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"

Тема Муравьиный алгоритм

Студент Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

В современной теории алгоритмов есть задачи, которые возможно решить только полным перебором всех вариантов. Но для многих задач невозможно вычислить решение за полиномиальное время. Поэтому существуют алгоритмы, которые позволяют найти решение, близкое к идеальному, за достижимое время.

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближенных решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Цель лабораторной работы – разработать ПО, которое решает задачу коммивояжёра двумя способами: полным перебором и муравьиным алгоритмом.

Для достижения поставленной цели необходимо выполнить следующее:

- рассмотреть формализацию задачи коммивояжёра;
- рассмотреть муравьиный алгоритм;
- привести схемы реализации алгоритмов;
- определить средства программной реализации;
- реализовать два алгоритма для решения задачи;
- протестировать разработанное ПО;
- оценить реализацию алгоритмов по времени и памяти.

1 Аналитический раздел

В этом разделе будут рассмотрены задача о коммивояжёре и муравьиный алгоритм для её решения.

1.1 Задача о коммивояжёре

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через все указанные города по одному разу с последующим возвратом в исходный город. Маршрут должен проходить через каждый город только один раз.[1]

Задача коммивояжёра относится к числу трансвычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

1.2 Муравьиный алгоритм

Идея муравьиного алгоритма – моделирование поведения муравьев, связанное с их способностью быстро находить кратчайшие пути и адаптироваться к изменяющимся условиям, т.е. искать новые кратчайшие пути [2]. При своём движении муравей метит свой путь феромоном, и эта информация используется прочими для выбора пути. Таким образом, более короткие пути будут сильнее обогащаться феромоном и станут более предпочтительны для всей колонии. С помощью моделирования испарение феромона, т.е. отрицательной обратной связи, гарантируется, что найденное локально оптимальное решение не будет единственным – будут предприняты попытки поиска других путей.

Опишем правила поведения муравья применительно к решению задачи коммивояжера [3]:

- муравьи имеют 'память' - запоминают уже посещенные города, чтобы избегать повторений: обозначим через $J_{i,k}$ список городов, которые

необходимо пройти муравью k, находящемуся в городе i;

- муравьи обладают 'зрением' - видимость есть эвристическое желание посетить город j, если муравей находится в городе i.

  Уместно считать видимость обратно пропорциональной расстоянию между соответствующими городами $\eta_{i,j} = 1/D_{i,j}$;

- муравьи обладают 'обонянием' – могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьев.

  Обозначим количество феромона на ребре (i, j) в момент времени t через $\tau_{i,j}(t)$.

Вероятность перехода из вершины i в вершину j определяется по формуле 1.1

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где $\tau_{i,j}-$ количество феромонов на ребре i до j; $\eta_{i,j}-$ эвристическое расстояние от i до j; $\alpha-$ параметр влияния феромона; $\beta-$ параметр влияния расстояния.

Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора.

Пусть $T_k(t)$ есть маршрут, пройденный муравьём k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано по формуле 1.2:

$$\Delta\tau_{i,j}^{k} = \begin{cases} Q/L_k & \text{Если k-ый муравей прошел по ребру ij;} \\ 0 & \text{Иначе} \end{cases} \tag{1.2}$$

где Q - количество феромона, переносимого муравьём;
Тогда:

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \tag{1.3}$$

1.3 Вывод

Была рассмотрена задача о коммивояжёра и проанализировано применение муравьиного алгоритма для решения этой задачи.

2 Конструкторский раздел

В этом разделе будут приведены требования к ПО, схемы реализации алгоритмов, а также выбранные классы эквивалентности для тестирования ПО.

2.1 Требования к ПО

Ниже будет представлен список требований к разрабатываемому программному обеспечению.

Требования к входным данным:

- на вход подаётся квадратная матрица расстояний, состоящая из целых чисел;
- в матрице как минимум две строки;
- матрица может быть не симметричной.

Требования к выводу:

- программа должна вывести минимальный путь и соответствующее ему минимальное расстояние.

2.2 Схемы алгоритмов

На рисунке 2.1 будет приведена схема реализации алгоритма полного перебора.

На рисунках 2.2-2.3 будет приведена схема реализации муравьиного алгоритма.

Рисунок. 2.1: Схема реализации алгоритма полного перебора

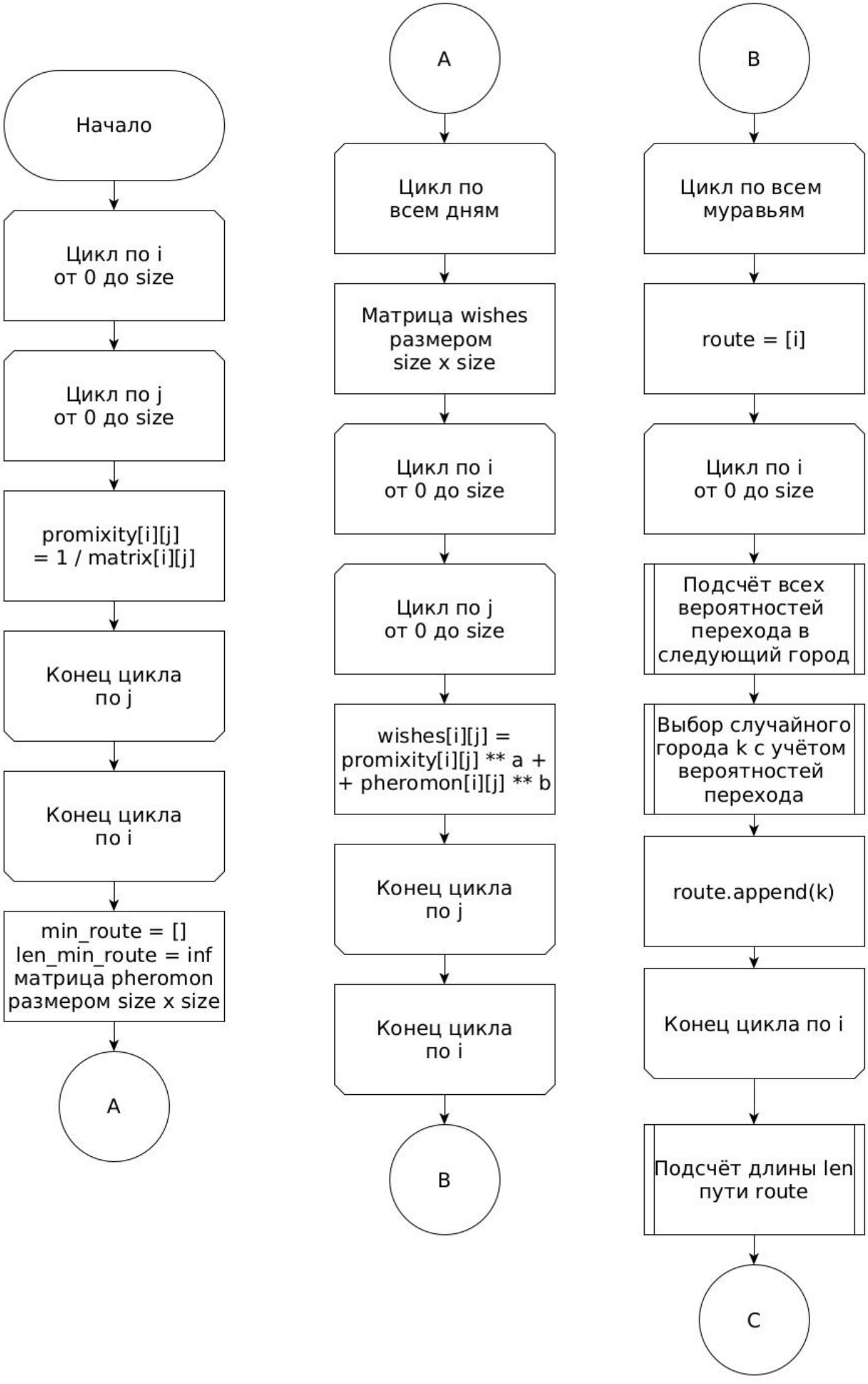

Рисунок. 2.2: Схема реализации муравьиного алгоритма (часть 1)

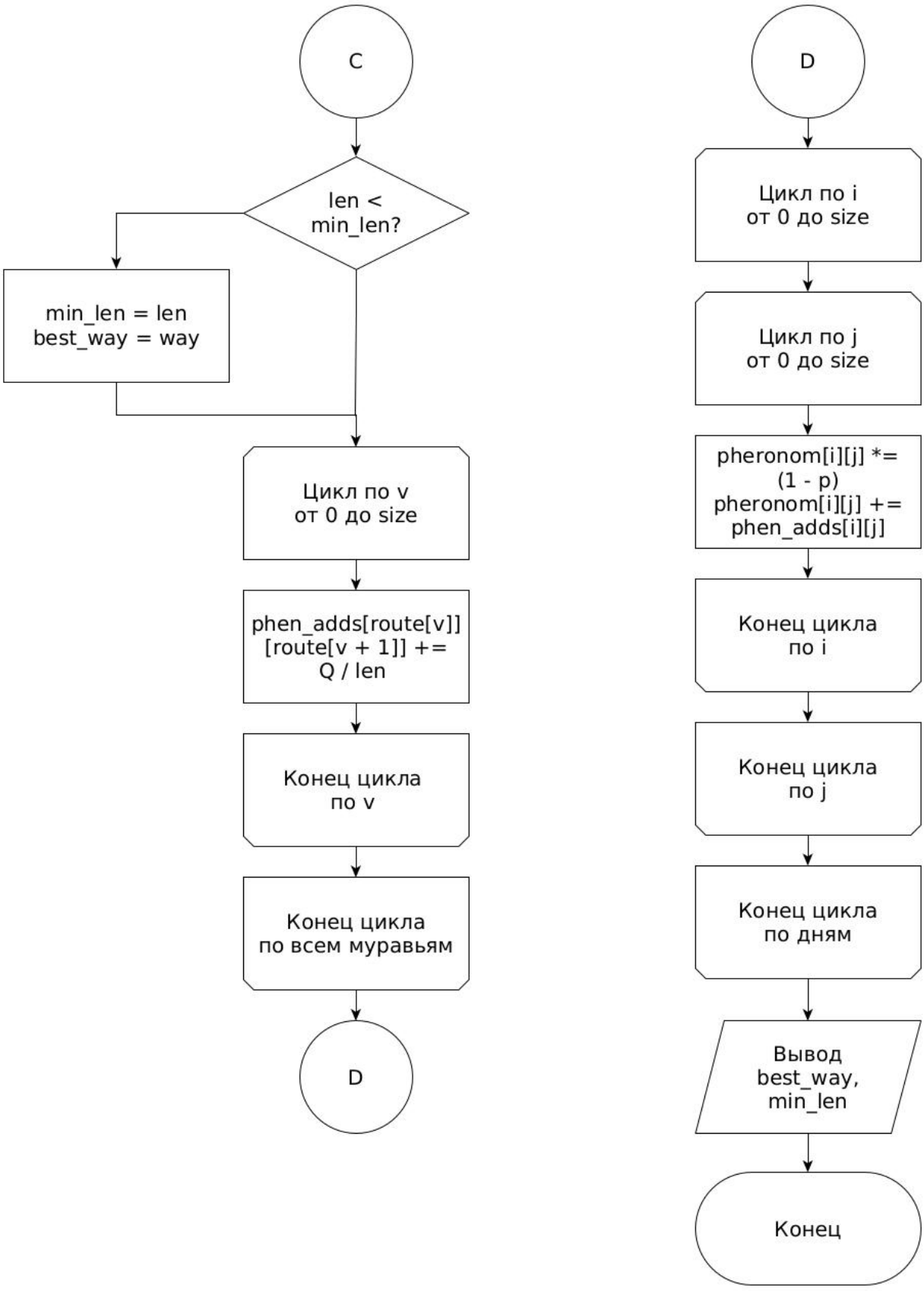

Рисунок. 2.3: Схема реализации муравьиного алгоритма (часть 2)

2.3 Типы данных для алгоритмов

Тестирование алгоритма будет производиться на матрице из целых чисел. Тем не менее, алгоритмы универсальны, и для их использования подходят любые вещественные типы.

Размер массива может быть произвольным из тех, что допустимы по требованиям ко вводу.

2.4 Способ тестирования

Тестирование программы будет производиться методом чёрного ящика. Такой подход выбран, так как от реализаций алгоритмов требуется в первую очередь правильность работы. Сама по себе реализация не требует тестировки, так как в точности повторяет теоретические принципы, сформированные в аналитическом разделе.

В качестве классов эквивалентности были выбраны следующие сущности:

- матрица размером 2x2;
- матрица размером 5x5;
- несимметричная матрица размером 5x5;
- матрица, состоящая из одинаковых чисел.

2.5 Вывод

Были приведены требования к ПО, схемы реализации алгоритмов. Был определен способ тестирования алгоритмов.

3 Технологическая часть

В этом разделе будут приведены листинги кода и результаты функционального тестирования.

3.1 Средства реализации программного обеспечения

В качестве языка программирования выбран Python 3.9, так как имеется опыт разработки проектов на этом языке. Для замера процессорного времени используется функция process_time_ns из библиотеки time. В её результат не включается время, когда процессор не выполняет задачу [4].

3.2 Листинг кода

На листинге 3.1 представлена реализация алгоритма полного перебора. На листинге 3.2 представлена реализация муравьиного алгоритма.

Листинг 3.1: Реализация алгоритма полного перебора

```
# Полный перебор
def full_research(matrix, size):

    all_combinations = permutations(range(size))
    min_dist = float("inf")
    best_way = []
    for combination in all_combinations:
        combination = list(combination)
        combination.append(combination[0])
        sum = 0
        for i in range(size):
            sum += matrix[combination[i]][combination[i + 1]]
        if (sum < min_dist):
            min_dist = sum
            best_way = combination

    print("Полный перебор: ")
    print("Минимальный путь: ", best_way)
```

Листинг 3.2: Реализация муравьиного алгоритма

```
    print("Его длина: ", min_dist)

def create_promixity_table(matrix, size):
    promixity_table = [[0] * size for i in range(size)]
    for i in range(size):
        for j in range(size):
            if (matrix[i][j] != 0):
                promixity_table[i][j] = 1 / matrix[i][j]
            else:
                promixity_table[i][j] = 0
    return promixity_table

def count_wishes_table(promixity_table, size, a, b, pheromon_table):
    wishes_table =  [[0] * size for i in range(size)]

    for i in range(size):
        for j in range(size):
            wishes_table[i][j] = pow(promixity_table[j][i], a) * pow(
               pheromon_table[j][i], b)

    return wishes_table


def count_all_possibity_ant(wishes_table, current_city, size, root):
    result_possibilities = []

    for i in range(size):
        if i in root:
            result_possibilities.append(0)
        else:
            current_prob = wishes_table[current_city][i]
            result_possibilities.append(current_prob)

    sum_probs = sum(result_possibilities)

    if (sum_probs != 0):
        for i in range(size):
            result_possibilities[i] /= sum_probs

    return result_possibilities

def count_way(matrix, root):
    N = len(root) - 1
    sum_way = 0

    for i in range(N):
        sum_way += matrix[root[i]][root[i + 1]]
```

```
    return sum_way


def ant_iteration(promixity_table, size, a, b, q, p, pheromon_table,
   min_route, len_min_route, matrix):

    wishes_table = count_wishes_table(promixity_table, size, a, b,
       pheromon_table)
    pheromon_adds = [[0] * size for i in range(size)]

    for i in range(size):
        route = [i]
        for j in range(size - 1):
            temp_probs = count_all_possibity_ant(wishes_table, route[-1],
               size, route)

            prob = random()
            temp_sum = 0
            k = 0
            while k < size:
                if (temp_probs[k] != 0):
                    temp_sum += temp_probs[k]
                    if (temp_sum > prob):
                        break
                k += 1
            if (k == size):
                k = size - 1
            route.append(k)

        route.append(route[0])

        route_len = count_way(matrix, route)
        if (route_len < len_min_route):
            len_min_route = route_len
            min_route = route

        for v in range(size):
            pheromon_adds[route[v]][route[v + 1]] += q / route_len

    for i in range(size):
        for j in range(size):
            pheromon_table[i][j] *= (1 - p)
            pheromon_table[i][j] += pheromon_adds[i][j]

    return pheromon_table, min_route, len_min_route

def ant(matrix, size, a, b, q, p, time):
```

```
    promixity_table = create_promixity_table(matrix, size)
    pheromon_table = [[0.2] * size for i in range(size)]
    min_route = []
    len_min_route = float("inf")

    for i in range(time):
        pheromon_table, min_route, len_min_route = ant_iteration(
            promixity_table, size, a, b, q, p, pheromon_table, min_route,
            len_min_route, matrix)
```

3.3 Функциональные тесты

В таблице 3.1 приведены результаты функциональных тестов. В первом столбце – исходная матрица расстояний. Во втором столбце – ожидаемый результат: кратчайший путь В третьем столбце – его длина.

Таблица 3.1: Функциональные тесты

Матрица	Кратчайший путь	Длина
0 1 1 0	[0, 1, 0]	2
0 5 2 3 4 5 0 7 1 2 2 7 0 9 5 3 1 9 0 2 4 2 5 2 0	[0, 2, 4, 1, 3, 0]	13
0 7 2 9 4 2 0 7 1 3 2 7 0 9 5 1 1 2 0 2 4 7 5 2 0	[0, 2, 4, 3, 1, 0]	12
0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0	[0, 1, 2, 3, 4, 0]	5

Все тесты алгоритмами были пройдены успешно.

3.4 Вывод

Были приведены листинги кода. Были проведены функциональные тесты. Все алгоритмы справились с тестированием.

4 Исследовательская часть

В этом разделе будет продемонстрирована работа программы, а также приведены результаты тестирования алгоритмов.

4.1 Демонстрация работы программы

На рисунке 4.1 приведена демонстрация работы программы.

```
Введите файл, в котором лежит матрица расстояний между городами: tes
--------------------------------------------------------------------
Исходная матрица:
0 7 2 9 4
2 0 7 1 3
2 7 0 9 5
1 1 2 0 2
4 7 5 2 0
--------------------------------------------------------------------
Полный перебор:
Минимальный путь:  [0, 2, 4, 3, 1, 0]
Его длина:  12
--------------------------------------------------------------------
Введите параметр alpha для муравьиного алгоритма: 1
Введите параметр beta для муравьиного алгоритма: 1
Введите параметр Q для муравьиного алгоритма: 12
Введите параметр p для муравьиного алгоритма: 0.5
Введите максимальное число итераций для муравьиного алгоритма: 1000
--------------------------------------------------------------------
Муравьиный алгоритм:
Минимальный путь:  [4, 3, 1, 0, 2, 4]
Его длина:  12
```

Рисунок. 4.1: Демонстрация программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

- операционная система: Ubuntu 20.04.1 LTS;
- память: 8 GB;
- процессор: Intel Core i5-1135G7 @ 2.40GHz [5].
- количество ядер процессора: 8

Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.3 Тестирование программы

Тестирование будет производиться в два этапа.

На первом этапе будет измерено время работы алгоритмов в зависимости от размера матрицы. Параметры для муравьиного алгоритма стандартные: коэффициенты alpha и beta равны 1, Q = 10, а p = 0.5. Число итераций муравьиного алгоритма – 1000.

На втором этапе будет протестирована работа муравьиного алгоритма в зависимости от параметров. Будет проанализирован конечный ответ и итоговая ошибка алгоритма. Будет использована матрица 15x15.

В таблице 4.1 представлены результаты первого этапа тестирования. Все алгоритмы выдавали один и тот же результат при всех размерах матрицы.

Таблица 4.1: Результаты тестов

Размер матрицы	Перебор	Муравьиный алгоритм
2	97774	26896807
3	56545	52780514
4	96257	84012718
5	174203	140044655
6	1493556	181273911
7	10373570	254475702
8	71372341	344602604
9	565912635	435430808
10	6325175130	619145862
11	75224132010	911105412

На рисунке 4.2 показан график зависимости времени работы алгоритмов от размера матрицы.

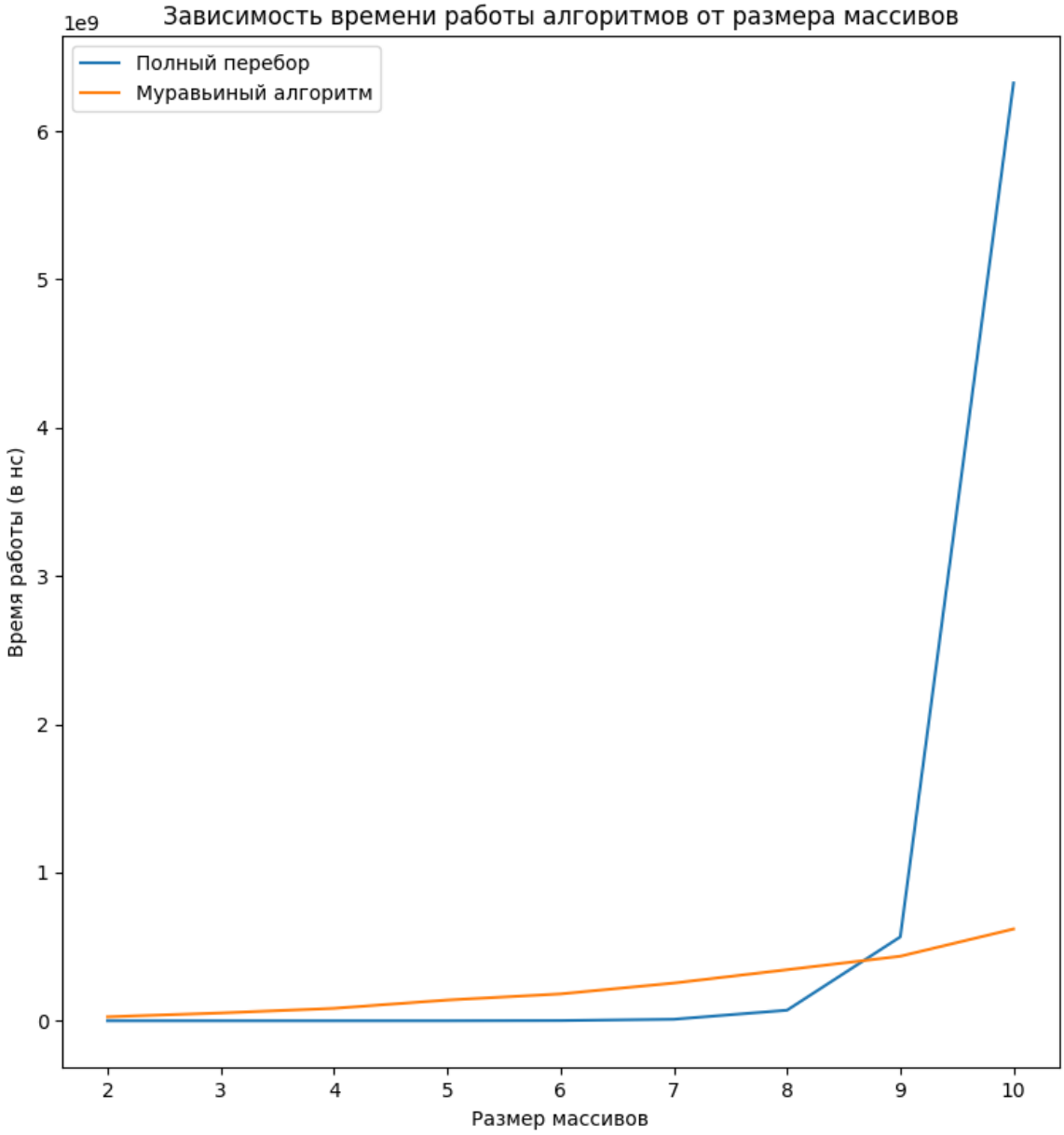

Рисунок. 4.2: График зависимости времени работы алгоритмов от размера матрицы

Результаты тестирования второго случая представлены в таблице 4.2. Полная таблица результатов представлена в приложении 1. В первых пяти столбцах – параметры муравьиного алгоритма, в шестом – выдаваемый результат, в седьмом – ошибка.

Таблица 4.2: Результаты тестов

α	β	Q	p	max_iter	Результат	Ошибка
1	1	1	0.5	10	804	212
1	1	1	0.5	1000	592	0
1	1	5	0.5	10	681	89
1	1	5	0.5	1000	592	0
1	2	1	0.5	10	705	113
1	2	1	0.5	1000	592	0
1	2	5	0.5	10	592	0
1	2	5	0.5	1000	705	113
1	3	1	0.5	10	592	0
1	3	1	0.5	1000	683	91
2	1	1	0.5	10	592	0
2	1	1	0.5	1000	592	0
2	1	5	0.5	10	672	80
2	1	5	0.5	1000	592	0
2	2	1	0.5	10	705	113
2	2	1	0.5	1000	592	0
2	3	1	0.5	10	592	0
2	3	1	0.5	1000	705	113
2	3	5	0.5	10	592	0
2	3	5	0.5	1000	592	0
3	2	1	0.5	10	592	0
3	2	1	0.5	1000	592	0
3	2	5	0.5	10	592	0
3	2	5	0.5	1000	683	91
3	3	1	0.5	10	592	0
3	3	1	0.5	1000	592	0
3	3	5	0.5	10	683	91
3	3	5	0.5	1000	663	71

В 72 случаев из 108 программа выдавала правильный результат.

Проанализируем ошибки. В таблице 4.3 приведены сведения о средней ошибке в зависимости от количества итераций.

Таблица 4.3: Результаты тестов

Количество итераций	Количество ошибок	Средняя ошибка
10	17	51.84
100	10	27
1000	9	23.47

Видно, что с ростом количества итераций количество ошибок и средняя ошибка падают. От значения остальных параметров зависимостей не обнаружено, что о говорит о том, что значения параметром должны регулироваться в зависимости от задачи.

Лучшие параметры - параметры, у которых в столбце ошибок значение 0. Например, лучшей комбинации аргументов соответствует строка 2 в таблице 4.3. Полный список параметров, при котором программа выдаёт правильный ответ, можно увидеть в приложении 1.

4.4 Вывод

Применимость алгоритмов зависит от того, насколько велик размер матрицы расстояний. При размере $N < 9$ алгоритм полного перебора работает быстрее, чем муравьиный алгоритм. При $N = 2$ перебор быстрее в 275 раз, а при $N = 8$ в 4 раза. Но при дальнейшем увеличении матрицы муравьиный алгоритм работает быстрее. При $N = 12$ муравьиный алгоритм превосходит перебор в 82 раза. При $N > 12$ время для подбора подсчитать не удалось – оно слишком велико.

Муравьиный алгоритм при различных параметрах выдаёт разные ответы. Регулировка параметров производится вручную, и значения параметров зависят от задачи.

5 Заключение

В ходе работы была рассмотрена задача коммивояжёра и два алгоритма для её решения: муравьиный и полный перебор. Были составлены схемы алгоритмов. Было реализовано работоспособное ПО, удалось провести анализ зависимости затрат по времени от размера матрицы расстояний.

Применимость алгоритмов зависит от того, насколько велик размер матрицы расстояний. При размерах $N < 9$ алгоритм полного перебора работает быстрее, чем муравьиный алгоритм. При $N = 2$ перебор быстрее в 275 раз, а при $N = 8$ в 4 раза.

Но при дальнейшем увеличении матрицы муравьиный алгоритм работает быстрее. При $N = 12$ муравьиный алгоритм превосходит перебор в 82 раза. При $N > 12$ время для подбора подсчитать не удалось – оно слишком велико.

Муравьиный алгоритм при различных параметрах выдаёт разные ответы. Регулировка параметров производится вручную, и подходящие значения параметров зависят от задачи.

Литература

[1] Perl problems - Commis Voyageur [Электронный ресурс]. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения: 12.12.2020).

[2] Муравьиные алгоритмы [Электронный ресурс]. Режим доступа: http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D1%83%D1%80%D0%B0%D0%B2%D1%8C%D0%B8%D0%BD%D1%8B%D0%B5_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B (дата обращения: 12.12.2020).

[3] С.Д. Штовба. Муравьиные алгоритмы. 1970.

[4] time - Time access and conversions. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 21.09.2021).

[5] Процессор Intel® Core™ i5-1135G7. Режим доступа https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 15.10.2021).

Приложение 1

α	β	Q	p	max_iter	Результат	Ошибка
1	1	1	0.5	10	804	212
1	1	1	0.5	100	592	0
1	1	1	0.5	1000	592	0
1	1	3	0.5	10	715	123
1	1	3	0.5	100	592	0
1	1	3	0.5	1000	592	0
1	1	5	0.5	10	681	89
1	1	5	0.5	100	592	0
1	1	5	0.5	1000	592	0
1	1	10	0.5	10	592	0
1	1	10	0.5	100	592	0
1	1	10	0.5	1000	592	0
1	2	1	0.5	10	705	113
1	2	1	0.5	100	592	0
1	2	1	0.5	1000	592	0
1	2	3	0.5	10	663	71
1	2	3	0.5	100	683	91
1	2	3	0.5	1000	705	113
1	2	5	0.5	10	592	0
1	2	5	0.5	100	592	0
1	2	5	0.5	1000	705	113
1	2	10	0.5	10	592	0
1	2	10	0.5	100	683	91
1	2	10	0.5	1000	592	0
1	3	1	0.5	10	592	0
1	3	1	0.5	100	592	0
1	3	1	0.5	1000	683	91
1	3	3	0.5	10	737	145
1	3	3	0.5	100	683	91
1	3	3	0.5	1000	592	0
1	3	5	0.5	10	683	91

1	3	5	0.5	100	592	0
1	3	5	0.5	1000	683	91
1	3	10	0.5	10	681	89
1	3	10	0.5	100	683	91
1	3	10	0.5	1000	592	0
2	1	1	0.5	10	592	0
2	1	1	0.5	100	592	0
2	1	1	0.5	1000	592	0
2	1	3	0.5	10	592	0
2	1	3	0.5	100	592	0
2	1	3	0.5	1000	592	0
2	1	5	0.5	10	672	80
2	1	5	0.5	100	592	0
2	1	5	0.5	1000	592	0
2	1	10	0.5	10	592	0
2	1	10	0.5	100	592	0
2	1	10	0.5	1000	592	0
2	2	1	0.5	10	705	113
2	2	1	0.5	100	592	0
2	2	1	0.5	1000	592	0
2	2	3	0.5	10	705	113
2	2	3	0.5	100	683	91
2	2	3	0.5	1000	592	0
2	2	5	0.5	10	683	91
2	2	5	0.5	100	683	91
2	2	5	0.5	1000	592	0
2	2	10	0.5	10	592	0
2	2	10	0.5	100	683	91
2	2	10	0.5	1000	592	0
2	3	1	0.5	10	592	0
2	3	1	0.5	100	683	91
2	3	1	0.5	1000	705	113
2	3	3	0.5	10	592	0
2	3	3	0.5	100	592	0

2	3	3	0.5	1000	592	0
2	3	5	0.5	10	592	0
2	3	5	0.5	100	592	0
2	3	5	0.5	1000	592	0
2	3	10	0.5	10	705	113
2	3	10	0.5	100	592	0
2	3	10	0.5	1000	592	0
3	1	1	0.5	10	683	91
3	1	1	0.5	100	592	0
3	1	1	0.5	1000	592	0
3	1	3	0.5	10	592	0
3	1	3	0.5	100	592	0
3	1	3	0.5	1000	592	0
3	1	5	0.5	10	592	0
3	1	5	0.5	100	592	0
3	1	5	0.5	1000	592	0
3	1	10	0.5	10	592	0
3	1	10	0.5	100	592	0
3	1	10	0.5	1000	592	0
3	2	1	0.5	10	592	0
3	2	1	0.5	100	592	0
3	2	1	0.5	1000	592	0
3	2	3	0.5	10	720	128
3	2	3	0.5	100	592	0
3	2	3	0.5	1000	592	0
3	2	5	0.5	10	592	0
3	2	5	0.5	100	592	0
3	2	5	0.5	1000	683	91
3	2	10	0.5	10	705	113
3	2	10	0.5	100	592	0
3	2	10	0.5	1000	592	0
3	3	1	0.5	10	592	0
3	3	1	0.5	100	683	91
3	3	1	0.5	1000	592	0

3	3	3	0.5	10	592	0
3	3	3	0.5	100	592	0
3	3	3	0.5	1000	683	91
3	3	5	0.5	10	683	91
3	3	5	0.5	100	592	0
3	3	5	0.5	1000	663	71
3	3	10	0.5	10	592	0
3	3	10	0.5	100	745	153
3	3	10	0.5	1000	663	71