



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ
по Лабораторной работе № 6
по курсу «Анализ алгоритмов»
на тему: «Методы решения задачи коммивояжёра»
Вариант № 10141

Студент ИУ7-56Б
(Группа)

(Подпись, дата)

Александрова А. Д.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Волкова Л. Л.
(И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитический раздел	4
1.1 Алгоритм полного перебора	4
1.2 Муравьиный алгоритм	4
2 Конструкторский раздел	7
2.1 Требования к программе	7
2.2 Проектирование алгоритмов	7
3 Технологический раздел	10
3.1 Средства реализации	10
3.2 Реализация алгоритмов	10
3.3 Функциональные тесты	13
4 Исследовательский раздел	14
4.1 Технические характеристики	14
4.2 Временные характеристики	14
4.3 Параметризация муравьиного алгоритма	15
ЗАКЛЮЧЕНИЕ	16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	17
ПРИЛОЖЕНИЕ А	18

ВВЕДЕНИЕ

Целью данной лабораторной работы является исследование методов решения задачи коммивояжера.

Задачи лабораторной работы:

- 1) реализация алгоритмов решения задачи коммивояжера — полного перебора и муравьиного алгоритма;
- 2) исследование ресурсной эффективности реализованных алгоритмов для разных параметров;
- 3) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Индивидуальный вариант для выполнения данной лабораторной работы:

- неориентированный граф;
- с «элитными» муравьями;
- с возвращением в начальную точку.

1 Аналитический раздел

Задача коммивояжера — это классическая задача в теории графов и комбинаторной оптимизации [1]. Она формулируется следующим образом:

Предположим, что у вас есть набор городов и известные расстояния между каждой парой городов. Задача состоит в том, чтобы найти самый короткий маршрут, который начинается в одном городе, проходит через все остальные города ровно один раз и возвращается в исходный город.

1.1 Алгоритм полного перебора

Для решения задачи коммивояжера для n городов методом полного перебора, необходимо построить все $n!$ возможных путей, посчитать для каждого длину и, сравнив полученные значения, выбрать один из путей минимальной длины.

Таким образом, данный алгоритм обладает высокой сложностью $O(n!)$, что делает его неэффективным, а при больших значениях n — технически невозможным.

1.2 Муравьиный алгоритм

Муравьиный алгоритм — это метаэвристический метод, вдохновленный поведением муравьев, который используется для решения различных задач оптимизации, включая задачу коммивояжера. Основная идея заключается в имитации процесса поиска пищи муравьями и их способности оставлять феромоны, которые влияют на поведение других муравьев. Реализация же заключается в моделировании нескольких, заданных параметром t_{max} , дней жизни муравьиной колонии.

Каждый день муравьи выходят из вершин графа (количество муравьев равно количеству вершин в графе) и находят по одному решению каждый. Взаимодействуют они при помощи феромона, который оставляют на пройденных маршрутах. Количество феромонов зависит от качества найденного решения (например, длины маршрута). Чем короче маршрут, тем больше феромонов оставляет муравей. Феромоны испаряются со временем согласно формуле (1.1), что позволяет избежать прежнего доминирования неэффективных решений.

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1 - p), \quad (1.1)$$

где $p \in (0; 1)$ — коэффициент испарения.

Матрица феромонов инициализируется начальным значением феромона τ_0 .

При выборе следующего города муравей учитывает количество феромонов на ребрах и расстояние до следующего города. Вероятность перехода от города i к городу j определяется формулой (1.2)

$$p_{ij} = \begin{cases} \frac{\eta_{ij}^\alpha \cdot \tau_{ij}^\beta}{\sum_{q \notin J_k} \eta_{iq}^\alpha \cdot \tau_{iq}^\beta}, & j \notin J_k \\ 0, & j \in J_k \end{cases} \quad (1.2)$$

где a — коэффициент жадности, b — коэффициент стадности, τ_{ij} — количество феромонов на ребре ij , η_{ij} — привлекательность ребра ij , J_k — список посещенных за текущий день городов.

В конце дня в каждую дугу $i - j$ для всех полученных маршрутов P добавляется $\Delta\tau_{ij}$ феромона, где

$$\Delta\tau_{ij}(t) = \frac{Q}{\sigma(P)}, \quad (1.3)$$

где Q — количество феромона, которое распределяется на все дуги за день, а $\sigma(P)$ — сумма всех дуг маршрута P .

Этапы работы алгоритма в течение одного дня.

- 1) Каждый муравей оказывается в вершине с номером $k \in [1, n]$ и принимает за $V = \{k\}$ список посещенных в этот день вершин.
- 2) Каждый муравей оценивает вероятность перехода в следующий город по формуле (1.2).
- 3) Каждый муравей выбирает город для перехода (город с наибольшей вероятностью перехода).
- 4) Если есть непосещенные города, переходим к шагу 2.
- 5) Муравей оказался в вершине, с которой начинал путь в начале дня. Происходит испарение феромона по формуле (1.1).

- 6) Все маршруты P просматриваются и для каждого в составляющие его дуги добавляются феромоны по формуле (1.3).
- 7) Если есть "элитные" муравьи, шаг 6 повторяется для наилучших маршрутов.

Сложность такого алгоритма составляет $O(t_{max} \cdot n^2)$.

2 Конструкторский раздел

2.1 Требования к программе

К программе предъявлены следующие требования:

- на вход подается матрица расстояний, описывающая граф;
- результатом работы программы является целое число — длина оптимального пути и массив индексов вершин, описывающий этот путь;
- необходимо наличие пользовательского интерфейса для выбора действий и алгоритма работы.

2.2 Проектирование алгоритмов

В процессе проектирования программы были разработаны алгоритмы решения задачи коммивояжера, представленные на рисунках 2.1 и 2.2.

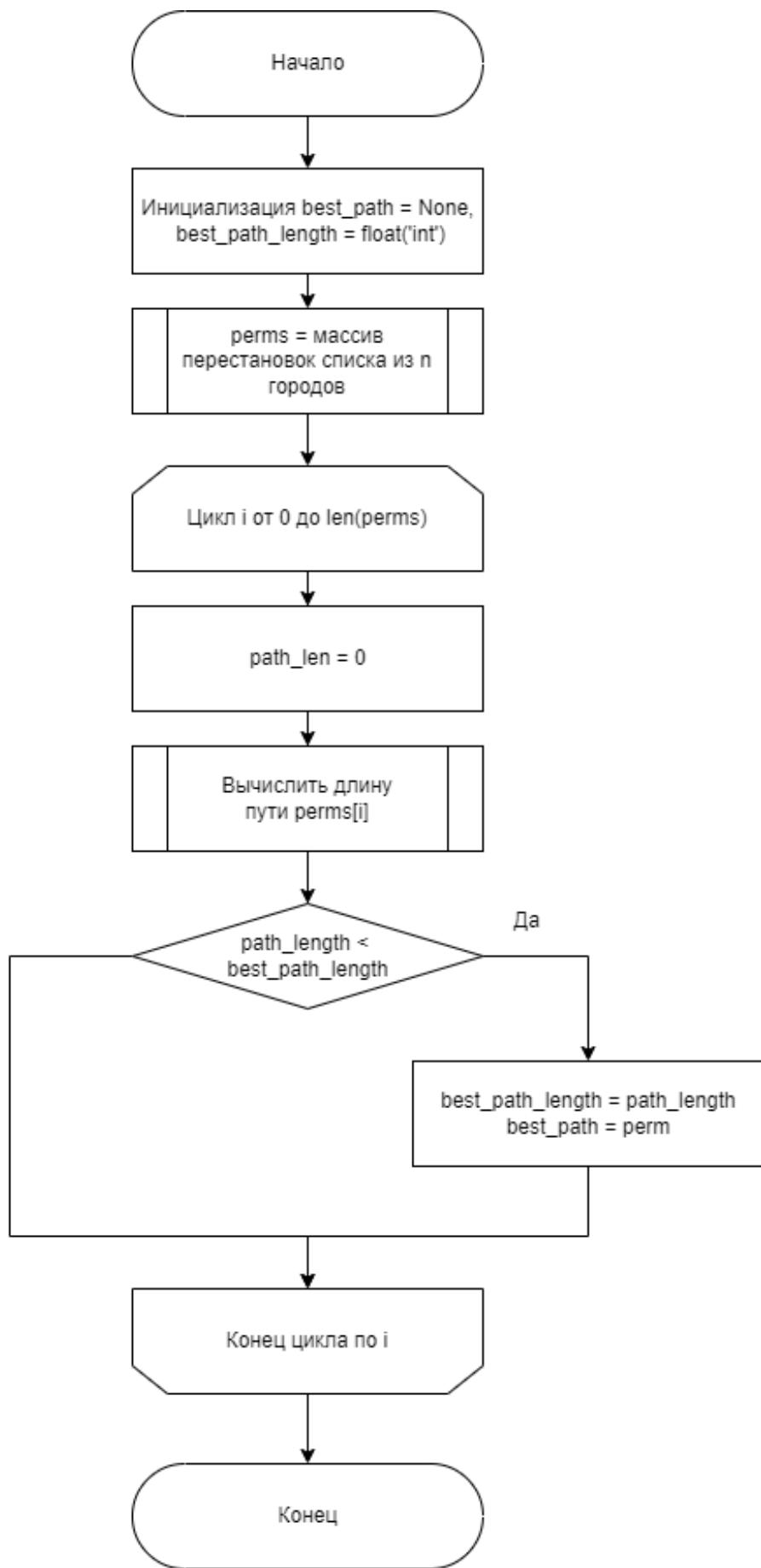


Рисунок 2.1 – Алгоритм полного перебора

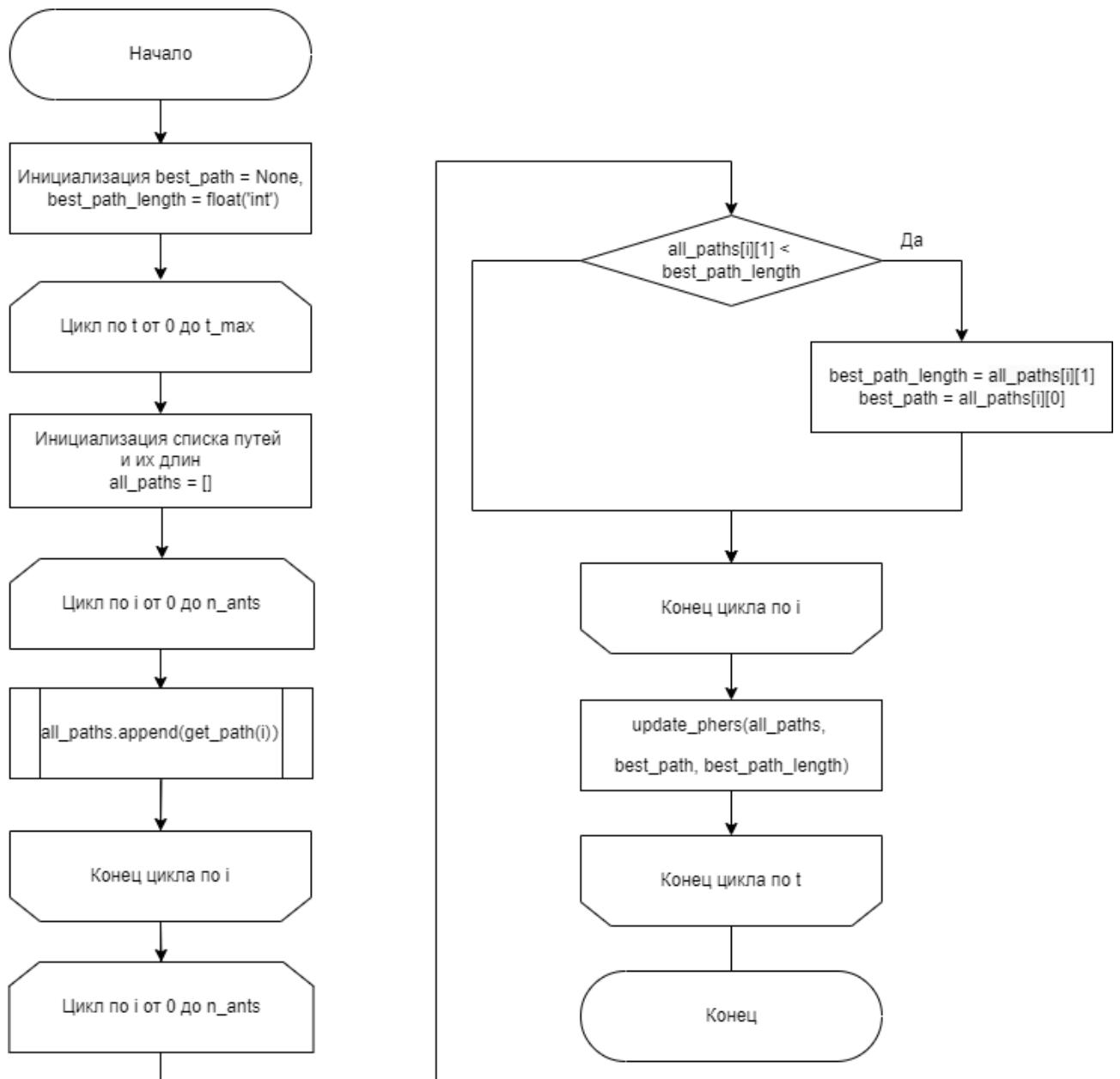


Рисунок 2.2 – Муравьиный алгоритм

3 Технологический раздел

3.1 Средства реализации

Для реализации данной лабораторной был выбран язык Python [2], так как он содержит все необходимые для реализаций алгоритмов инструменты.

3.2 Реализация алгоритмов

Реализации алгоритмов представлены в листингах 3.1–3.5.

Листинг 3.1 – Реализация алгоритма полного перебора

```
def brute_force_search(graph):
    n = len(graph)
    best_path_length = float('inf')
    best_path = None
    perms = permutations(list(range(n)))
    for perm in perms:
        path_length = 0
        for i in range(n - 1):
            path_length += graph[perm[i]][perm[i + 1]]
        path_length += graph[perm[-1]][perm[0]]
        if path_length < best_path_length:
            best_path_length = path_length
            best_path = perm
    best_path = list(best_path)
    best_path.append(best_path[0])
    return best_path, best_path_length
```

Листинг 3.2 – Основная функция муравьиного алгоритма

```
def calculate(self):
    best_path = None
    best_path_length = float('inf')
    for t in range(self.t_max):
        all_paths = []
        for i in range(self.n_ants):
            path, length = self.get_path(i)
            all_paths.append((path, length))
        for path, length in all_paths:
            if length < best_path_length:
                best_path_length = length
                best_path = path
```

```

        self.update_pheros(all_paths, best_path, best_path_length)
    return best_path, best_path_length

```

Листинг 3.3 – Генерация пути одного муравья

```

def get_path(self, ant_number):
    visited = set()
    path = []
    current_node = ant_number
    visited.add(current_node)
    path.append(current_node)
    length = 0
    while len(visited) < self.n_nodes:
        next_node = self.choose_next_node(current_node, visited)
        path.append(next_node)
        length += self.graph[current_node][next_node]
        visited.add(next_node)
        current_node = next_node
    length += self.graph[path[-1]][path[0]]
    path.append(path[0])
    return path, length

```

Листинг 3.4 – Выбор следующей вершины, в которую будет совершен переход

```

def choose_next_node(self, visited, current_node):
    probabilities = []
    total = 0
    for next_node in range(self.n_nodes):
        if next_node not in visited:
            pher = self.pher[current_node][next_node] **
                  self.herd_coef
            visibility = (1 /
                          self.graph[current_node][next_node]) **
                          self.greed_coef
            value = pher * visibility
            total += value
            probabilities.append((next_node, value))
    if total == 0:
        probabilities = [(node, 1 / len(probabilities)) for
                         node, _ in probabilities]
    else:
        probabilities = [(node, value / total) for node, value
                         in probabilities]
    r = random.random()

```

```

cumulative = 0
for node, prob in probabilities:
    cumulative += prob
    if r <= cumulative:
        return node
return probabilities[-1][0]

```

Листинг 3.5 – Обновление матрицы феромонов

```

def update_pher(self, paths, best_path, best_path_length):
    self.pher *= (1 - self.evap_rate) # Испарение феромона
    for path, length in paths:
        for i in range(len(path) - 1):
            from_node = path[i]
            to_node = path[i + 1]
            if length > 0:
                self.pher[from_node][to_node] += self.pher_sum /
                    length
    if best_path is not None:
        for i in range(len(best_path) - 1):
            from_node = best_path[i]
            to_node = best_path[i + 1]
            if best_path_length > 0:
                self.pher[from_node][to_node] += self.pher_sum /
                    best_path_length

```

3.3 Функциональные тесты

Функциональные тесты приведены в таблице 3.1 и были пройдены успешно всеми алгоритмами.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 1 & 4 & 16 \\ 1 & 0 & 9 & 25 \\ 4 & 9 & 0 & 36 \\ 16 & 25 & 36 & 0 \end{pmatrix}$	54, [0, 2, 1, 3, 0]	54, [0, 2, 1, 3, 0]
$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 0 & 4 \\ 5 & 6 & 0 \end{pmatrix}$	10, [0, 1, 2, 0]	10, [0, 1, 2, 0]
$\begin{pmatrix} 0 & 1 & 2 & 12 & 3 \\ 3 & 0 & 4 & 43 & 5 \\ 5 & 6 & 0 & 24 & 1 \\ 63 & 43 & 24 & 0 & 2 \\ 3 & 6 & 14 & 4 & 0 \end{pmatrix}$	29, [0, 3, 4, 1, 2, 0]	29, [0, 3, 4, 1, 2, 0]

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

- Процессор: Intel(R) Core(TM) i5-8250U CPU @ 1.60ГГц 1.80 ГГц.
- Оперативная память: 8 ГБайт.
- Операционная система: Windows 11 Home версии 23H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Временные характеристики

Для замеров времени были использованы функции библиотеки time [3].

Исследование произведено на графах с количеством вершин 1 до 10. Для каждого размера графа замеры проводились 50 раз. Средние значения полученных результатов представлены в таблице 4.1.

Таблица 4.1 – Среднее время выполнения алгоритмов, мкс

Кол-во вершин графа	Полный перебор	Муравьиный алгоритм
1	1	8
2	2	18
3	7	39
4	23	68
5	59	93
6	143	138
7	260	223
8	2959	436
9	14874	812
10	311094	1448

Таким образом, был сделан вывод, что на небольших значениях количества вершин (до 5 включительно) алгоритм полного перебора работает быстрее муравьиного, но на более больших количествах вершин скорость муравьиного алгоритма начинает значительно превышать скорость алгоритма полного перебора.

4.3 Параметризация муравьиного алгоритма

В ходе исследования муравьиного алгоритма изменялся входной параметр a — коэффициент жадности. Исследование проводилось на трех графах, каждый из которых состоял из 10 вершин. Для каждого графа и каждого набора параметров алгоритм запускался 20 раз, были вычислены максимальные, средние и медианные значения отклонения длины найденного маршрута от эталонной длины. Результаты исследования представлены в Приложении А. Из полученных данных следует, что с увеличением коэффициента жадности наблюдается снижение максимальных, минимальных и средних отклонений. Это указывает на то, что точность муравьиного алгоритма возрастает при увеличении коэффициента жадности.

Таким образом рекомендованные значения параметров:

- коэффициент жадности = 0.9;
- коэффициент стадности = 0.1;
- количество итераций = 500.

ЗАКЛЮЧЕНИЕ

В результате исследования сделан вывод, что на малых размерах графов для решения задачи коммивояжера допустимо использования алгоритма полного перебора, но на графах с большим количеством вершин муравьиный алгоритм работает значительно быстрее. Но для корректной работы муравьиного алгоритма необходимо подобрать правильные параметры, в противном случае найденный маршрут может оказаться не оптимальным.

Цель работы достигнута. В ходе выполнения данной лабораторной работы были решены следующие задачи:

- 1) реализация алгоритмов решения задачи коммивояжера — полного перебора и муравьиного алгоритма;
- 2) исследование ресурсной эффективности реализованных алгоритмов для разных параметров;
- 3) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчёто–пояснительная записка к работе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Борознов В. О.* Исследование решения задачи коммивояжера. — АГТУ, Вестник Астраханского государственного технического университета. [Электронный ресурс]. — Режим доступа: <https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera/viewer> (дата обращения: 20.12.2024).
2. Welcome to Python [Электронный ресурс]. — Режим доступа: <https://www.python.org> (дата обращения: 20.12.2024).
3. Документация модуля time [Электронный ресурс]. — Режим доступа: <https://docs.python.org/3/library/time.html#functions> (дата обращения: 20.12.2024).

ПРИЛОЖЕНИЕ А