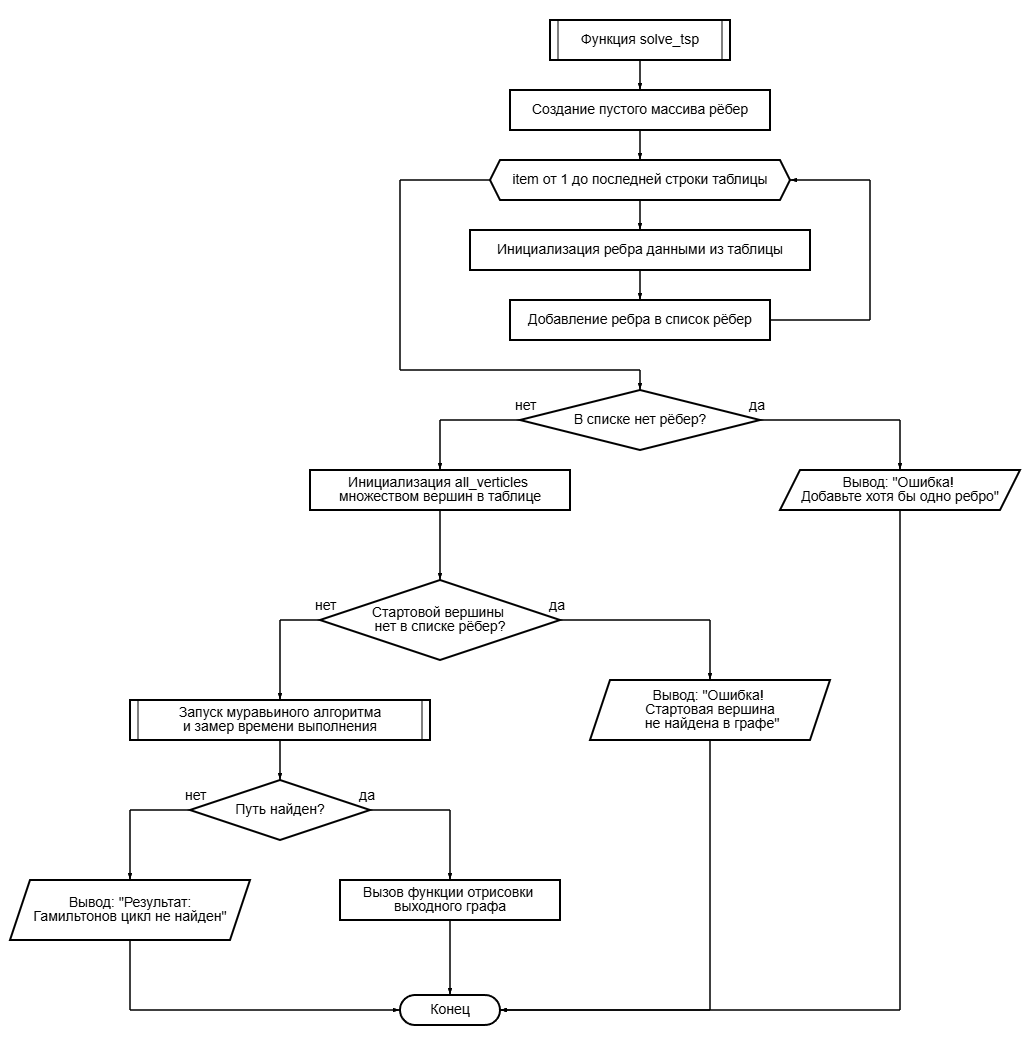
Модификация "Шаблоны" (Templates) направлена на ускорение сходимости алгоритма за счёт использования заранее определённых паттернов (шаблонов) построения маршрутов. Основные идеи:

* *Предопределённые структуры:* Шаблоны — это частичные решения или часто встречающиеся подмаршруты (например, ближайший сосед, оптимальные отрезки из локального поиска).
* *Гибридный подход:* Муравьи комбинируют случайное построение маршрута с вставкой шаблонов, что уменьшает время поиска.
* *Адаптация шаблонов:* Шаблоны могут динамически обновляться на основе лучших найденных решений.

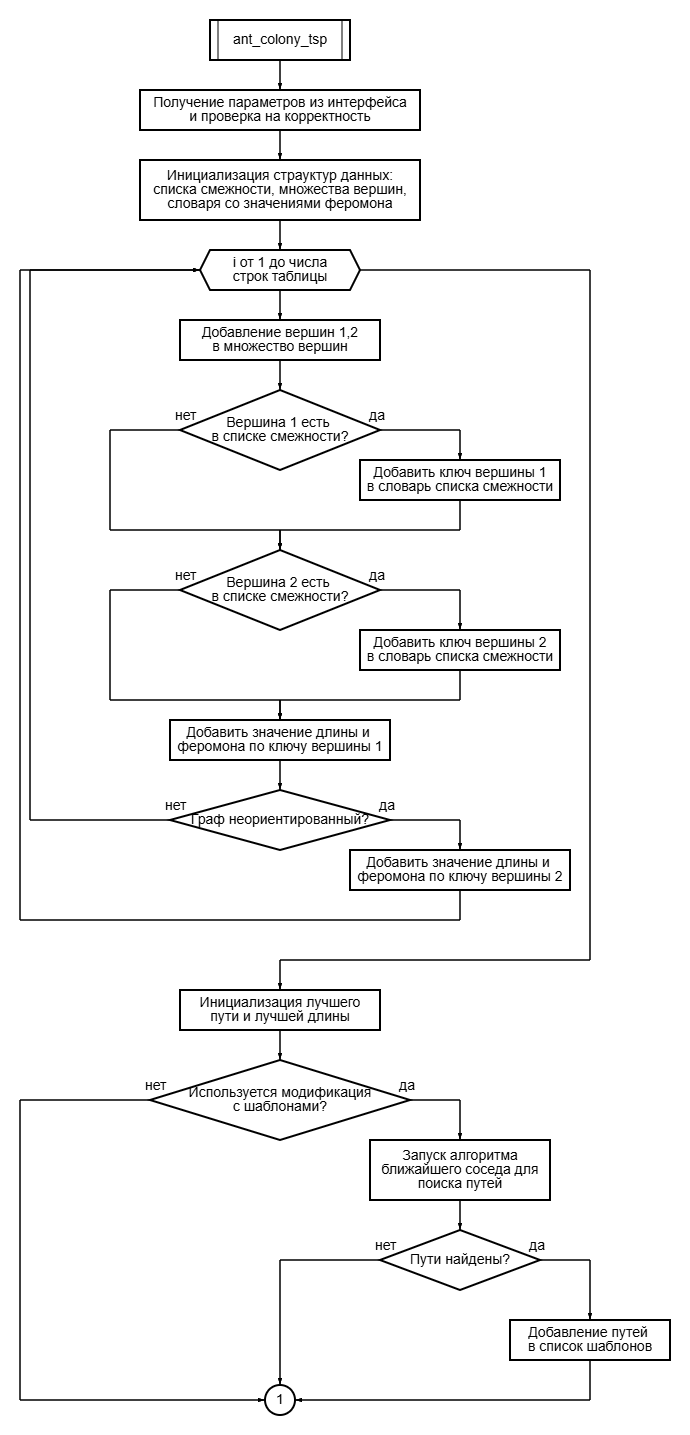
# **Описание алгоритма**

1. *Создание GUI (tkinter)*  
   Инициализация главного окна с тремя основными областями: панель параметров, область визуализации графов и таблица ребер.
2. *Обработка событий мыши (on\_click)*  
   При клике создается вершина (красный круг с меткой), а при последующем клике на другую вершину открывается диалог simpledialog для ввода веса ребра (которое затем добавляется в edges[] и Treeview) и феромона.
3. *Визуализация графа (draw\_input\_graph)*Отрисовываются овалы вершин, линии ребер (create\_line) с подписями весов (create\_text), обновляя canvas.delete("all") при каждом изменении.
4. *Запуск муравьиного алгоритма (ant\_colony\_tsp)*Основная функция ant\_colony\_tsp(). Инициализируются параметры (α, β, испарение). Производит построение маршрутов муравьями и обновление феромонов.
5. *Муравьиный алгоритм с шаблонами*Блок if use\_templates: в ant\_colony\_tsp(). Генерирует шаблоны через nearest\_neighbor\_tsp(). Затем добавляет шаблоны к путям муравьев (ants\_paths.extend(templates)).
6. *Управление данными (tree/edges)*Treeview отображает ребра в формате (v1, v2, weight), синхронизируясь с глобальным списком edges[], при этом pandas.DataFrame используется для внутренних расчетов.
7. *Запуск расчета (solve\_tsp)*Функции solve\_tsp() и solve\_tsp\_with\_templates(). Производят преобразование данных из Treeview в DataFrame, вызывают алгоритм и замер времени, а также визуализируют результатов (вывод пути и отрисовка графа)
8. *Отображение результата (draw\_output\_graph)*Отображаем финальный путь зелеными линиями (width=2), сохраняя исходную визуализацию вершин, с проверкой path[] на корректность.

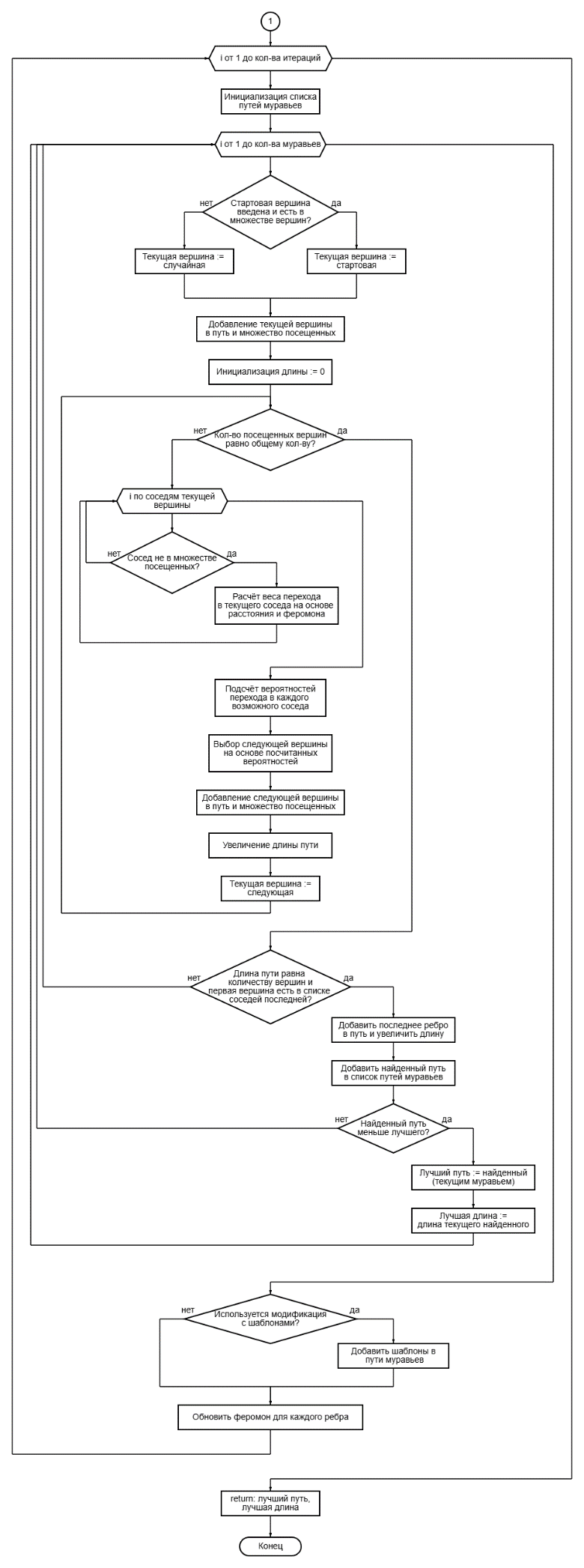
*Блок-схемы, иллюстрирующие шаги алгоритма показаны на рисунках 4.1-4.3*

**

*Рисунок 4.1. Блок-схема основной функции solve\_tsp*

**

*Рисунок 4.2. Блок-схема муравьиного алгоритма. Часть 1.*

**

*Рисунок 4.3. Блок-схема муравьиного алгоритма. Часть 2.*

# **Описание программы**

## **5.1 Описание функций**

В программе используется 11 функций, выполняющие вспомогательные процедуры программы. В таблице 5.1.1 представлено их описание.

*Таблица 5.1.1. Описание функций*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| nearest\_neighbor\_tsp | graph (DataFrame), start\_vertex (str) | Генерирует начальное решение TSP жадным алгоритмом (ближайший сосед). Возвращает путь и его длину или (None, None), если цикл не найден. |
| ant\_colony\_tsp | graph (DataFrame), start\_vertex (str), use\_templates (bool, default=False) | Реализует муравьиный алгоритм (МА) с обновлением феромонов. Если use\_templates=True, использует шаблонные пути для ускорения сходимости. Возвращает лучший найденный путь и его длину. |
| add\_edge\_to\_table | v1 (str), v2 (str), length (float), pheromone (float, default=0.1) | Добавляет ребро в Treeview таблицу с указанными вершинами, длиной и феромоном. |
| on\_click | event (tkinter.Event) | Обрабатывает клики на холсте: создаёт вершины, запрашивает длину и феромон для рёбер, вызывает отрисовку графа. |
| draw\_input\_graph | — | Отрисовывает входной граф (вершины и рёбра) на input\_canvas с подписями длин и феромонов. |
| draw\_output\_graph | path (list) | Визуализирует найденный путь зелёными линиями на output\_canvas. Если путь пуст, очищает холст. |
| solve\_tsp | — | Запускает муравьиный алгоритм без шаблонов: проверяет входные данные, вызывает ant\_colony\_tsp, выводит результат в интерфейс и на график. |
| solve\_tsp\_with\_templates | — | Аналогично solve\_tsp, но с использованием шаблонных путей (use\_templates=True). |
| clear\_canvas | — | Сбрасывает состояние программы: очищает холсты, таблицу рёбер и результаты. |
| set\_graph\_type | type (str: "directed" или "undirected") | Устанавливает тип графа (ориентированный/неориентированный) и обновляет логику работы рёбер. |
| load\_graph\_from\_file | — | Загружает граф из CSV-файла (колонки: "вершина 1", "вершина 2", "длина", "феромон"). Автоматически размещает вершины на холсте. |

# 

# **Рекомендации пользователю**

*Запуск программы:*

* Убедитесь, что у вас установлен Python 3.7 или новее и проверьте наличие необходимых библиотек

*Настройка графа:*

* Кликните левой кнопкой мыши на верхнем белом холсте, чтобы создать вершину (вершины автоматически нумеруются как V1, V2, V3 и т.д.).
* Или же нажмите кнопку "Загрузить граф из файла" (выберите CSV-файл с колонками: "вершина 1", "вершина 2", "длина", "феромон").

*Ввод параметров:*

* В поле "Стартовая вершина" введите название вершины (например, V1)
* Установите параметры алгоритма:
* α (вес феромона): 0.5–2.0, оптимально 1.0
* β (вес расстояния): 1.0–5.0, оптимально 2.0–3.0
* ρ (испарение феромона): 0.1–0.5, оптимально 0.3–0.5
* Количество муравьев: 10–50, оптимально ≈ числу вершин
* Количество итераций: 50–500, оптимально 100–200
* Q (количество феромона): 50–200, оптимально 100
* Начальный феромон (τ₀): 0.01–0.1, оптимально 0.1

*Выбор алгоритма:*

* "Муравьиный алгоритм" - классический муравьиный алгоритм
* "С шаблонами" - шаблонная версия муравьиного алгоритма (шаблоны создаются на основе решений жадного алгоритма)

*Если решение не найдено,* появится предупреждение - проверьте:

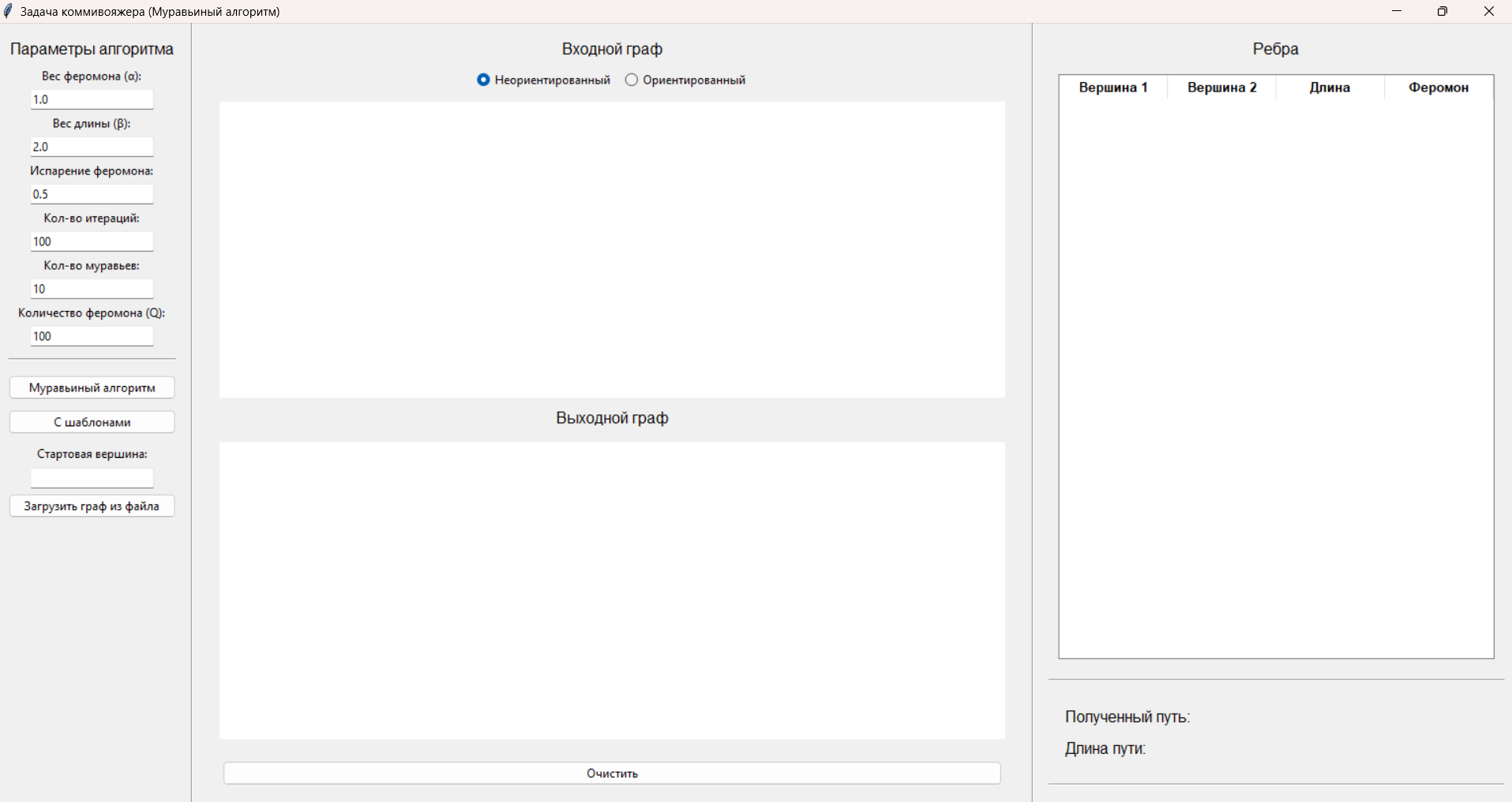
* Все ли вершины связаны;
* Правильно ли указана стартовая вершина;
* И Корректность введенных данных.

1. **Рекомендации программисту**

* Для работы с программой убедитесь, что установлены все необходимые зависимости: tkinter, pandas и random.
* Запускайте программу через командную строку или IDE (например, PyCharm, VSCode) для удобства отладки и отслеживания ошибок.
* Программа реализует муравьиный алгоритм для задачи коммивояжера, а также его шаблонную версию. Для иной модификации алгоритм измените логику в функции solve\_tsp\_with\_templates.
* При работе с графом вершины добавляются кликом на холст, а рёбра создаются между двумя вершинами с указанием длины. Если нужно программно добавить вершины и рёбра, модифицируйте глобальные переменные vertices и edges и добавьте соответствующие функции.
* Имейте в виду, что шаблонный МА сходится быстрее классического МА за счёт использования шаблонов, но увеличивает риск застревания в локальном минимуме.
* Вы можете изменить логику создания шаблонов (например, использовать не жадные алгоритмы, а предыдущие запуски муравьиного), а также стратегию включения шаблонов в МА (например, вероятностный выбор для муравья между ребром из шаблона и оптимальным ребром классического алгоритма на данной итерации).
* Время выполнения: Для больших графов (>30 вершин) время работы может значительно увеличиться, поэтому рекомендуется начать тестирование с малых графов и постепенно увеличивать размер.

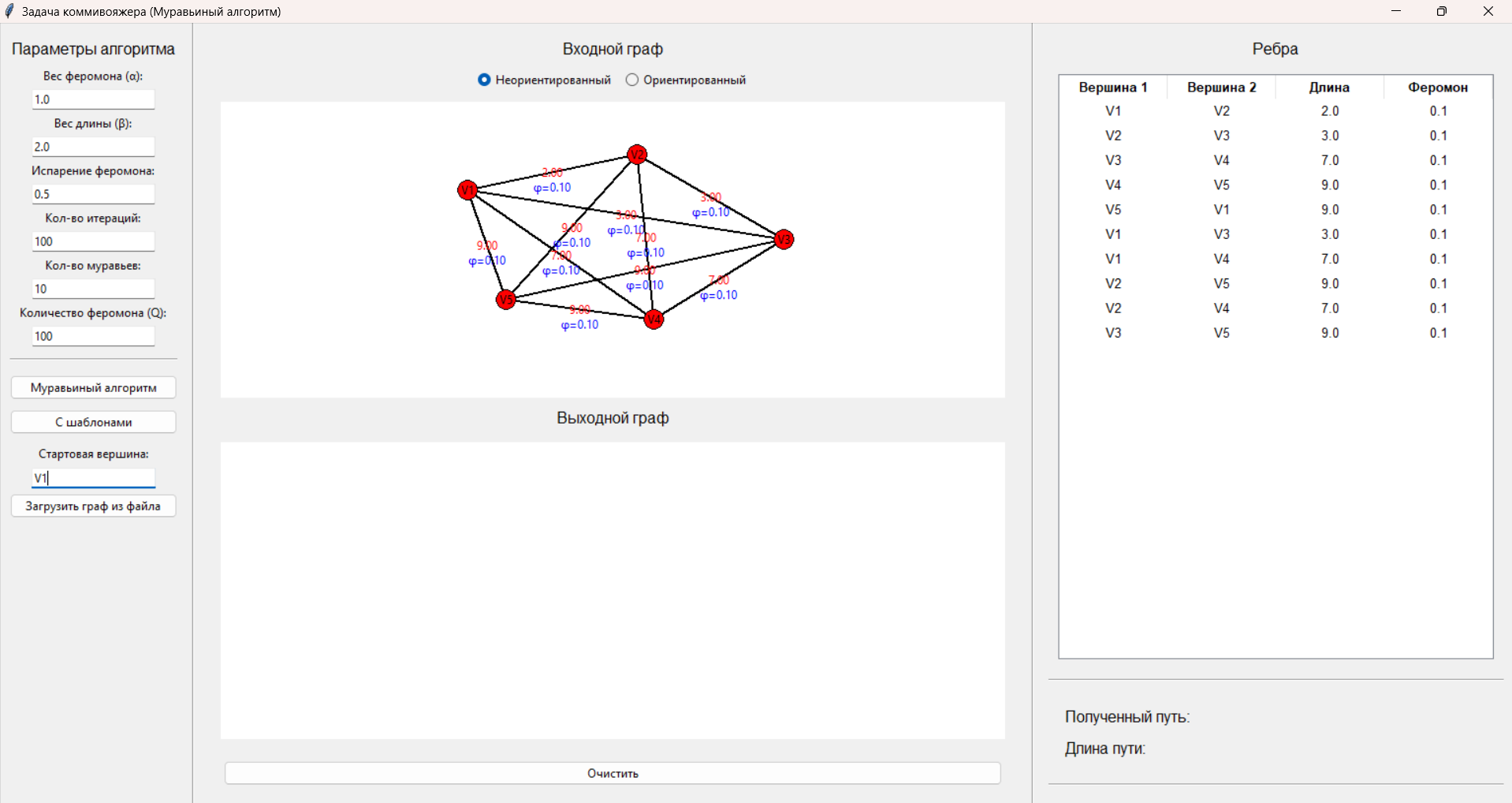
1. **Контрольный пример**

В этом разделе продемонстрирован пример запуска муравьиного алгоритма на примере заданного неориентированного графа. Тонкости взаимодействия с программой указаны в рекомендациях пользователю, здесь же приведена демонстрация работы с программой.



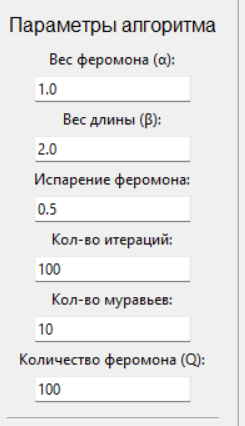
*Рисунок 8.1 Начальный интерфейс программы*

Для ввода вершин графа достаточно кликнуть мышью в нужное место верхнего холста, а для ввода ребра нужно последовательно кликнуть на две уже существующие вершины и ввести длину и феромон (опционально) в появившемся диалоговом окне. На рисунке 8.2 показан пример введенного графа и соответствующая ему таблица ребер и значений феромона.

**

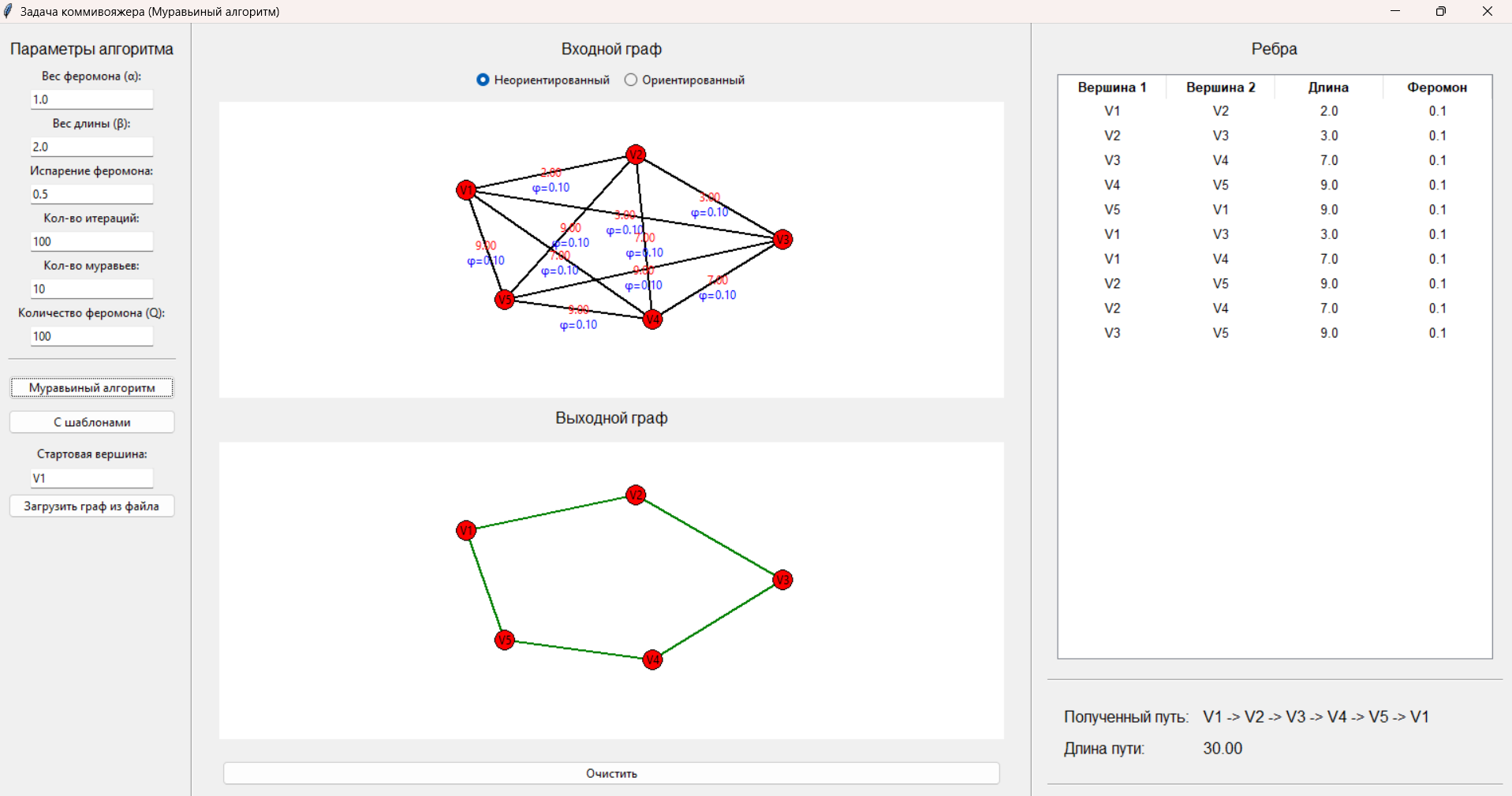
*Рисунок 8.2 Окно программы после ввода графа*

В верхней области левой части окна есть поля для ввода гиперпараметров, вы можете самостоятельно выставить нужные значения или оставить дефолтные (советы по эффективному выбору параметров даны в рекомендациях пользователю и программисту).



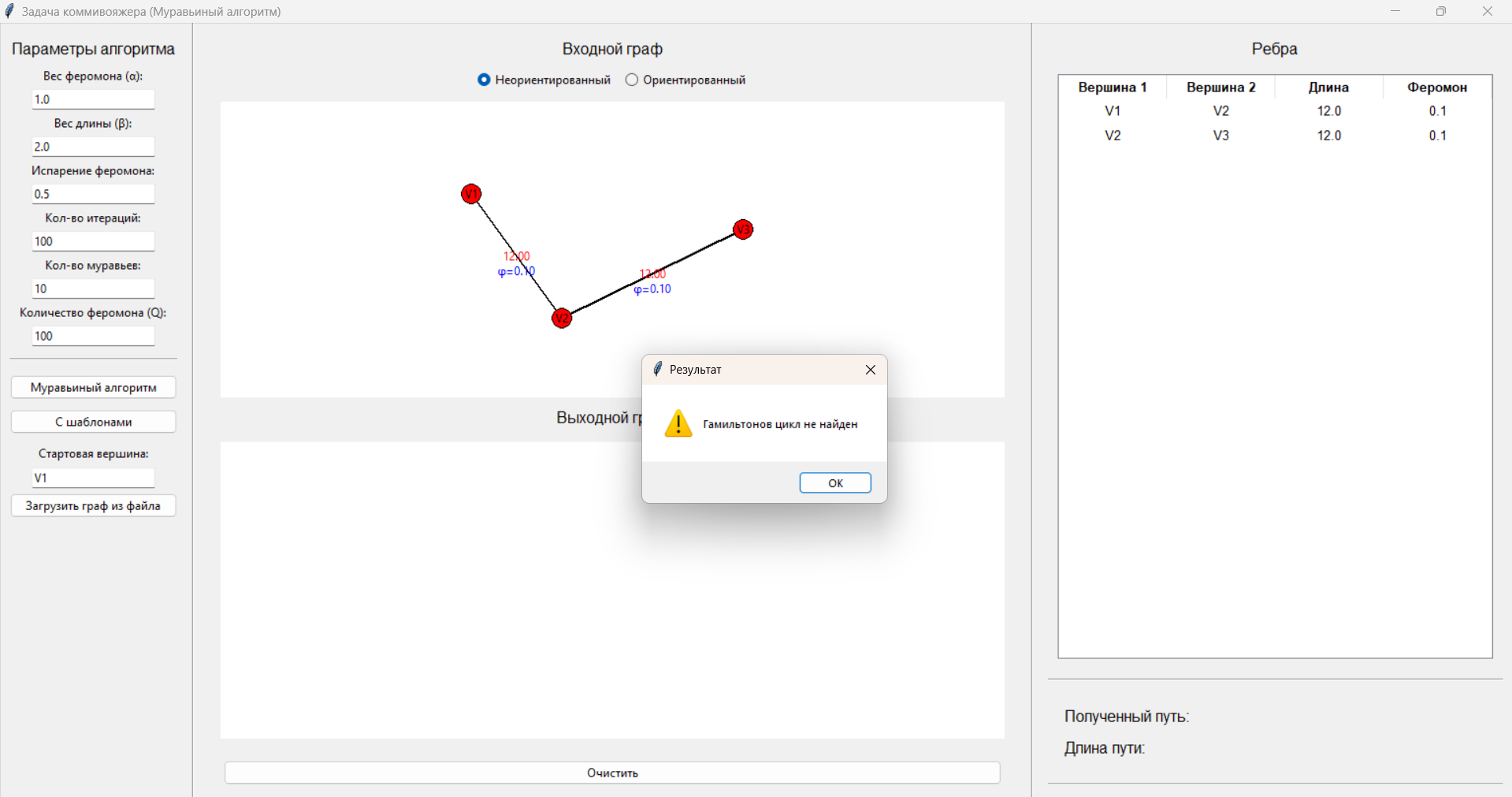
*Рисунок 8.3 Поля для ввода гиперпараметров*

Для получения результата необходимо после ввода графа выбрать стартовую вершину и нажать на кнопку «Муравьиный алгоритм» (в зависимости от ваших целей можете выбрать либо расчет по классическому алгоритму, либо по модифицированной версии). Результат появится в нижнем правом углу окна.

**

*Рисунок 8.4 Окно программы после нажатия кнопки «Муравьиный алгоритм»*

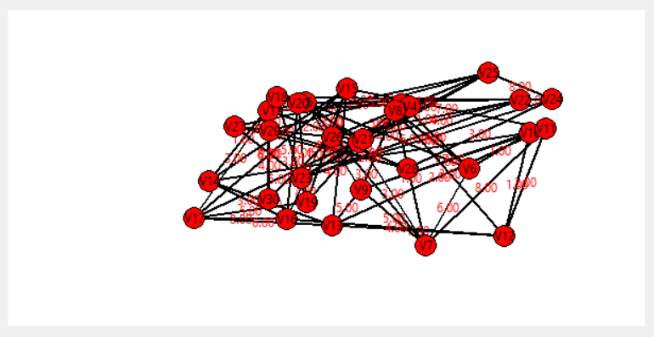
Если алгоритм не найдет гамильтонов цикл, то вы увидите предупреждение во всплывающем окне. В таком случае проверьте корректность введенных данных и выбранный тип графа.

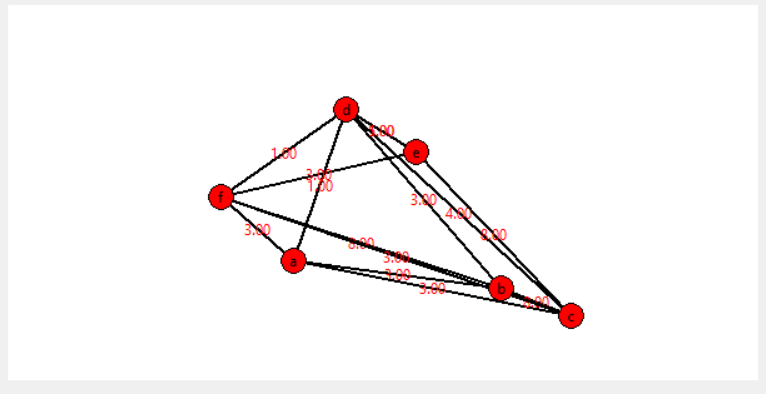
**

*Рисунок 8.5 Запуск с некорректными данными*

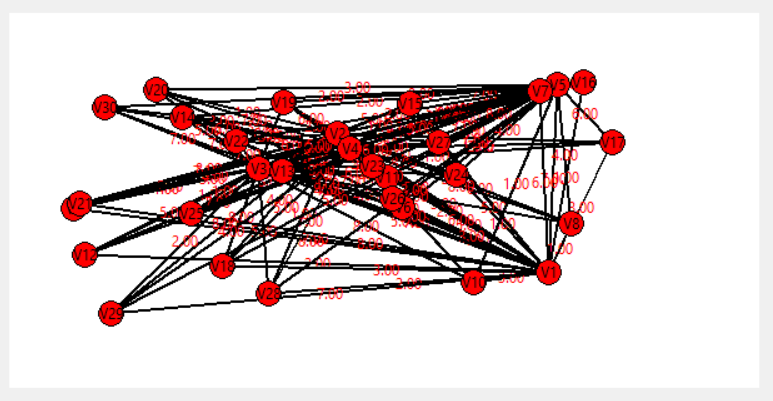
# **Анализ результатов**

Запустим алгоритм (классический и с модификацией) на трех тестовых графах (их картинки можно увидеть ниже). Каждый из них представляет собой граф разной сложности: с разным числом вершин и конфигурацией ребер.





*Рисунок 9.1. Входной граф №1 Рисунок 9.2. Входной граф №2*



*Рисунок 9.3. Входной граф №3*

Результаты работы после пяти запусков можно представить в виде следующей таблицы (в ней также присутствуют предыдущие результаты для алгоритмов ближайшего соседа и имитации отжига).

*Таблица 9.1. Анализ результатов*

|  |  |  |  |
| --- | --- | --- | --- |
| **Метрика** | **Граф 1** | **Граф 2** | **Граф 3** |
| Кол-во ребер | 14 | 52 | 103 |
| Лучшее решение (ближ. сосед) | 16 | 93 | 96 |
| Время (ближ. сосед) | 0.0010 сек | 0.0032 сек | 0.0065 сек |
| Лучшее решение (ближ. сосед  рандомизированный) | 16 | 93 | 97 |
| Время (ближ. сосед  рандомизированный) | 0.0010 сек | 0.0033 сек | 0.0063 сек |
| Лучшее решение (кл. отжиг) | 16 | 93 | 73 |
| Время (классич. отжиг) | 0.3906 сек | 0.5402 сек | 0.7065 сек |
| Лучшее решение (сверх. отжиг) | 16 | 93 | 87 |
| Время (сверхбыстрый отжиг) | 0.0095 сек | 0.0869 сек | 0.1050 сек |
| Лучшее решение  (муравьиный алгоритм) | 16 | 93 | 83 |
| Время (муравьиный алгоритм) | 0.0122 сек | 0.0393 сек | 0.0586 сек |
| Лучшее решение  (МА с шаблонами) | 16 | 93 | 77 |
| Время (МА с шаблонами) | 0.0140 сек | 0.0572 сек | 0.0922 сек |

1. Граф 1:  
   Все алгоритмы нашли оптимальное решение с длиной маршрута 16. Жадные алгоритмы (ближайший сосед и его рандомизированная версия) показали наилучшее быстродействие - около 0.001 секунды. Более сложные методы (отжиг и муравьиный алгоритм) не дали преимущества в качестве решения, а время их выполнения оказалось значительно выше.
2. Граф 2:  
   На графе среднего размера все методы, за исключением жадных, сошлись на решении длиной 93. Жадные алгоритмы так же продемонстрировали наилучшее быстродействие (около 0.003 сек), но не смогли найти оптимальный путь. Более сложные методы (отжиг и муравьиный с модификациями) показали сопоставимое качество решений при большем времени работы.
3. Граф 3:  
   На большом графе проявились существенные различия между алгоритмами. Классический отжиг показал наилучший результат (длина 73), однако потребовал наибольшего времени вычислений (0.7 сек). Сверхбыстрый отжиг (VFSA) и муравьиный алгоритм показали худшее качество (87 и 77 соответственно), но более быстрое время работы. Жадные методы дали наихудшие результаты (96-97), но при минимальном времени выполнения. Это показывает, что для больших графов классический отжиг является одним из лучших методов при наличии времени, а VFSA и муравьиные алгоритмы представляют компромисс между качеством и быстродействием (при небольшом числе итераций).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Жадные методы | Классический отжиг | Сверхбыстрый отжиг | Муравьиный алгоритм | МА с шаблонами |
| Скорость выполнения | Высокая | Низкая | Быстрее классического | Средняя | Немного медленнее МА |
| Качество на малых графах | Оптимальное решение | Оптимальное решение | Оптимальное решение | Оптимальное решение | Оптимальное решение |
| Качество на больших графах | Худшие результаты | Лучшие результаты | Близко к классическому | Чуть хуже VFSA | Улучшает базовый МА |

1. Общий итог:
2. **Вывод**

Реализован код программы с графическим интерфейсом, реализующий муравьиный алгоритм для решения задачи коммивояжера. Проведен анализ скорости и точности решения TSP разными семействами алгоритмов. В том числе доказаны теоретические результаты быстрой скорости выполнения алгоритма ближайшего соседа и более высокой точности сложных эвристических алгоритмов (имитации отжига и муравьиных).

# **Листинг программы**

import tkinter as tk

from tkinter import ttk, messagebox, simpledialog, filedialog

import pandas as pd

import random

import time

import math

​

# Глобальные переменные для хранения вершин и рёбер

vertices = {}

edges = []

current\_edge = None

graph\_type = "undirected"

​

DEFAULT\_PHEROMONE = 0.1

DEFAULT\_PHEROMONE\_WEIGHT = 1.0

DEFAULT\_LENGTH\_WEIGHT = 2.0

DEFAULT\_EVAPORATION = 0.5

DEFAULT\_ITERATIONS = 100

DEFAULT\_ANTS = 10

DEFAULT\_Q = 100

​

def nearest\_neighbor\_tsp(graph, start\_vertex):

   adjacency\_list = {}

   for \_, row in graph.iterrows():

       v1, v2, length = row['вершина 1'], row['вершина 2'], float(row['длина'])

       if v1 not in adjacency\_list:

           adjacency\_list[v1] = {}

       if v2 not in adjacency\_list:

           adjacency\_list[v2] = {}

       adjacency\_list[v1][v2] = length

       if graph\_type == "undirected":

           adjacency\_list[v2][v1] = length

​

   visited = set()

   path = [start\_vertex]

   visited.add(start\_vertex)

   total\_distance = 0

​

   current\_vertex = start\_vertex

​

   while len(visited) < len(adjacency\_list):

       nearest\_vertex = None

       nearest\_distance = float('inf')

​

       for neighbor, distance in adjacency\_list[current\_vertex].items():

           if neighbor not in visited and distance < nearest\_distance:

               nearest\_vertex = neighbor

               nearest\_distance = distance

​

       if nearest\_vertex is None:

           break

​

       path.append(nearest\_vertex)

       visited.add(nearest\_vertex)

       total\_distance += nearest\_distance

       current\_vertex = nearest\_vertex

​

   if len(path) == len(adjacency\_list) and start\_vertex in adjacency\_list[current\_vertex]:

       path.append(start\_vertex)

       total\_distance += adjacency\_list[current\_vertex][start\_vertex]

       return path, total\_distance

   else:

       return None, None

​

def ant\_colony\_tsp(graph, start\_vertex=None, use\_templates=False):

   try:

       alpha = float(entry\_pheromone.get() or DEFAULT\_PHEROMONE\_WEIGHT)

       beta = float(entry\_length.get() or DEFAULT\_LENGTH\_WEIGHT)

       evaporation = float(entry\_evaporation.get() or DEFAULT\_EVAPORATION)

       iterations = int(entry\_iterations.get() or DEFAULT\_ITERATIONS)

       ants\_count = int(entry\_ants.get() or DEFAULT\_ANTS)

       Q = float(entry\_q.get() or DEFAULT\_Q)

   except ValueError:

       messagebox.showerror("Ошибка", "Некорректные параметры алгоритма")

       return None, None

​

   adjacency\_list = {}

   vertices = set()

   pheromone = {}

   for \_, row in graph.iterrows():

       v1, v2, length = row['вершина 1'], row['вершина 2'], float(row['длина'])

       initial\_pheromone = float(row['феромон']) if 'феромон' in row and row['феромон'] else DEFAULT\_PHEROMONE

       vertices.add(v1)

       vertices.add(v2)

       if v1 not in adjacency\_list:

           adjacency\_list[v1] = {}

       if v2 not in adjacency\_list:

           adjacency\_list[v2] = {}

       adjacency\_list[v1][v2] = length

       pheromone[(v1, v2)] = initial\_pheromone

       if graph\_type == "undirected":

           adjacency\_list[v2][v1] = length

           pheromone[(v2, v1)] = initial\_pheromone

   vertices = list(vertices)

   best\_path = None

   best\_length = float('inf')

   templates = []

   if use\_templates:

       # Создаем несколько шаблонных путей

       for v in vertices[:min(5, len(vertices))]:

           path, length = nearest\_neighbor\_tsp(graph, v)

           if path and length:

               templates.append((path, length))

   for iteration in range(iterations):

       ants\_paths = []

       for ant in range(ants\_count):

           if start\_vertex and start\_vertex in vertices:

               current\_vertex = start\_vertex

           else:

               current\_vertex = random.choice(vertices)

           path = [current\_vertex]

           visited = set([current\_vertex])

           path\_length = 0

           while len(visited) < len(vertices):

               possible\_transitions = []

               total = 0.0

               for neighbor in adjacency\_list[current\_vertex]:

                   if neighbor not in visited:

                       tau = pheromone.get((current\_vertex, neighbor), DEFAULT\_PHEROMONE) \*\* alpha

                       eta = (1.0 / adjacency\_list[current\_vertex][neighbor]) \*\* beta

                       weight = tau \* eta

                       possible\_transitions.append((neighbor, weight))

                       total += weight

               if not possible\_transitions:

                   break  # нет возможных переходов

               if total > 0:

                   probabilities = [(neighbor, weight/total) for neighbor, weight in possible\_transitions]

                   probabilities.sort(key=lambda x: -x[1])

                   r = random.random()

                   cumulative = 0.0

                   for neighbor, prob in probabilities:

                       cumulative += prob

                       if r <= cumulative:

                           next\_vertex = neighbor

                           break

                   else:

                       next\_vertex = probabilities[-1][0]

               else:

                   next\_vertex = random.choice([t[0] for t in possible\_transitions])

               path.append(next\_vertex)

               visited.add(next\_vertex)

               path\_length += adjacency\_list[current\_vertex][next\_vertex]

               current\_vertex = next\_vertex

           if len(path) == len(vertices) and path[0] in adjacency\_list[path[-1]]:

               path.append(path[0])

               path\_length += adjacency\_list[path[-2]][path[-1]]

               ants\_paths.append((path, path\_length))

               if path\_length < best\_length:

                   best\_length = path\_length

                   best\_path = path.copy()

       # Добавляем шаблонные пути (если используется модификация)

       if use\_templates and templates:

           ants\_paths.extend(templates)

       for edge in pheromone:

           pheromone[edge] \*= (1.0 - evaporation)

       for path, path\_length in ants\_paths:

           delta\_pheromone = Q / path\_length

           for i in range(len(path)-1):

               edge = (path[i], path[i+1])

               if edge in pheromone:

                   pheromone[edge] += delta\_pheromone

       if iteration % 10 == 0:

           print(f"Iteration {iteration}: best length = {best\_length}")

   return best\_path, best\_length

​

def add\_edge\_to\_table(v1, v2, length, pheromone=DEFAULT\_PHEROMONE):

   tree.insert("", "end", values=(v1, v2, length, pheromone))

​

def on\_click(event):

   global current\_edge

​

   for vertex, (x, y) in vertices.items():

       if abs(event.x - x) < 10 and abs(event.y - y) < 10:  # 10 пикселей

           if current\_edge is None:

               current\_edge = vertex

           else:

               # Запрашиваем длину ребра

               length = simpledialog.askfloat("Длина ребра", f"Введите длину ребра между {current\_edge} и {vertex}:")

               if length is None:

                   return

​

               # Запрашиваем начальный феромон

               pheromone = simpledialog.askfloat("Феромон ребра",

                                                f"Введите начальный феромон для ребра между {current\_edge} и {vertex} (по умолчанию {DEFAULT\_PHEROMONE}):",

                                                initialvalue=DEFAULT\_PHEROMONE)

               if pheromone is None:

                   pheromone = DEFAULT\_PHEROMONE

               edges.append((current\_edge, vertex, length, pheromone))

               add\_edge\_to\_table(current\_edge, vertex, length, pheromone)

               current\_edge = None

               draw\_input\_graph()

           break

   else:

       vertex = f"V{len(vertices) + 1}"

       vertices[vertex] = (event.x, event.y)

       draw\_input\_graph()

​

def draw\_input\_graph():

   input\_canvas.delete("all")

​

   for edge in edges:

       if len(edge) == 3:

           v1, v2, length = edge

           pheromone = DEFAULT\_PHEROMONE

       else:

           v1, v2, length, pheromone = edge

       x1, y1 = vertices[v1]

       x2, y2 = vertices[v2]

       input\_canvas.create\_line(x1, y1, x2, y2, width=2, fill="black")

       input\_canvas.create\_text((x1 + x2) / 2, (y1 + y2) / 2, text=f"{length:.2f}", fill="red")

       input\_canvas.create\_text((x1 + x2) / 2, (y1 + y2) / 2 + 15, text=f"φ={pheromone:.2f}", fill="blue")

​

   for vertex, (x, y) in vertices.items():

       input\_canvas.create\_oval(x - 10, y - 10, x + 10, y + 10, fill="red")

       input\_canvas.create\_text(x, y, text=vertex, fill="black")

​

def draw\_output\_graph(path):

   output\_canvas.delete("all")

​

   for i in range(len(path) - 1):

       v1, v2 = path[i], path[i + 1]

       x1, y1 = vertices[v1]

       x2, y2 = vertices[v2]

       output\_canvas.create\_line(x1, y1, x2, y2, width=2, fill="green")

​

   for vertex, (x, y) in vertices.items():

       output\_canvas.create\_oval(x - 10, y - 10, x + 10, y + 10, fill="red")

       output\_canvas.create\_text(x, y, text=vertex, fill="black")

​

def solve\_tsp():

   edges\_table = []

   for item in tree.get\_children():

       edge = tree.item(item, "values")

       edges\_table.append(edge)

​

   if not edges\_table:

       messagebox.showwarning("Ошибка", "Добавьте хотя бы одно ребро!")

       return

​

   graph = pd.DataFrame(edges\_table, columns=["вершина 1", "вершина 2", "длина", "феромон"])

​

   start\_vertex = entry\_start.get().strip()

   if not start\_vertex:

       messagebox.showwarning("Ошибка", "Введите стартовую вершину!")

       return

​

   all\_vertices = set(graph['вершина 1']).union(set(graph['вершина 2']))

   if start\_vertex not in all\_vertices:

       messagebox.showwarning("Ошибка", f"Стартовая вершина '{start\_vertex}' не найдена в графе!")

       return

​

   start\_time = time.time()

   path, total\_distance = ant\_colony\_tsp(graph, start\_vertex)

   end\_time = time.time()

   execution\_time = end\_time - start\_time

   print(f"Время выполнения муравьиного алгоритма: {execution\_time:.4f} секунд")

​

   if path is None:

       messagebox.showwarning("Результат", "Гамильтонов цикл не найден")

       result\_path.config(text="")

       result\_distance.config(text="")

       draw\_output\_graph([])

   else:

       result\_path.config(text=f"{' -> '.join(path)}")

       result\_distance.config(text=f"{total\_distance:.2f}")

       draw\_output\_graph(path)

​

def solve\_tsp\_with\_templates():

   edges\_table = []

   for item in tree.get\_children():

       edge = tree.item(item, "values")

       edges\_table.append(edge)

​

   if not edges\_table:

       messagebox.showwarning("Ошибка", "Добавьте хотя бы одно ребро!")

       return

​

   graph = pd.DataFrame(edges\_table, columns=["вершина 1", "вершина 2", "длина", "феромон"])

​

   start\_vertex = entry\_start.get().strip()

   if not start\_vertex:

       messagebox.showwarning("Ошибка", "Введите стартовую вершину!")

       return

​

   all\_vertices = set(graph['вершина 1']).union(set(graph['вершина 2']))

   if start\_vertex not in all\_vertices:

       messagebox.showwarning("Ошибка", f"Стартовая вершина '{start\_vertex}' не найдена в графе!")

       return

​

   start\_time = time.time()

   path, total\_distance = ant\_colony\_tsp(graph, start\_vertex, use\_templates=True)

   end\_time = time.time()

   execution\_time = end\_time - start\_time

   print(f"Время выполнения муравьиного алгоритма с шаблонами: {execution\_time:.4f} секунд")

​

   if path is None:

       messagebox.showwarning("Результат", "Гамильтонов цикл не найден")

       result\_path.config(text="")

       result\_distance.config(text="")

       draw\_output\_graph([])

   else:

       result\_path.config(text=f"{' -> '.join(path)}")

       result\_distance.config(text=f"{total\_distance:.2f}")

       draw\_output\_graph(path)

​

def clear\_canvas():

   global vertices, edges, current\_edge

   vertices = {}

   edges = []

   current\_edge = None

   input\_canvas.delete("all")

   output\_canvas.delete("all")

   tree.delete(\*tree.get\_children())

   result\_path.config(text="")

   result\_distance.config(text="")

   draw\_input\_graph()

   draw\_output\_graph([])

​

def set\_graph\_type(type):

   global graph\_type

   graph\_type = type

   messagebox.showinfo("Тип графа", f"Тип графа изменён на {'неориентированный' if type == 'undirected' else 'ориентированный'}")

​

def load\_graph\_from\_file():

   file\_path = filedialog.askopenfilename(filetypes=[("CSV files", "\*.csv")])

   if not file\_path:

       return

​

   try:

       df = pd.read\_csv(file\_path)

       required\_columns = ["вершина 1", "вершина 2", "длина"]

       if not all(col in df.columns for col in required\_columns):

           messagebox.showerror("Ошибка", f"Файл должен содержать колонки: {', '.join(required\_columns)}")

           return

​

       clear\_canvas()

       for \_, row in df.iterrows():

           v1, v2, length = row['вершина 1'], row['вершина 2'], float(row['длина'])

           pheromone = float(row['феромон']) if 'феромон' in row and pd.notna(row['феромон']) else DEFAULT\_PHEROMONE

           if v1 not in vertices:

               vertices[v1] = (random.randint(50, 550), random.randint(50, 250))

           if v2 not in vertices:

               vertices[v2] = (random.randint(50, 550), random.randint(50, 250))

           edges.append((v1, v2, length, pheromone))

           add\_edge\_to\_table(v1, v2, length, pheromone)

​

       draw\_input\_graph()

   except Exception as e:

       messagebox.showerror("Ошибка", f"Не удалось загрузить файл: {e}")

​

root = tk.Tk()

root.title("Задача коммивояжера (Муравьиный алгоритм)")

​

# Левый фрейм для параметров и кнопок

left\_frame = ttk.Frame(root)

left\_frame.pack(side=tk.LEFT, fill=tk.BOTH, padx=10, pady=10)

​

label\_parameters = ttk.Label(left\_frame, text="Параметры алгоритма", font=("Arial", 12))

label\_parameters.pack(pady=5)

​

label\_pheromone = ttk.Label(left\_frame, text="Вес феромона (α):")

label\_pheromone.pack(pady=2)

entry\_pheromone = ttk.Entry(left\_frame)

entry\_pheromone.insert(0, str(DEFAULT\_PHEROMONE\_WEIGHT))

entry\_pheromone.pack(pady=2)

​

label\_length = ttk.Label(left\_frame, text="Вес длины (β):")

label\_length.pack(pady=2)

entry\_length = ttk.Entry(left\_frame)

entry\_length.insert(0, str(DEFAULT\_LENGTH\_WEIGHT))

entry\_length.pack(pady=2)

​

label\_evaporation = ttk.Label(left\_frame, text="Испарение феромона:")

label\_evaporation.pack(pady=2)

entry\_evaporation = ttk.Entry(left\_frame)

entry\_evaporation.insert(0, str(DEFAULT\_EVAPORATION))

entry\_evaporation.pack(pady=2)

​

label\_iterations = ttk.Label(left\_frame, text="Кол-во итераций:")

label\_iterations.pack(pady=2)

entry\_iterations = ttk.Entry(left\_frame)

entry\_iterations.insert(0, str(DEFAULT\_ITERATIONS))

entry\_iterations.pack(pady=2)

​

label\_ants = ttk.Label(left\_frame, text="Кол-во муравьев:")

label\_ants.pack(pady=2)

entry\_ants = ttk.Entry(left\_frame)

entry\_ants.insert(0, str(DEFAULT\_ANTS))

entry\_ants.pack(pady=2)

​

label\_q = ttk.Label(left\_frame, text="Количество феромона (Q):")

label\_q.pack(pady=2)

entry\_q = ttk.Entry(left\_frame)

entry\_q.insert(0, str(DEFAULT\_Q))

entry\_q.pack(pady=2)

​

separator\_count = ttk.Separator(left\_frame, orient=tk.HORIZONTAL)

separator\_count.pack(fill=tk.X, pady=10)

​

button\_solve = ttk.Button(left\_frame, text="Муравьиный алгоритм", command=solve\_tsp)

button\_solve.pack(fill=tk.BOTH, pady=5)

​

button\_solve\_templates = ttk.Button(left\_frame, text="С шаблонами", command=solve\_tsp\_with\_templates)

button\_solve\_templates.pack(fill=tk.BOTH, pady=5)

​

label\_start = ttk.Label(left\_frame, text="Стартовая вершина:")

label\_start.pack(pady=5)

entry\_start = ttk.Entry(left\_frame)

entry\_start.pack()

​

button\_load = ttk.Button(left\_frame, text="Загрузить граф из файла", command=load\_graph\_from\_file)

button\_load.pack(fill=tk.BOTH, pady=5)

​

separator\_left = ttk.Separator(root, orient=tk.VERTICAL)

separator\_left.pack(side=tk.LEFT, fill=tk.Y, padx=5)

​

# Центральный фрейм для холстов

center\_frame = ttk.Frame(root)

center\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)

​

label\_input\_graph = ttk.Label(center\_frame, text="Входной граф", font=("Arial", 12))

label\_input\_graph.pack(pady=5)

​

graph\_type\_frame = ttk.Frame(center\_frame)

graph\_type\_frame.pack(pady=5)

​

graph\_type\_var = tk.StringVar(value="undirected")

radio\_undirected = ttk.Radiobutton(graph\_type\_frame, text="Неориентированный", variable=graph\_type\_var, value="undirected", command=lambda: set\_graph\_type("undirected"))

radio\_undirected.pack(side=tk.LEFT, padx=5)

radio\_directed = ttk.Radiobutton(graph\_type\_frame, text="Ориентированный", variable=graph\_type\_var, value="directed", command=lambda: set\_graph\_type("directed"))

radio\_directed.pack(side=tk.LEFT, padx=5)

​

input\_canvas = tk.Canvas(center\_frame, width=600, height=300, bg="white")

input\_canvas.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)

​

label\_output\_graph = ttk.Label(center\_frame, text="Выходной граф", font=("Arial", 12))

label\_output\_graph.pack(pady=2)

​

output\_canvas = tk.Canvas(center\_frame, width=600, height=300, bg="white")

output\_canvas.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

​

control\_frame = ttk.Frame(center\_frame)

control\_frame.pack(fill=tk.X, padx=10, pady=10)

​

button\_clear = ttk.Button(control\_frame, text="Очистить", command=clear\_canvas)

button\_clear.pack(fill=tk.BOTH, padx=5)

​

input\_canvas.bind("<Button-1>", on\_click)

​

separator\_right = ttk.Separator(root, orient=tk.VERTICAL)

separator\_right.pack(side=tk.LEFT, fill=tk.Y, padx=5)

​

# Правый фрейм для таблицы и результатов

right\_frame = ttk.Frame(root, width=300)

right\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, padx=10, pady=10)

​

label\_edges = ttk.Label(right\_frame, text="Ребра", font=("Arial", 12))

label\_edges.pack(pady=5)

​

style = ttk.Style()

style.layout("my.Treeview", [

  ("Treeview.field", {"sticky": "nswe", "border": "1", "children": [

      ("Treeview.padding", {"sticky": "nswe", "children": [

          ("Treeview.treearea", {"sticky": "nswe"})

      ]})

  ]})

])

style.configure("my.Treeview", font=("Arial", 10), rowheight=25, borderwidth=1, relief="solid")

style.configure("my.Treeview.Heading", font=("Arial", 10, "bold"), background="gray")

style.map("my.Treeview", background=[("selected", "#347083")])

​

tree = ttk.Treeview(right\_frame, columns=("вершина 1", "вершина 2", "длина", "феромон"), show="headings", style="my.Treeview")

tree.heading("вершина 1", text="Вершина 1")

tree.heading("вершина 2", text="Вершина 2")

tree.heading("длина", text="Длина")

tree.heading("феромон", text="Феромон")

tree.column("вершина 1", width=110, anchor="center")

tree.column("вершина 2", width=110, anchor="center")

tree.column("длина", width=110, anchor="center")

tree.column("феромон", width=110, anchor="center")

tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

​

separator\_results\_top = ttk.Separator(right\_frame, orient=tk.HORIZONTAL)

separator\_results\_top.pack(fill=tk.X, pady=10)

​

result\_frame = ttk.Frame(right\_frame)

result\_frame.pack(fill=tk.BOTH, padx=10, pady=10)

​

label\_path = ttk.Label(result\_frame, text="Полученный путь:", font=("Arial", 12))

label\_path.grid(row=0, column=0, padx=5, pady=5, sticky="w")

result\_path = ttk.Label(result\_frame, text="", font=("Arial", 12), wraplength=400)

result\_path.grid(row=0, column=1, padx=5, pady=5, sticky="w")

​

label\_distance = ttk.Label(result\_frame, text="Длина пути:", font=("Arial", 12))

label\_distance.grid(row=1, column=0, padx=5, pady=5, sticky="w")

result\_distance = ttk.Label(result\_frame, text="", font=("Arial", 12))

result\_distance.grid(row=1, column=1, padx=5, pady=5, sticky="w")

​

separator\_results\_bottom = ttk.Separator(right\_frame, orient=tk.HORIZONTAL)

separator\_results\_bottom.pack(fill=tk.X, pady=10)

​root.mainloop()