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»

Лабораторная работа №6
по дисциплине «Анализ алгоритмов»

Тема Задача коммивояжёра

Студент Тузов Даниил Александрович

Группа ИУ7-52Б

Преподаватель Строганов Дмитрий Владимирович

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В 6 лабораторной работе рассматриваются методы решения задачи коммивояжёра.

Целью работы является разработка ПО, решающего задачу коммивояжёра. Для достижения поставленной цели необходимо решить следующие задачи:

— сформулировать задачу коммивояжёра;
— рассмотреть метод решения задачи с помощью алгоритма полного перебором;
— рассмотреть метод решения задачи с помощью муравьиного алгоритма;
— реализовать алгоритмы;
— сравнить время выполнения влгоритмов;
— выполнить параметризацию;
— обосновать полученные результаты.

1 Аналитическая часть

В этой части сформулировано условие задачи коммивояжёра, а также приведены теоретические аспекты алгоритмов полного перебора и муравьиного алгоритма для решения задачи коммивояжёра.

1.1 Задачи коммивояжера

Пусть задан граф $G = (V, E)$, где V – множество вершин ($|V| = n$), а E – множество ребер ($|E| = m$). Каждое ребро $(i, j) \in E$ имеет длину c_{ij}, которая задается матрицей расстояний $C = \|c_{ij}\|$. Если между вершинами i и j нет ребра, соответствующий элемент матрицы считается равным бесконечности ($c_{ij} = \infty$) [1].

Требуется найти такой путь в графе, который содержит все вершины и при этом его длина минимальна.

1.2 Алгоритм полного перебора

Один из возможных алгоритмов решения задачи коммивояжёра состоит в том, чтобы перебрать все возможные перестановки номеров вершин в графе. При этом для каждой перестановки считается длина пути. Среди найденных длин выбирается минимальная. Сложность такого алгоритма $O(N!)$, где N – количество вершин в графе [1].

1.3 Муравьиный алгоритм

Муравьиный алгоритм — это метод решения задачи коммивояжера, основанный на моделировании поведения муравьиной колонии. Каждый муравей прокладывает маршрут, используя информацию о феромонах, оставленных другими муравьями на графе. В процессе движения муравей оставляет феромон на своем пути, чтобы другие могли ориентироваться на него. Постепенно феромоны на оптимальном маршруте накапливаются, так как он используется наиболее часто. [2]

Характеристики муравья:

— зрение – муравей способен оценивать длину ребер;

— память – запоминает посещенные вершины;

— обоняние – реагирует на феромоны, оставленные другими муравьями.

Для оценки привлекательности перехода используется функция (1.1):

$$\eta_{ij} = \frac{1}{D_{ij}}, \tag{1.1}$$

где D_{ij} — расстояние между вершинами i и j.

Вероятность перехода муравья k из текущей вершины i в вершину j рассчитывается по формуле (1.2):

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q \in J_{ik}} \tau_{iq}^a \eta_{iq}^b}, & \text{если вершина } j \text{ еще не посещена муравьем } k, \\ 0, & \text{иначе,} \end{cases} \tag{1.2}$$

где:

— a — параметр влияния феромона;
— b — параметр влияния длины пути;
— τ_{ij} — количество феромонов на ребре (i, j);
— η_{ij} — видимость (обратная расстоянию).

В конце дня уровень феромонов на ребрах обновляется по формуле (1.3):

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}, \tag{1.3}$$

где p — коэффициент испарения феромона, а $\Delta\tau_{ij}$ определяется как:

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij}^k, \tag{1.4}$$

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{если ребро } (i, j) \text{ посещено муравьем } k, \\ 0, & \text{иначе,} \end{cases} \tag{1.5}$$

где Q — параметр, связанный с длиной оптимального пути, а L_k — длина маршрута муравья k.

1.4 Вывод

В этой части была сформулирована задача коммивояжёра и рассмотрены теоретические аспекты ее решения.

2 Конструкторская часть

В этой части представлены описания алгоритмов.

2.1 Описание алгоритма полного перебора

На вход алгоритму подается матрица `mat` на выходе длина минимального пути и сам минимальный путь.

На рисунке 2.1 приведено описание алгоритма полного перебора.

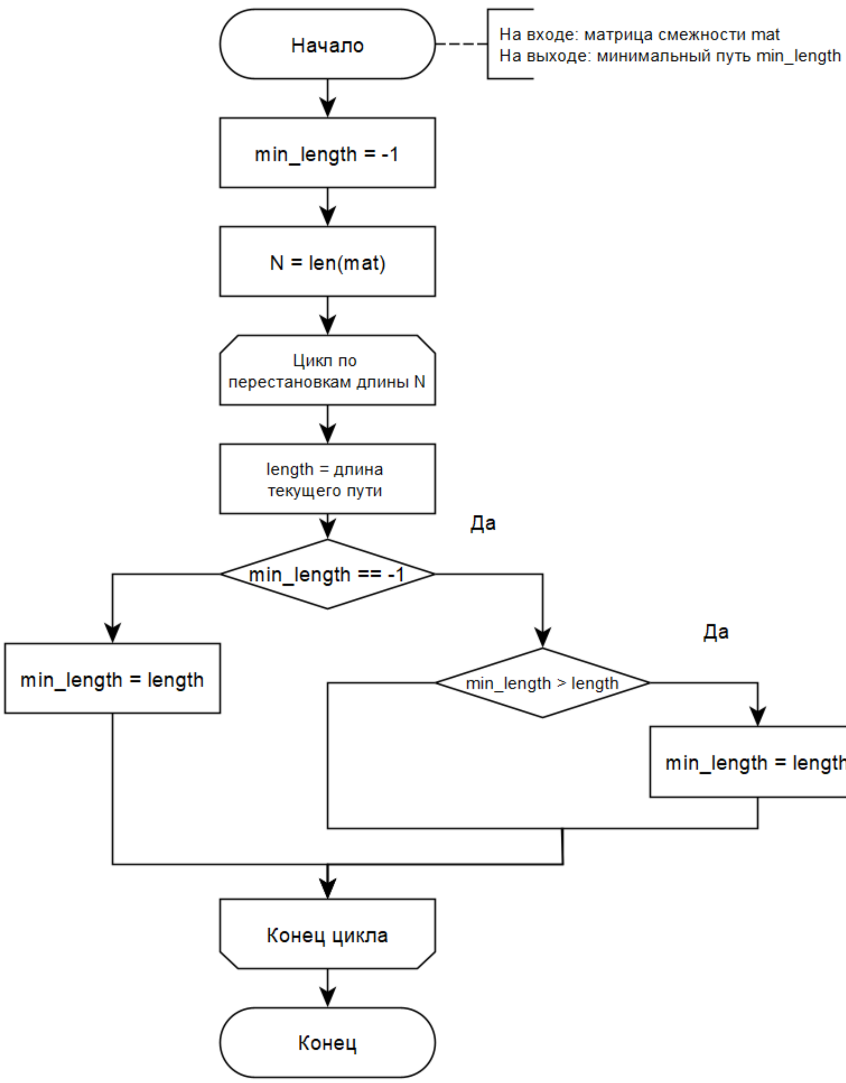

Рисунок 2.1 — Описание алгоритма полного перебора

2.2 Описание муравьиного алгоритма

На вход алгоритму подается матрица `mat` и коэффициенты α, ρ и t_{max} на выходе длина минимального пути и сам минимальный путь.

На рисунке 2.1 приведено описание муравьиного алгоритма.

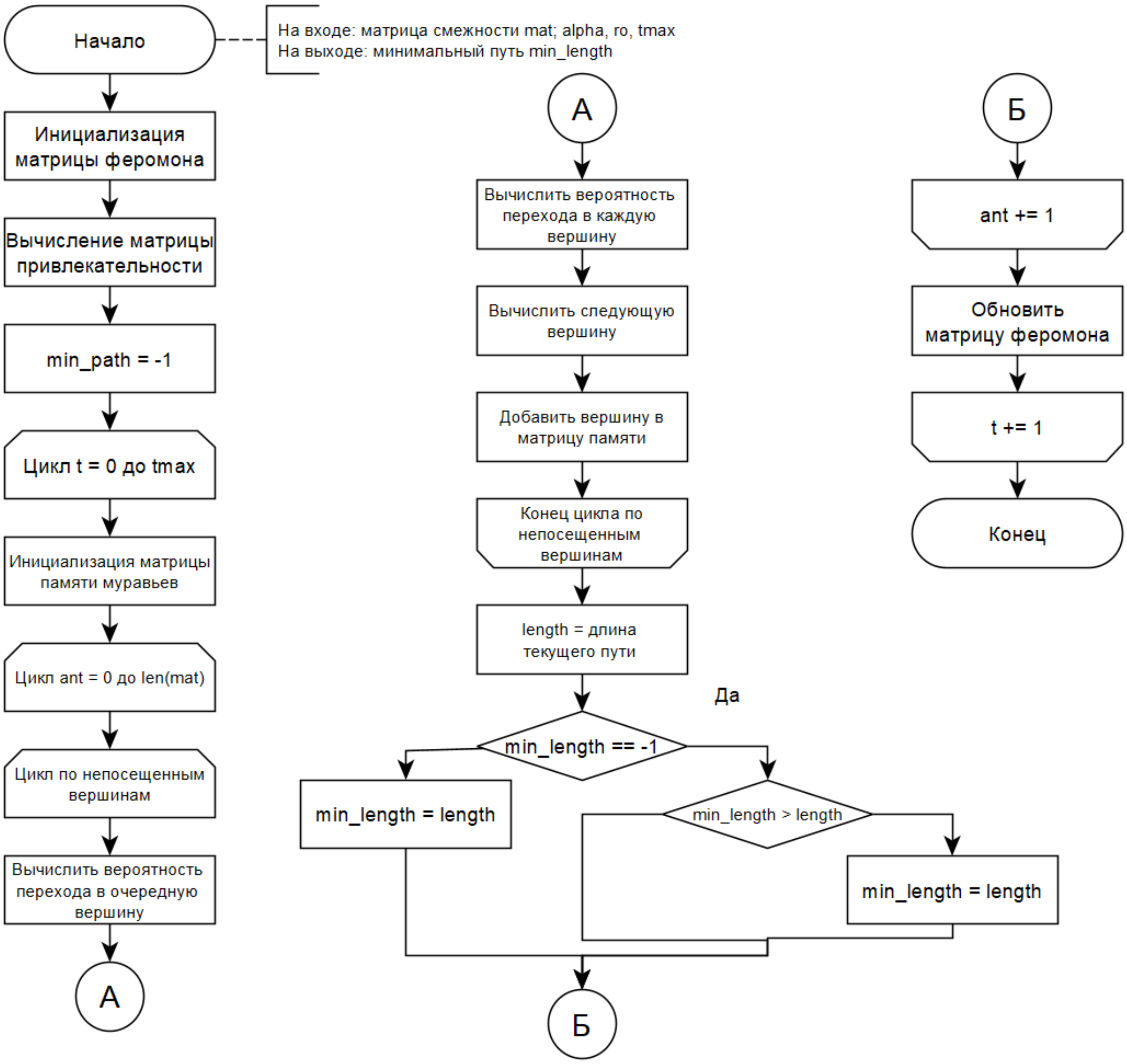

Рисунок 2.2 — Описание муравьиного алгоритма

2.3 Вывод

В этом разделе на основе теоретических аспектов были представлены описания алгоритма полного перебора и муравьиного алгоритма.

3 Технологическая часть

В этом разделе обоснованы средства реализации, а так же представлены реализации алгоритмов и функциональные тесты.

3.1 Средства реализации

Для реализации алгоритмов в этой лабораторной работы был выбран язык $Python$ [3], потому что в нем нет автоматического сборщика мусора.

3.2 Реализация алгоритмов

В листингах 3.1—3.2 представлены коды написанных алгоритмов.

Листинг 3.1 — Алгоритм полного перебора

```
def standard_alg(mat):
    n = len(mat)
    vertexes = [ i + 1 for i in range(n) ]
    min_path_len = -1
    best_path = tuple()
    for path in permutations(vertexes):
        path_len = 0
        for i in range(len(path) - 1):
            a, b = path[i], path[i + 1]
            path_len += mat[a - 1][b - 1]
        if min_path_len == -1 or min_path_len > path_len:
            min_path_len = path_len
            best_path = path
    return list(best_path), min_path_len
```

Листинг 3.2 — Муравьиный алгоритм

```
def ant_alg(mat, alpha, ro, t_max):
    n, q = len(mat), calc_q(mat)
    pheromone = init_pheromone(n)
    attract = init_attract(mat)
    min_path_len = -1
    best_path = list()
    for t in range(t_max):
        memory = init_memory(n)
        for ant in range(n):
            while len(memory[ant]) != n:
                p = calc_p(pheromone, attract, memory[ant], n, alpha)
                memory[ant].append(calc_next(p))
            path_len = calc_length(mat, memory[ant])
            if min_path_len == -1 or min_path_len > path_len:
                min_path_len = path_len
```

```
                best_path = memory[ant]
        pheromone = update_pheromone(mat, memory, pheromone, q, ro)
    for i in range(len(best_path)):
        best_path[i] += 1
    return best_path, min_path_len

def calc_q(mat):
    n, q = len(mat), 0
    for i in range(n):
        for j in range(n):
            if i != j:
                q += mat[i][j]
    return q / (n ** 3 - n)

def init_pheromone(n):
    return [[1 for i in range(n)] for j in range(n)]

def init_attract(mat):
    return [[(1 / mat[i][j] if i != j else 0) for i in range(len(mat))] for
        j in range(len(mat))]

def init_memory(n):
    memory = list()
    for i in range(n):
        memory.append([i])
    return memory

def calc_p(pheromon, attract, memory, n, alpha):
    beta = 1 - alpha
    p = [0] * n
    for new_pos in range(n):
        if new_pos in memory:
            p[new_pos] = 0
        else:
            last_pos = memory[-1]
            p[new_pos] = pheromon[last_pos][new_pos] ** beta * attract[
                last_pos][new_pos] ** alpha
    psum = sum(p)
    for i in range(n):
        p[i] /= psum
    return p

def calc_next(p):
    point = random()
    i = 0
    while i < len(p):
        if point - p[i] < 0:
```

```
            return i
        point -= p[i]
        i += 1
    return len(p)

def calc_length(mat, path):
    length = 0
    for i in range(len(path) - 1):
        a, b = path[i], path[i + 1]
        length += mat[a][b]
    return length

def update_pheromone(mat, memory, pheromone, q, ro):
    n = len(mat)
    for i in range(n):
        for j in range(n):
            delta = 0
            for ant in range(n):
                length = calc_length(mat, memory[ant])
                delta += q / length
            pheromone[i][j] = pheromone[i][j] * (1 - ro) + delta
            if pheromone[i][j] < 0.01:
                pheromone[i][j] = 0.01
    return pheromone
```

3.3 Функциональные тесты

В таблице 3.1 представлены результаты тестирования программы для алгоритма полного перебора.

Таблица 3.1 — Функциональные тесты

Матрица смежности	Ожидаемый результат	Статус
$\begin{pmatrix} 0 & 16 & 10 & 14 & 4 & 12 & 14 & 20 \\ 18 & 0 & 6 & 2 & 13 & 6 & 18 & 13 \\ 14 & 16 & 0 & 10 & 18 & 9 & 1 & 15 \\ 11 & 17 & 3 & 0 & 15 & 3 & 2 & 4 \\ 2 & 14 & 12 & 5 & 0 & 12 & 3 & 14 \\ 5 & 1 & 9 & 10 & 20 & 0 & 1 & 15 \\ 11 & 8 & 16 & 10 & 15 & 13 & 0 & 18 \\ 11 & 20 & 3 & 2 & 12 & 13 & 20 & 0 \end{pmatrix}$	[5, 1, 6, 2, 4, 8, 3, 7], 25	OK
$\begin{pmatrix} 0 & 20 & 7 & 6 & 17 & 9 & 20 & 20 & 14 \\ 8 & 0 & 6 & 12 & 1 & 2 & 12 & 20 & 18 \\ 19 & 7 & 0 & 12 & 19 & 5 & 20 & 7 & 14 \\ 2 & 9 & 4 & 0 & 13 & 10 & 14 & 3 & 20 \\ 13 & 20 & 1 & 10 & 0 & 17 & 8 & 5 & 20 \\ 9 & 13 & 8 & 16 & 6 & 0 & 7 & 13 & 19 \\ 15 & 4 & 7 & 3 & 17 & 1 & 0 & 14 & 5 \\ 19 & 1 & 9 & 20 & 1 & 6 & 15 & 0 & 17 \\ 17 & 19 & 9 & 15 & 4 & 6 & 16 & 4 & 0 \end{pmatrix}$	[9, 8, 2, 5, 3, 6, 7, 4, 1], 24	OK
$\begin{pmatrix} 0 & 16 & 11 & 8 & 10 & 4 & 12 & 4 & 9 & 7 \\ 4 & 0 & 12 & 5 & 12 & 13 & 19 & 2 & 17 & 14 \\ 1 & 2 & 0 & 9 & 18 & 15 & 16 & 19 & 19 & 2 \\ 5 & 19 & 17 & 0 & 18 & 8 & 7 & 7 & 4 & 1 \\ 12 & 17 & 2 & 20 & 0 & 13 & 20 & 20 & 6 & 11 \\ 16 & 4 & 8 & 16 & 5 & 0 & 3 & 17 & 2 & 17 \\ 9 & 12 & 9 & 7 & 11 & 14 & 0 & 7 & 19 & 6 \\ 11 & 20 & 5 & 9 & 10 & 19 & 19 & 0 & 13 & 17 \\ 19 & 16 & 12 & 19 & 16 & 13 & 9 & 4 & 0 & 13 \\ 13 & 8 & 4 & 8 & 20 & 7 & 12 & 14 & 12 & 0 \end{pmatrix}$	[5, 3, 1, 6, 9, 7, 4, 10, 2, 8], 36	OK

3.4 Вывод

В этом разделе на основе описаний алгоритмов были написаны и представлены коды алгоритмов. Помимо кодов представлены функциональные тесты, на которых был протестирован алгоритм полного перебора.

4 Исследовательская часть

В этом разделе приведен график со сравнением времени работы алгоритмов на графах с разным количеством вершин, а так же представлен результат параметризации муравьиного алгоритма.

4.1 Технические характеристики ЭВМ

Все замеры проводились на ЭВМ, характеристики которой приведены ниже:
— процессор – 12th Gen Intel(R) Core(TM) i5-12450H 2.00 ГГц;
— оперативная память – 16,0 ГБ;
— тип системы – 64-разрядная операционная система, процессор x64;
— операционная система – Windows 11;
— версия ОС – 23H2;
— 12 логических ядер.

4.2 Сравнение времени работы

Время выполнения работы алгоритмов представлено в секундах. Проводилось 10 замеров. Результаты представлены в таблице 4.1 и на рисунке 4.1.

Таблица 4.1 — Сравнение алгоритмов по времени выполнения

Количество вершин	Алгоритм полного перебора	Муравьиный алгоритм
2	0	0.002
3	0	0.002
4	0	0.008
5	0	0.013
6	0	0.023
7	0	0.039
8	0.02	0.059
9	0.15	0.086
10	1.46	0.098

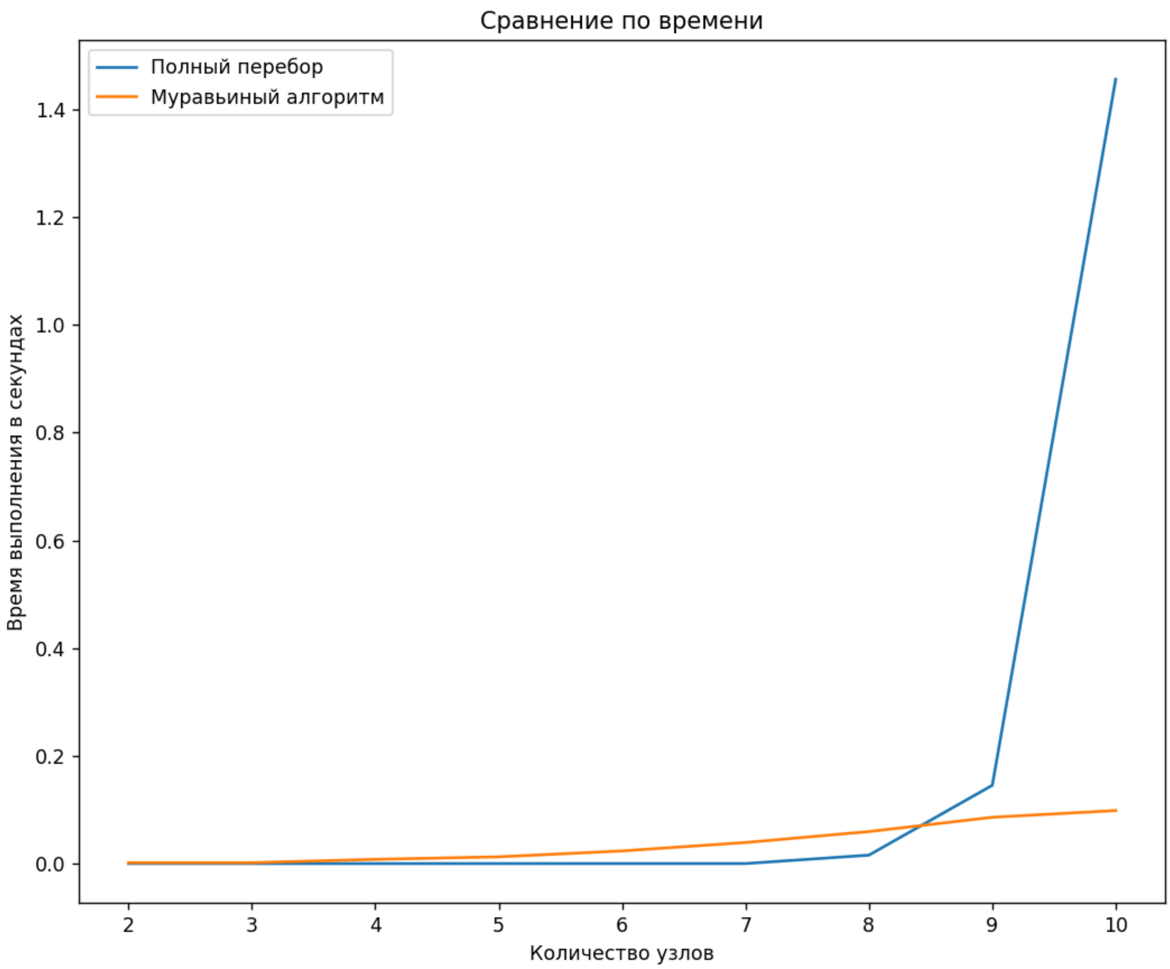

Рисунок 4.1 — Сравнение алгоритмов по времени выполнения

4.3 Параметризация муравьиного алгоритма

Параметризация выполнялась для 3 графов, представленных в технологической части в разделе с функциональными тестами. Разультат параметризации представлен в приложении А.

4.4 Вывод

В результате сравнения алгоритмов по времени сделан вывод, что муравьиный алгоритм проигрывает для графа с количеством вершин меньшим 9. Начиная с количества узлов равным 9, муравьиный алгоритм превосходит алгоритм полного перебора. Наилучшими параметрами параметризации являются $\alpha = 0.75$, $\rho = 0.5$, $t_{max} = 2000$.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы поставленная цель была достигнута. Были решены все задачи:

— сформулирована задача коммивояжёра;
— рассмотрены методы решения задачи коммивояжера;
— реализованы алгоритмы;
— проведено сравнение алгоритмов по времени;
— выполнена параметризация для муравьиного алгоритма;
— обоснованы полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Меламед, Игорь Ильич and Сергеев, С И and Сигал, Израиль Хаимович. Задача коммивояжера. Вопросы теории. — Автоматика и телемеханика, 1989, с.3-33 (дата обращения: 09.12.2024)

2. Штовба С.Д. Муравьиные алгоритмы. — Exponenta Pro. Математика в приложениях, 2003, с.70-75 (дата обращения: 09.12.2024)

3. Язык Python / [Электронный ресурс] // Режим доступа: https://docs.python.org/3/index.html (дата обращения: 09.12.2024)

4. Библиотека random / [Электронный ресурс] // Режим доступа: https:// docs.python.org/3/library/ random.html (дата обращения: 09.12.2024)

5. Библиотека time / [Электронный ресурс] // Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 09.12.2024)

6. Библиотека matplotlib / [Электронный ресурс] // Режим доступа: https:// matplotlib.org (дата обращения: 09.12.2024)

ПРИЛОЖЕНИЕ А