



**МИНОБРНАУКИ РОССИИ**  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**"МИРЭА - Российский технологический университет"**  
**РТУ МИРЭА**

---

**Институт Информационных Технологий**  
**Кафедра Вычислительной Техники**

**ПРАКТИЧЕСКАЯ РАБОТА №4**

**по дисциплине**  
**«Системный анализ данных в системах поддержки принятия**  
**решений»**  
**Муравьиный алгоритм**

Студент группы: ИКБО-04-22

Кликушин В.И.  
(Ф. И.О. студента)

Преподаватель

Железняк Л.М.  
(Ф.И.О. преподавателя)

Москва 2024

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 МУРАВЬИНЫЙ АЛГОРИТМ .....	4
1.1 Описание алгоритма .....	4
1.2 Постановка задачи.....	5
1.3 Математическая модель .....	5
1.4 Ручной расчёт .....	9
1.5 Программная реализация .....	14
ЗАКЛЮЧЕНИЕ .....	15
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	16
ПРИЛОЖЕНИЯ.....	17

# ВВЕДЕНИЕ

Алгоритм муравьиной колонии был предложен в начале 1990-х годов итальянским ученым Марко Дориго. Основным источником вдохновения для создания алгоритма — наблюдение за поведением реальных муравьиных колоний, особенно за тем, как они находят кратчайшие пути между источниками пищи и гнездом. Муравьи, используя феромоны для меток маршрутов, демонстрируют способность коллективного интеллекта: несмотря на ограниченные когнитивные возможности отдельных особей, колония в целом находит оптимальные решения.

Муравьиный алгоритм широко используется для решения комбинаторных задач, таких как поиск кратчайших путей, распределение ресурсов и оптимизация маршрутов. Его ключевое преимущество — способность эффективно находить хорошие решения даже в задачах с высокой сложностью и большим числом ограничений.

В логистике алгоритм помогает находить оптимальные маршруты доставки, снижая транспортные издержки и время выполнения заказов. В области телекоммуникаций он применяется для маршрутизации данных, обеспечивая высокую надежность и пропускную способность сетей. В задачах проектирования сетей или систем управления ресурсами алгоритм используется для распределения вычислительных мощностей и уменьшения затрат. Кроме того, он находит применение в биоинформатике, анализе данных и решении сложных задач графовой структуры.

Способность алгоритма эффективно моделировать процессы взаимодействия множества агентов делает его мощным инструментом в научных и инженерных областях. Его универсальность и надежность обеспечивают успех в применении к задачам, где традиционные методы сталкиваются с трудностями.

# 1 МУРАВЬИНЫЙ АЛГОРИТМ

## 1.1 Описание алгоритма

Муравьиный алгоритм использует множество агентов-муравьев, каждый из которых представляет потенциальное решение задачи. Эти агенты перемещаются в дискретном пространстве поиска, обычно представленном в виде графа, где вершины соответствуют элементам задачи, а ребра — возможным переходам между ними. Ключевая особенность алгоритма заключается в использовании искусственных феромонов, которые обновляются в процессе работы. Муравьи, выполняя переходы между вершинами графа, оставляют феромонный след, интенсивность которого зависит от качества найденного решения. Каждый муравей строит свое решение последовательно, переходя от одной вершины графа к другой на основе вероятностного правила. В ходе работы алгоритма реализуется механизм испарения феромона, который предотвращает его чрезмерное накопление и позволяет избегать преждевременной сходимости. Также обновление феромона способствует усилению тех маршрутов, которые были найдены наиболее успешными муравьями.

Обобщая выше сказанное, алгоритм состоит из следующих ключевых шагов:

1. Инициализация начальных параметров (количество муравьёв, коэффициент испарения феромона и другие).
2. Построение решения каждым муравьем (Формула 1.4.1).
3. Уменьшение концентрации феромона на рёбрах графа (Формула 1.4.2).
4. Добавление феромона на рёбра, которые участвовали в построенных маршрутах (Формулы 1.4.3–1.4.4).
5. Завершение работы, если критерий останова выполнен, иначе возврат к пункту номер два.

## **1.2 Постановка задачи**

Цель работы: реализовать задачу коммивояжера муравьиным алгоритмом для нахождения приближённого оптимального маршрута.

Изучить муравьиный алгоритм, выбрать предметную область для задачи коммивояжера, произвести ручной расчёт двух итераций алгоритма для двух муравьёв, разработать программную реализацию муравьиного алгоритма для задачи коммивояжера.

Условие оригинальной задачи коммивояжёра: «Представим себе коммивояжёра, который должен посетить ряд городов, находящихся в разных местах. Его цель — начать и завершить путь в одном и том же городе, посетив каждый из остальных городов ровно один раз и минимизировав при этом общий пройденный путь (или затраты на поездку). Задача заключается в нахождении такого маршрута, который будет иметь минимальную длину среди всех возможных маршрутов, удовлетворяющих этим условиям».

Выбранная предметная область для задачи коммивояжёра: оптимизация маршрута подачи документов в высшие учебные заведения города Москвы.

Условие задачи коммивояжёра для выбранной предметной области: Абитуриент приезжает в Москву для подачи документов в несколько высших учебных заведений. Его цель — начать и завершить путь в отеле, где он остановился, посетить приёмные комиссии всех университетов ровно один раз и минимизировать при этом общее время и затраты на перемещение по городу. Необходимо найти оптимальный маршрут, который позволит посетить все выбранные университеты, минимизируя суммарное время, затраченное на дорогу, учитывая, что каждый университет можно посетить только один раз.

## **1.3 Математическая модель**

Особенность задачи заключается в том, что её модель представлена в виде ориентированного взвешенного полного графа. Между каждой парой вершин

(университетов и отелем) существует направленное ребро с уникальным весом для каждого направления. Это означает, что путь от одной вершины к другой может иметь одну стоимость (время достижения), а обратный путь — совершенно другую. В отличие от неориентированного графа, где вес ребра между двумя вершинами одинаков независимо от направления, в задаче учитываются асимметрии, такие как односторонние улицы, разная интенсивность движения или различия в транспортных затратах по направлению туда и обратно.

Представленная модель ориентированного взвешенного графа не только абстрактно описывает структуру задачи, но и отражает её практическое содержание.

Для ручного расчёта число университетов в задаче взято равным шести. С начальной стартовой точкой в отеле граф содержит семь вершин. Количество возможных маршрутов для рассматриваемой задачи может быть рассчитано по Формуле 1.3.1.

$$(n - 1)! = (7 - 1)! = 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720, \quad (1.3.1)$$

где  $n$  – количество вершин в полном взвешенном ориентированном графе.

Будем считать, что между двумя вершинами в описанном графе существует условно два ребра, каждое из которых представлено уникальным весом. В таком случае, число рёбер, вес которых необходимо заполнить, рассчитано по Формуле 1.3.2.

$$E = n * (n - 1) = 7 * 6 = 42, \quad (1.3.2)$$

где  $E$  – количество рёбер в полном взвешенном ориентированном графе.

Местоположения приёмных комиссий университетов и отеля представлены их реальными адресами. Адрес объекта представлен названием города, наименованием улицы, номером дома (строения). Допустимо указание

почтового индекса. В качестве отправной точки выбран отель «Звёзды Арбата». Информация о пунктах маршрута отображена в Таблице 1.3.1.

Таблица 1.3.1 – Характеристики пунктов маршрута

Наименование пункта	Сокращённое название	Адрес
Отель «Звёзды Арбата»	Отель	Москва, Новый Арбат, 32
Московский государственный технический университет им. Н.Э. Баумана	МГТУ	Москва, 2-я Бауманская ул., д. 5, стр. 1
Московский физико-технический институт	МФТИ	Московская область, г. Долгопрудный, Институтский переулок, д. 9.
Национальный исследовательский ядерный университет «МИФИ»	МИФИ	Москва, Каширское шоссе, 31
Российская академия народного хозяйства и государственной службы при Президенте РФ	РАНХиГС	проспект Вернадского, 84с1, Москва, 119606
Университет науки и технологий МИСИС	МИСИС	Ленинский проспект, 2/4, Москва, 119049
МИРЭА – Российский технологический университет	МИРЭА	проспект Вернадского, 86с2, Москва

Для расчёта расстояний между вершинами в графе использовался сервис «Google Maps». Под расстоянием понимается время, необходимое на то, чтобы добраться из одной точки в другую по лучшему маршруту (преимущественно автомобиль). Данные получены 02.11.2024 в 14:15 часов. С учётом времени суток, времени года, погодных условий, пробок и аварий на дорогах, а также прочих факторов полученные сведения о времени на маршрут могут отличаться.

Сведения о маршрутах могут быть получены из любого другого навигационного сервиса, предоставляющего актуальную информацию о времени в пути.

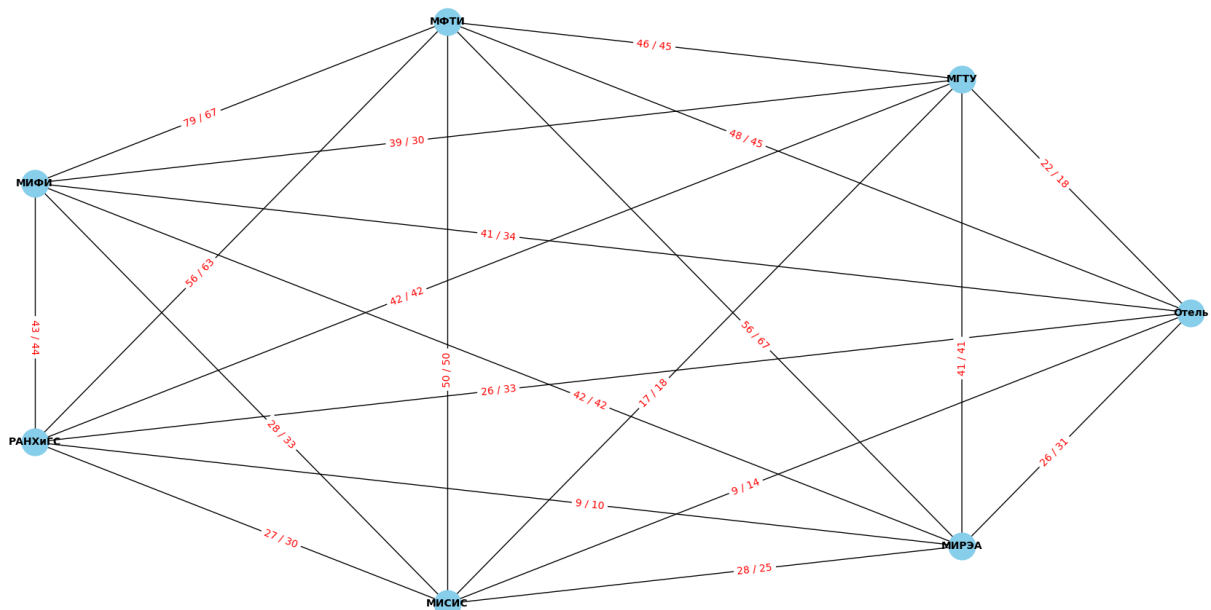
Рассчитанные длины рёбер сведены в Таблицу 1.3.2 с указанием сокращённых названий вершин, составляющих ребро.

Таблица 1.3.2 – Длины рёбер в графе

Ребро	Длина ребра
Отель - МГТУ	22
Отель - МФТИ	48
Отель - МИФИ	41
Отель - РАНХиГС	26
Отель - МИСИС	9
Отель - МИРЭА	26
МГТУ - Отель	18
МГТУ - МФТИ	46
МГТУ - МИФИ	39
МГТУ - РАНХиГС	42
МГТУ - МИСИС	17
МГТУ - МИРЭА	41
МФТИ - Отель	45
МФТИ - МГТУ	45
МФТИ - МИФИ	79
МФТИ - РАНХиГС	56
МФТИ - МИСИС	50
МФТИ - МИРЭА	56
МИФИ - Отель	34
МИФИ - МГТУ	30
МИФИ - МФТИ	67
МИФИ - РАНХиГС	43
МИФИ - МИСИС	28
МИФИ - МИРЭА	42
РАНХиГС - Отель	33
РАНХиГС - МГТУ	42
РАНХиГС - МФТИ	63
РАНХиГС - МИФИ	44
РАНХиГС - МИСИС	27
РАНХиГС - МИРЭА	9
МИСИС - Отель	14
МИСИС - МГТУ	18
МИСИС - МФТИ	50
МИСИС - МИФИ	33
МИСИС - РАНХиГС	30
МИСИС - МИРЭА	28
МИРЭА - Отель	31
МИРЭА - МГТУ	41
МИРЭА - МФТИ	67
МИРЭА - МИФИ	42
МИРЭА - РАНХиГС	10
МИРЭА - МИСИС	25

Единицей измерения для длины ребра является число минут на дорогу между пунктами. Граф, представляющий модель задачи с заданными рёбрами и вершинами представлен на Рисунке 1.3.1.





**Рисунок 1.3.1 – Граф модели задачи**

Веса рёбер на графе представлены двумя числами, записанными через косую черту.

## 1.4 Ручной расчёт

Коэффициент  $\alpha$  принят равным 1,1. Константа  $p$ , определяющая скорость испарения равна 0,25. Феромоны для рёбер инициализированы случайными значениями от нуля до единицы. Начальной точкой для всех муравьёв является «Отель».

Ниже приведены случайно заполненные концентрации феромонов для каждой дуги, соединяющий отель с университетами:

$$\tau_{01}(0) = 0.8329;$$

$$\tau_{02}(0) = 0.7895;$$

$$\tau_{03}(0) = 0.8434;$$

$$\tau_{04}(0) = 0.3393;$$

$$\tau_{05}(0) = 0.0015;$$

$$\tau_{06}(0) = 0.8038,$$

где  $\tau_{ij}$  – концентрация феромона между вершинами (i,j),

0 – номер итерации.

В каждой вершине каждый муравей должен выбрать следующую дугу пути на основе вероятностей перехода. (Формула 1.4.1).

$$p_{ij}^k(t) = \frac{\tau_{ij}^\alpha(t)}{\sum_{j \in N_i^k} \tau_{ij}^\alpha(t)}, \quad (1.4.1)$$

где  $p$  – вычисленная вероятность;

$k$  – номер муравья;

$i$  – индекс вершины отправления;

$j$  – индекс вершины назначения;

$t$  – номер итерации алгоритма;

$\tau_{ij}$  – концентрация феромона между вершинами (i,j);

$\alpha$  – положительная константа, которая определяет влияние концентрации феромона;

$N_i^k$  – множество возможных вершин, связанных с вершиной с индексом  $i$ .

Ниже рассчитана вероятность перехода из вершины под номером 0 (Отель) в вершины, которые не были посещены для первого муравья на первой итерации:

$$p_{01}^1(0) = \frac{\tau_{01}^\alpha(0)}{\sum_{j \in N_0^1} \tau_{0j}^\alpha(0)} = \frac{0,8329^{1,1}}{3,5098} = \frac{0,8178}{3,5098} = 0,2330$$

$$p_{02}^1(0) = \frac{\tau_{02}^\alpha(0)}{\sum_{j \in N_0^2} \tau_{0j}^\alpha(0)} = \frac{0,7895^{1,1}}{3,5098} = \frac{0,7710}{3,5098} = 0,2196$$

$$p_{03}^1(0) = \frac{\tau_{03}^\alpha(0)}{\sum_{j \in N_0^3} \tau_{0j}^\alpha(0)} = \frac{0,8434^{1,1}}{3,5098} = \frac{0,8291}{3,5098} = 0,2362$$

$$p_{04}^1(0) = \frac{\tau_{04}^\alpha(0)}{\sum_{j \in N_0^4} \tau_{0j}^\alpha(0)} = \frac{0,3393^{1,1}}{3,5098} = \frac{0,3045}{3,5098} = 0,0867$$

$$p_{05}^1(0) = \frac{\tau_{05}^\alpha(0)}{\sum_{j \in N_0^5} \tau_{0j}^\alpha(0)} = \frac{0,0015^{1,1}}{3,5098} = \frac{0,0007}{3,5098} = 0,0002$$

$$p_{06}^1(0) = \frac{\tau_{06}^\alpha(0)}{\sum_{j \in N_0^6} \tau_{0j}^\alpha(0)} = \frac{0,8038^{1,1}}{3,5098} = \frac{0,7864}{3,5098} = 0,2240$$

Далее генерируется случайное число  $r = 0,3714$ . Объявляется переменная-счётчик, которая будет хранить нарастающую сумму нескольких вероятностей  $p_{ij}^k(t)$ . Как только полученная сумма превысит сгенерированное число, будет принят узел под номером  $j$ . На первой итерации сумма равна 0,2330. Значит, вершина под номером 1 отбрасывается. На следующей итерации сумма равна 0,4526, что больше, чем сгенерированное число. Таким образом, вторая вершина, в которую направится муравей, имеет номер 2.

Вершина заносится в список посещённых и продолжается построение маршрута таким же образом.

Построенный маршрут для первого муравья на первой итерации представлен в Листинге 1.4.1.

*Листинг 1.4.1 – Маршрут первого муравья на первой итерации*

Отель -> МФТИ -> МИФИ -> РАНХиГС -> МГТУ -> МИСИС -> МИРЭА -> Отель
---

Стоимость построенного маршрута:  $48 + 79 + 43 + 42 + 17 + 28 + 31 = 288$ .

Аналогичным образом построен маршрут для второго муравья на первой итерации (Листинг 1.4.2).

*Листинг 1.4.2 – Маршрут второго муравья на первой итерации*

Отель -> МИФИ -> МГТУ -> МИРЭА -> РАНХиГС -> МФТИ -> МИСИС -> Отель
---

Стоимость построенного маршрута:  $41 + 30 + 41 + 10 + 63 + 50 + 14 = 249$ .

В конце итерации уменьшается концентрация феромона на дугах графа в соответствии с Формулой 1.4.2.

$$\tau_{ij}(t + 1) = (1 - p)\tau_{ij}(t) \quad (1.4.2)$$

где  $p$  – константа, определяющая скорость испарения.

Процесс уменьшения концентрации феромона для рёбер от вершины под номером нуль (Отель) до всех университетов описан ниже:

$$\tau_{01}(1) = (1 - 0,25)\tau_{01}(0) = 0,75 * 0,8329 = 0,6246$$

$$\tau_{02}(1) = (1 - 0,25)\tau_{02}(0) = 0,75 * 0,7895 = 0,5921$$

$$\tau_{03}(1) = (1 - 0,25)\tau_{03}(0) = 0,75 * 0,8434 = 0,6325$$

$$\tau_{04}(1) = (1 - 0,25)\tau_{04}(0) = 0,75 * 0,3395 = 0,2545$$

$$\tau_{05}(1) = (1 - 0,25)\tau_{05}(0) = 0,75 * 0,0015 = 0,0011$$

$$\tau_{06}(1) = (1 - 0,25)\tau_{06}(0) = 0,75 * 0,8038 = 0,6028$$

Муравей помечает свой пройденный путь, отложив на каждой дуге феромон в соответствие с Формулой 1.4.3.

$$\Delta\tau_{ij}^k(t) = \frac{1}{L^k(t)}, \quad (1.4.3)$$

где  $L^k$  – длина пути, построенная муравьём под номером k на итерации t.

Для каждой дуги графа концентрация феромона определяется по Формуле 1.4.4.

$$\tau_{ij}(t + 1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t), \quad (1.4.4)$$

где  $n_k$  – число муравьёв.

Маршрут второго муравья может быть задан с использованием порядковых номеров вершин (Листинг 1.4.3).

*Листинг 1.4.3 – Маршрут второго муравья на первой итерации, заданным индексами*

0 -> 3 -> 1 -> 6 -> 4 -> 2 -> 5 -> 0
--------------------------------------

Процесс обновления концентрации феромона на дугах, по которым прополз второй муравей представлен ниже:

$$\tau_{03}(1) = \tau_{03}(0) + \sum_{k=1}^{n_k} \Delta\tau_{03}^k(0) = 0,6325 + \frac{1}{249} = 0,6365;$$

$$\tau_{31}(1) = \tau_{31}(0) + \sum_{k=1}^{n_k} \Delta\tau_{31}^k(0) = 0,2175 + \frac{1}{249} = 0,2215;$$

$$\tau_{16}(1) = \tau_{16}(0) + \sum_{k=1}^{n_k} \Delta\tau_{16}^k(0) = 0,7071 + \frac{1}{249} = 0,7111;$$

$$\tau_{64}(1) = \tau_{64}(0) + \sum_{k=1}^{n_k} \Delta\tau_{64}^k(0) = 0,3531 + \frac{1}{249} = 0,3571;$$

$$\tau_{42}(1) = \tau_{42}(0) + \sum_{k=1}^{n_k} \Delta\tau_{42}^k(0) = 0,7092 + \frac{1}{249} = 0,7133;$$

$$\tau_{25}(1) = \tau_{25}(0) + \sum_{k=1}^{n_k} \Delta\tau_{25}^k(0) = 0,6243 + \frac{1}{249} = 0,6283;$$

$$\tau_{50}(1) = \tau_{50}(0) + \sum_{k=1}^{n_k} \Delta\tau_{50}^k(0) = 0,0186 + \frac{1}{249} = 0,0226;$$

Таким образом происходит обновление концентрации феромонов для каждой дуги, которая участвовала в маршрутах муравьёв.

Под конец первой итерации лучший путь найден вторым муравьём, и его длина составляет 249.

На второй итерации путь, построенный первым муравьём представлен в Листинге 1.4.4.

*Листинг 1.4.4 – Маршрут первого муравья на второй итерации*

ОТЕЛЬ -> МГТУ -> РАНХиГС -> МФТИ -> МИРЭА -> МИСИС -> МИФИ -> ОТЕЛЬ
---

Длина пути:  $22 + 42 + 63 + 56 + 25 + 33 + 34 = 275$ .

Путь, проложенный вторым муравьём представлен в Листинге 1.4.5.

*Листинг 1.4.5 – Маршрут второго муравья на второй итерации*

ОТЕЛЬ -> МГТУ -> МФТИ -> МИСИС -> МИРЭА -> МИФИ -> РАНХиГС -> ОТЕЛЬ
---

Длина пути:  $22 + 46 + 50 + 28 + 42 + 43 + 33 = 264$ .

Процесс испарения и обновления концентрации феромона остаётся прежним. Муравьи выбрали худшие маршруты в силу недостаточной концентрации феромона после первой итерации. С ростом количества муравьёв-агентов, числа итераций, а также уменьшением коэффициента, определяющего скорость испарения, алгоритм будет сходиться к одному маршруту.

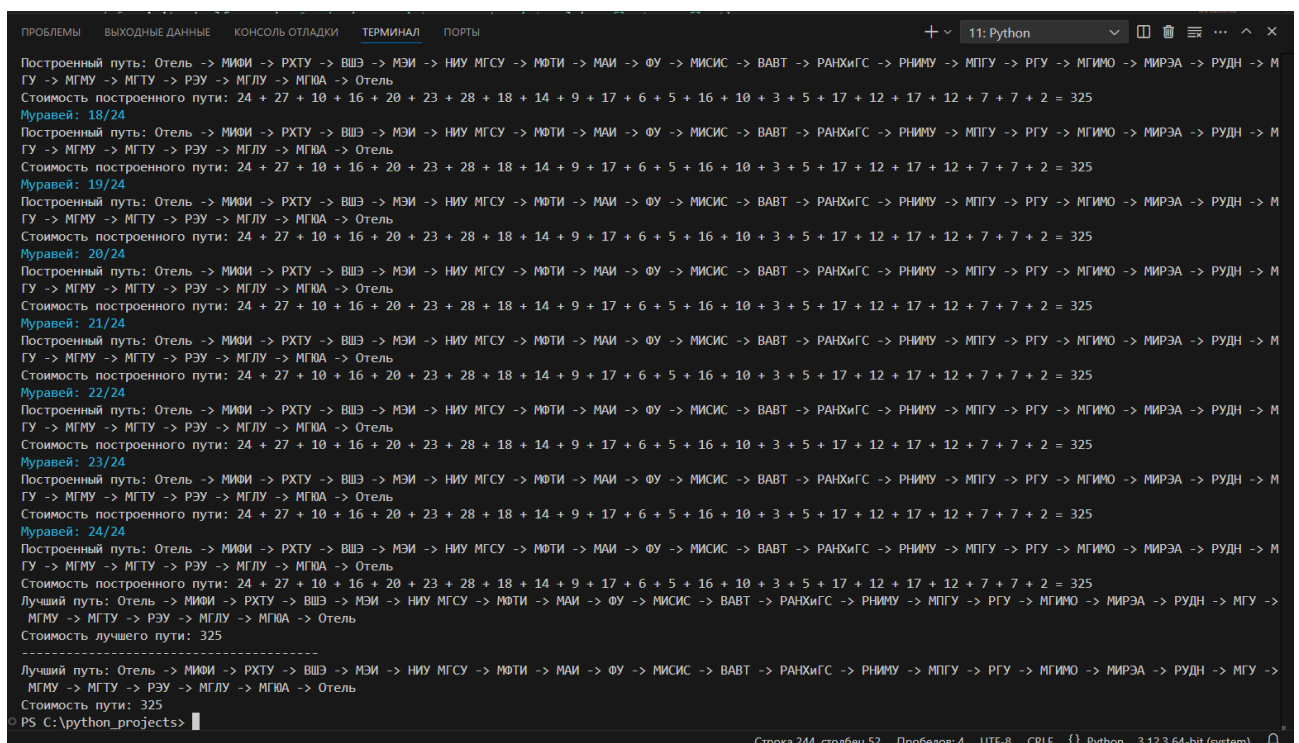
## 1.5 Программная реализация

Разработан класс `Vertex`, представляющий вершину графа. Экземпляр класса содержит атрибуты `name` — наименование вершины, `short_name` — сокращённое название вершины, `address` — физический адрес объекта.

Реализован класс `Graph`, который используется для представления модели задачи. Экземпляр класса `Graph` хранит атрибуты `vertices` — список экземпляров класса `Vertex` и `adjacency_matrix` — двумерный список, представляющий матрицу весов. Класс содержит минимальный набор операций для работы с графом: добавление ребра, добавление вершины, удаление ребра, удаление вершины, отрисовка графа, заполнение матрицы весов из файла. Для реализации муравьиного алгоритма написан отдельный класс.

Код простого муравьиного алгоритма для задачи коммивояжера представлен в Приложении А.

Результат работы муравьиного алгоритма представлен на Рисунке 1.5.1.



```
ПРОБЛЕМЫ Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ
+ 11: Python
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 18/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 19/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 20/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 21/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 22/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 23/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Муравей: 24/24
Построенный путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость построенного пути: 24 + 27 + 10 + 16 + 20 + 23 + 28 + 18 + 14 + 9 + 17 + 6 + 5 + 16 + 10 + 3 + 5 + 17 + 12 + 17 + 12 + 7 + 7 + 2 = 325
Лучший путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость лучшего пути: 325
-----
Лучший путь: Отель -> МИФИ -> РХТУ -> ВШЭ -> МЭИ -> НИУ МГСУ -> МФТИ -> МАИ -> ФУ -> МИСИС -> БАВТ -> РАНХиГС -> РНИМУ -> МПГУ -> РГУ -> МГИМО -> МИРЭА -> РУДН -> МГУ -> МГМУ -> МГТУ -> РЭУ -> МГЛУ -> МГЮА -> Отель
Стоимость пути: 325
PS C:\python_projects>
```

Рисунок 1.5.1 – Результат работы муравьиного алгоритма для задачи коммивояжера

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был изучен алгоритм муравьиной колонии, проведён его ручной расчёт двух итерация для двух муравьёв для, а также разработана программа на языке Python для решения задачи коммивояжера с обходом университетов.

Основное преимущество муравьиного алгоритма заключается в его способности эффективно находить приближённые решения для сложных комбинаторных задач благодаря механизму феромонной памяти, которая усиливает перспективные маршруты и помогает избегать неоптимальных решений. Алгоритм сочетает локальный поиск каждого муравья с глобальным взаимодействием через феромонное обновление, что делает его мощным инструментом для задач маршрутизации, планирования и оптимизации графов.

Проведённые эксперименты показали, что параметры алгоритма, такие как скорость испарения феромонов, количество муравьёв, число итераций, существенно влияют на его сходимость. Таким образом, эффективность муравьиного алгоритма определяется тонкой настройкой его параметров и выбором подходящей модели для конкретной задачи.

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс] : Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
2. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
3. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
4. Муравьиный алгоритм. [Электронный ресурс]: Википедия. – URL: [https://ru.wikipedia.org/wiki/Муравьиный\\_алгоритм](https://ru.wikipedia.org/wiki/Муравьиный_алгоритм) (Дата обращения: 18.11.2024).



## **ПРИЛОЖЕНИЯ**

Приложение А — Код реализации простого муравьиного алгоритма.

## Приложение А

### Код реализации простого муравьиного алгоритма

*Листинг А – Реализация простого муравьиного алгоритма*

```
import random
import certifi
import time
import csv
import re
import time
import functools
import matplotlib.pyplot as plt
import networkx as nx
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from dataclasses import dataclass, field
from tabulate import tabulate
from typing import List, Tuple

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--ignore-certificate-errors')
chrome_options.add_argument('--ignore-ssl-errors')
chrome_options.add_argument(f"--ssl-certificates-path={certifi.where()}")
chrome_options.add_experimental_option(
    "excludeSwitches", ['enable-automation', 'enable-logging'])

@dataclass(frozen=True)
class Vertex:
    """Класс для представления узла графа, который включает название,
    сокращенное имя и адрес."""
    name: str
    short_name: str = field(compare=False)
    address: str

    def __str__(self) -> str:
        return self.short_name

class Graph:
    def __init__(self, vertices: List[Vertex]):
        """Инициализирует граф с заданными узлами и матрицей смежности. """
        self.vertices = vertices
        self.adjacency_matrix = [[1 if i != j else 0 for j in range(
            len(vertices))] for i in range(len(vertices))]

    @property
    def vertices(self) -> List[Vertex]:
        return self.__vertices

    @vertices.setter
    def vertices(self, vertices: List[Vertex]) -> None:
        self.__vertices = vertices

    @property
    def adjacency_matrix(self) -> List[List[int]]:
        return self.__adjacency_matrix

    @adjacency_matrix.setter
```

### Продолжение Листинга А

```
def adjacency_matrix(self, adjacency_matrix: List[List[int]]) -> None:
    self.__adjacency_matrix = adjacency_matrix

def show_graph(self):
    '''Рисует граф, используя текущую матрицу весов.'''
    G = nx.Graph()
    for i, row in enumerate(self.adjacency_matrix):
        for j, weight in enumerate(row):
            if i < j and (weight != 0 or self.adjacency_matrix[j][i] != 0):
                G.add_edge(self.vertices[i].short_name,
self.vertices[j].short_name,
                        weight_ab=weight,
weight_ba=self.adjacency_matrix[j][i])
    pos = nx.circular_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=700,
            node_color="skyblue", font_size=10, font_weight="bold")
    edge_labels = {}
    for u, v, d in G.edges(data=True):
        edge_labels[(u, v)] = f"{d['weight_ab']} / {d['weight_ba']}"
    nx.draw_networkx_edge_labels(
        G, pos, edge_labels=edge_labels, font_color="red", label_pos=0.6)
    plt.show()

def print_adjacency_matrix(self, show_routes=True) -> None:
    '''Выводит матрицу весов в консоль.'''
    column_names = [vertex.short_name for vertex in self.vertices]
    table = tabulate(self.adjacency_matrix, headers=column_names,
                    tablefmt='simple', maxcolwidths=3)

    print(table)
    if show_routes:
        for i, vertex_i in enumerate(self.vertices):
            for j, vertex_j in enumerate(self.vertices):
                if i != j and self.adjacency_matrix[i][j] > 0:
                    print(f'Длина ребра от {vertex_i.short_name} до {
vertex_j.short_name}:
{self.adjacency_matrix[i][j]}')

    @staticmethod
    def timer(func):
        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            start = time.perf_counter()
            val = func(*args, **kwargs)
            end = time.perf_counter()
            work_time = end - start
            print(f'Время выполнения {func.__name__}: {
round(work_time, 4)} сек.')
            return val
        return wrapper

    @timer
    def set_weights(self, gui: bool = True) -> None:
        '''Заполняет матрицу смежности временем достижения между узлами,
используя Google Maps.'''
        if not gui:
            chrome_options.add_argument('--headless')
            with webdriver.Chrome(options=chrome_options) as browser:
                url = 'https://www.google.ru/maps'
                browser.get(url)
                route = WebDriverWait(browser, 3).until(
                    EC.element_to_be_clickable((By.CLASS_NAME, 'hArJGc')))
```

```
route.click()
time.sleep(0.5)
k = 1
for vertex_i in self.vertices:
    for vertex_j in self.vertices:
        if vertex_i != vertex_j:
            print(
                f'\033[91m{vertex_i.address}\033[0m -
\033[92m{vertex_j.address}\033[0m')
            departure_point = WebDriverWait(browser, 10).until(
                EC.element_to_be_clickable((By.CLASS_NAME, 'tactile-
searchbox-input'))))
            departure_point.clear()
            departure_point.send_keys(vertex_i.address)
            destination_point = WebDriverWait(browser, 10).until(
                EC.element_to_be_clickable((By.CSS_SELECTOR, '[aria-
controls="sbsg51"]'))))
            destination_point.clear()
            destination_point.send_keys(vertex_j.address)
            destination_point.send_keys(Keys.ENTER)
            result = WebDriverWait(browser, 10).until(
                EC.element_to_be_clickable((By.CLASS_NAME,
'Fk3sm')))).text
            print(f'Маршрут №{
k}/{len(self.vertices) ** 2 - len(self.vertices)}:
{result}')
            self.adjacency_matrix[self.vertices.index(
                vertex_i)][self.vertices.index(vertex_j)] = result
            k += 1

def set_weights_from_file(self, filename: str) -> None:
    '''Устанавливает веса из файла с матрицей смежности.'''
    with open(filename, 'r', encoding='utf-8') as file:
        reader = csv.reader(file)
        self.adjacency_matrix = [[int(i) for i in row] for row in reader]

def delete_vertex(self, vertex: Vertex) -> None:
    '''Удаляет узел и соответствующие ребра из графа.'''
    try:
        vertex_index = self.vertices.index(vertex)
    except ValueError:
        return
    self.vertices.remove(vertex)
    self.adjacency_matrix = [
        row[:vertex_index] + row[vertex_index+1:] for row in
self.adjacency_matrix
    ]
    self.adjacency_matrix = [
        row for i, row in enumerate(self.adjacency_matrix) if i !=
vertex_index
    ]

def delete_edge(self, first_vertex: Vertex, second_vertex: Vertex) -> None:
    '''Удаляет ребро между двумя узлами.'''
    try:
        first_index = self.vertices.index(first_vertex)
        second_index = self.vertices.index(second_vertex)
    except ValueError:
        return
    self.adjacency_matrix[first_index][second_index] = 0
    self.adjacency_matrix[second_index][first_index] = 0
```

```

    def set_edge(self, first_vertex: Vertex, second_vertex: Vertex, value: int)
-> None:
    '''Устанавливает вес ребра между двумя узлами.'''
    try:
        first_vertex_index = self.vertices.index(first_vertex)
        second_vertex_index = self.vertices.index(second_vertex)
    except ValueError:
        return
    self.adjacency_matrix[first_vertex_index][second_vertex_index] = value
    self.adjacency_matrix[second_vertex_index][first_vertex_index] = value

    def add_vertex(self, vertex: Vertex) -> None:
    '''Добавляет новый узел и обновляет матрицу смежности.'''
    self.vertices.append(vertex)
    self.adjacency_matrix.append(
        [1 for _ in range(len(self.vertices) - 1)])
    for i in range(len(self.vertices) - 1):
        self.adjacency_matrix[i].append(1)
    self.adjacency_matrix[len(self.vertices) - 1].append(0)

    def calculate_cost(self, path: List[Vertex]) -> Tuple[int, str]:
    '''Вычисляет стоимость (длину) маршрута для заданного пути.'''
    cost = 0
    calculations = []
    for i in range(len(path) - 1):
        v_from = self.vertices.index(path[i])
        v_to = self.vertices.index(path[i + 1])
        weight = self.adjacency_matrix[v_from][v_to]
        calculations.append(str(weight))
        cost += weight
    return cost, " + ".join(calculations) + f" = {cost}"

    def normalize_matrix(self):
    '''Нормализует матрицу весов'''
    for i in range(len(self.adjacency_matrix)):
        for j in range(len(self.adjacency_matrix)):
            value = str(self.adjacency_matrix[i][j])
            if re.fullmatch(r'\d+ ч \d+ мин.', value):
                hours = int(
                    re.search(r'\d+ ч', value).group().removesuffix('ч'))
                minutes = int(
                    re.search(r'\d+ мин.',
value).group().removesuffix('мин.'))
                new_value = hours * 60 + minutes
            elif re.fullmatch(r'\d ч', value):
                new_value = int(value.removesuffix('ч.')) * 60
            else:
                new_value = int(value.removesuffix('мин.'))
            self.adjacency_matrix[i][j] = new_value

    def save_matrix_to_csv(self, filename: str) -> None:
    '''Сохраняет матрицу весов в файл.'''
    with open(filename, 'w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        for row in self.adjacency_matrix:
            writer.writerow(row)

class SimpleAntColonyOptimization:
    def __init__(self, graph: Graph, k_max: int, num_ants: int, alpha: float, p:
float):

```

```

'''
Инициализирует алгоритм муравьиной колонии.
Параметры:
    graph (Graph): Граф, для которого выполняется оптимизация.
    k_max (int): Максимальное количество итераций.
    num_ants (int): Количество муравьев.
    alpha (float): Влияние феромонов.
    p (float): Коэффициент испарения феромонов.
'''
self.graph = graph
self.k_max = k_max
self.num_ants = num_ants
self.alpha = alpha
self.p = p
self.pheromones = [[random.random() if i != j else 0 for j in
graph.vertices]
                    for i in graph.vertices]
self.best_path = None
self.best_cost = float('inf')

def optimize(self, output = True):
    '''
    Выполняет оптимизацию для поиска лучшего пути.
    Возвращает:
        Tuple[List[int], float]: Лучший путь и его стоимость.
    '''
    for k in range(self.k_max):
        if output:
            print('\033[92m' + f"Итерация: {k + 1}/{self.k_max}" +
'\033[0m')
        paths = []
        path_costs = []
        for ant in range(self.num_ants):
            path = self.construct_path()
            cost, calculation = self.graph.calculate_cost(
                [self.graph.vertices[i] for i in path])
            if output:
                print('\033[96m' +
                    f"Муравей: {ant + 1}/{self.num_ants}" + '\033[0m')
                print(f"Построенный путь: {
                    ' -> '.join(self.graph.vertices[node].short_name for
node in path)}")
                print(f"Стоимость построенного пути: {calculation}")
            paths.append(path)
            path_costs.append(cost)
            if cost < self.best_cost:
                self.best_cost = cost
                self.best_path = path
        self.evaporate_pheromones()
        self.deposit_pheromones(paths, path_costs)
        if output:
            print(f"Лучший путь: {
                ' -> '.join(self.graph.vertices[node].short_name for node in
self.best_path)}")
            print(f"Стоимость лучшего пути: {self.best_cost}")
        return self.best_path, self.best_cost

def construct_path(self) -> List[int]:
    '''
    Строит маршрут для одного муравья.
    Возвращает:

```

```

        List[int]: Последовательность индексов вершин.
    """
    visited = set()
    current_node = 0
    path = [current_node]
    visited.add(current_node)
    while len(visited) < len(self.graph.vertices):
        next_node = self.choose_next_node(current_node, visited)
        path.append(next_node)
        visited.add(next_node)
        current_node = next_node
    path.append(0)
    return path

def choose_next_node(self, current_node: int, visited: set) -> int:
    """
    Выбирает следующую вершину на основе вероятностей.
    Параметры:
        current_node (int): Текущая вершина.
        visited (set): Набор уже посещенных вершин.
    Возвращает:
        int: Индекс следующей вершины.
    """
    probabilities = []
    neighbors = range(len(self.graph.vertices))
    for j in neighbors:
        if j not in visited:
            pheromone = self.pheromones[current_node][j]
            probabilities.append((j, pheromone ** self.alpha))
    total = sum(prob[1] for prob in probabilities)
    probabilities = [(node, prob / total) for node, prob in probabilities]
    random_choice = random.uniform(0, 1)
    cumulative = 0
    for node, prob in probabilities:
        cumulative += prob
        if random_choice <= cumulative:
            return node
    return neighbors[-1]

def evaporate_pheromones(self):
    """Испаряет феромоны на всех ребрах графа."""
    for i in range(len(self.pheromones)):
        for j in range(len(self.pheromones[i])):
            self.pheromones[i][j] *= (1 - self.p)

def deposit_pheromones(self, paths: List[List[int]], costs: List[float]):
    """
    Обновляет феромоны на ребрах на основе пройденных путей.
    Параметры:
        paths (List[List[int]]): Список всех пройденных путей.
        costs (List[float]): Список стоимости каждого пути.
    """
    for i in range(len(self.best_path) - 1):
        self.pheromones[self.best_path[i]][self.best_path[i + 1]] += 1 / self.best_cost

    for path, cost in zip(paths, costs):
        for i in range(len(path) - 1):
            self.pheromones[path[i]][path[i + 1]] += 1 / cost

vertex_0 = Vertex('Звезды Арбата',

```

*Продолжение Листинга А*

```
        'ОТЕЛЬ',
        'Москва, Новый Арбат, 32')
vertex_1 = Vertex('Московский государственный университет им. М.В. Ломоносова',
        'МГУ',
        'Москва, Западный административный округ, район Раменки,
территория Ленинские Горы, 1, стр. 52')
vertex_2 = Vertex('Московский государственный технический университет им. Н.Э.
Баумана',
        'МГТУ',
        'Москва, 2-я Бауманская ул., д. 5, стр. 1')
vertex_3 = Vertex('Московский физико-технический институт',
        'МФТИ',
        'Московская область, г. Долгопрудный, Институтский переулок,
д. 9.')
vertex_4 = Vertex('Национальный исследовательский ядерный университет «МИФИ»',
        'МИФИ',
        'Москва, Каширское шоссе, 31')
vertex_5 = Vertex('Высшая школа экономики',
        'ВШЭ',
        'Милютинский переулок, 2/9, Москва, 101000')
vertex_6 = Vertex('Московский государственный институт международных отношений
МИД РФ',
        'МГИМО',
        'проспект Вернадского, 76кГ, Москва, 119454')
vertex_7 = Vertex('Российская академия народного хозяйства и государственной
службы при Президенте РФ',
        'РАНХиГС',
        'проспект Вернадского, 84с1, Москва, 119606')
vertex_8 = Vertex('Финансовый университет при Правительстве РФ',
        'ФУ',
        'Ленинградский проспект, 51к1, Москва, 125167')
vertex_9 = Vertex('Первый Московский государственный медицинский университет им.
И.М. Сеченова',
        'МГМУ',
        'Трубецкая улица, 8с2, Москва, 119048')
vertex_10 = Vertex('Российский экономический университет им. Г.В. Плеханова',
        'РЭУ',
        'Стремянный переулок, 36, Москва, 115054')
vertex_11 = Vertex('Университет науки и технологий МИСИС',
        'МИСИС',
        'Ленинский проспект, 2/4, Москва, 119049')
vertex_12 = Vertex('Российский университет дружбы народов',
        'РУДН',
        'улица Миклухо-Маклая, 6, Москва, 117198')
vertex_13 = Vertex('Российский национальный исследовательский медицинский
университет им. Н.И. Пирогова',
        'РНИМУ',
        'улица Островитянова, 1с7, Москва, 117513')
vertex_14 = Vertex('Московский авиационный институт',
        'МАИ',
        'Волоколамское шоссе, 4к6, Москва, 125310')
vertex_15 = Vertex('Национальный исследовательский университет «МЭИ»',
        'МЭИ',
        'Красноказарменная ул., 17 строение 1Г, Москва, 111250')
vertex_16 = Vertex('Московский государственный юридический университет им. О.Е.
Кутафина',
        'МГЮА',
        'Садовая-Кудринская улица, 9с1, Москва, 123242')
vertex_17 = Vertex('Российский государственный университет нефти и газа им. И.
М. Губкина',
        'РГУ',
```



### Окончание Листинга А

```
'Ленинский проспект, 65к1, Москва, 119296')
vertex_18 = Vertex('Московский педагогический государственный университет',
                  'МПГУ',
                  'проспект Вернадского, 88, Москва, 119571')
vertex_19 = Vertex('Национальный исследовательский Московский государственный
строительный университет',
                  'НИУ МГСУ',
                  'Ярославское шоссе, 26к1, Москва, 129337')
vertex_20 = Vertex('Московский государственный лингвистический университет',
                  'МГЛУ',
                  'улица Остоженка, 38с1, Москва, 119034')
vertex_21 = Vertex('Всероссийская академия внешней торговли',
                  'ВАВТ',
                  'Воробьёвское шоссе, 6А, Москва, 119285')
vertex_22 = Vertex('Российский химико-технологический университет им. Д.И.
Менделеева',
                  'РХТУ',
                  'Миусская площадь, 9, Москва')
vertex_23 = Vertex('МИРЭА - Российский технологический университет',
                  'МИРЭА',
                  'проспект Вернадского, 86с2, Москва')
vertices = [vertex_0, vertex_1, vertex_2, vertex_3, vertex_4,
            vertex_5, vertex_6, vertex_7, vertex_8, vertex_9,
            vertex_10, vertex_11, vertex_12, vertex_13, vertex_14,
            vertex_15, vertex_16, vertex_17, vertex_18, vertex_19,
            vertex_20, vertex_21, vertex_22, vertex_23]

graph = Graph(vertices)
graph.set_weights()
graph.print_adjacency_matrix(False)
graph.normalize_matrix()
# graph.set_weights_from_file('universities_info.csv')
# graph.print_adjacency_matrix(False)
# graph.show_graph()

saco = SimpleAntColonyOptimization(
    graph, k_max=200, num_ants=24, alpha=1.190657414100356,
    p=0.4160457456608921)
best_path, best_cost = sacco.optimize()
print('-' * 40)
print("Лучший путь:",
      " -> ".join(graph.vertices[node].short_name for node in best_path))
print("Стоимость пути:", best_cost)

# test_graph = Graph([vertex_0, vertex_2, vertex_3, vertex_4,
#                      vertex_7, vertex_11, vertex_23])
# # test_graph.set_weights()
# # test_graph.print_adjacency_matrix(False)
# # test_graph.normalize_matrix()
# # test_graph.save_matrix_to_csv('shit.csv')
# test_graph.set_weights_from_file('universities_test.csv')
# # test_graph.print_adjacency_matrix()
# # test_graph.show_graph()

# sacco = SimpleAntColonyOptimization(
#     test_graph, k_max=100, num_ants=7, alpha=1.1, p=0.25)
# best_path, best_cost = sacco.optimize()
# print('-' * 40)
# print("Лучший путь:",
#       " -> ".join(test_graph.vertices[node].short_name for node in best_path))
# print("Стоимость пути:", best_cost)
```