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 Комми вояжёр

Студент Пронина Л.Ю.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Задача коммивояжера 4

1.2 Алгоритм полного перебора для решения задачи коммивояжера 4

1.3 Муравьиный алгоритм 4

1.4 Вариация муравьиного алгоритма с элитными муравьями . . . 6

2 Конструкторская часть **7**

2.1 Требования к программному обеспечению 7

2.2 Разработка алгоритмов 7

3 Технологическая часть **13**

3.1 Средства реализации 13

3.2 Описание используемых типов данных 13

3.3 Листинги кода 13

3.4 Сведения о модулях программы 16

3.5 Функциональные тесты 17

4 Исследовательская часть **18**

4.1 Технические характеристики 18

4.2 Демонстрация работы программы 18

4.3 Время выполнения реализаций алгоритмов 20

4.4 Постановка эксперимента 21

4.4.1 Класс данных 1 22

4.4.2 Класс данных 2 24

4.4.3 Класс данных 3 25

Заключение **29**

Список используемых источников **30**

Введение

Лабораторная работа посвящена изучению одной из самых известных задач комбинаторной оптимизации — проблеме коммивояжера. Проблема коммивояжера заключается в поиске оптимального пути для коммивояжера, который должен посетить заданное количество городов и пройти через каждый город ровно один раз, чтобы минимизировать общую длину пути.

Целью данной работы является описание муравьиного алгоритма, примененного к задаче коммивояжера. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) описать муравьиный алгоритм и алгоритм полного перебора для решения задачи коммивояжера;

2) реализовать эти алгоритмы;

3) провести параметризацию муравьиного алгоритма и сравнить время работы реализаций муравьиного алгоритма и алгоритма полного перебора;

4) описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В этом разделе будет представлена информация о задаче коммивояжера, а также о путях ее решения — полным перебором и муравьиным алгоритмом, а также информация о модификации муравьиного алгоритма с элитными муравьями.

1.1 Задача коммивояжера

Задача коммивояжера — (задача о бродячем торговце) одна из самых важных задач транспортной логистики, в которой рассматриваются вершины графа, а также матрица смежности для расстояния между вершинами. Ее суть — найти такой порядок посещения вершин графа, при котором путь будет минимален, а каждая вершина будет посещена только один раз [1].

1.2 Алгоритм полного перебора для решения задачи коммивояжера

Полный перебор для задачи коммивояжера — имеет высокую сложность алгоритма ($n!$). Идея в полном переборе всех возможных путей в графе и выбор наименьшего из них. Решение будет получено, но потребуются большие затраты по времени выполнения даже при небольшом количестве вершин в графе [2].

1.3 Муравьиный алгоритм

Муравьиный алгоритм — основан на принципе поведения колонии муравьев [3].

Муравьи действуют, ощущая некий феромон. Каждый муравей, чтобы другие могли ориентироваться, оставляет на своем пути феромоны. При большом количестве прохождения муравьев, наибольшее количество феромона остается на оптимальном пути.

Суть в том, что муравей в одиночку ничего не может, поскольку способен выполнять только максимально простые задачи. Но при условии большого количества таких муравьев они могут самоорганизовываться для решения сложных задач.

Пусть муравей имеет следующие характеристики:

1) зрение — способен определить длину ребра;

2) память — запоминает пройденный маршрут;

3) обоняние — чувствует феромон.

Также введем целевую функцию (1.1).

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

где D_{ij} — расстояние из текущего пункта i до заданного пункта j.

А также понадобится формула вычисления вероятности перехода в заданную точку (1.2).

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q=1}^{m} \tau_{iq}^a \eta_{iq}^b}, \text{вершина не была посещена ранее муравьем k}, \\ 0, \text{иначе} \end{cases} \tag{1.2}$$

где a — параметр влияния длины пути, b — параметр влияния феромона, τ_{ij} — расстояния от города i до j, η_{ij} — количество феромонов на ребре ij.

После завершения движения всех муравьев, формула обновляется феромон по формуле (1.3):

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

При этом

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

где

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k, \text{ребро посещено k-ым муравьем}, \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

Путь выбирается по следующей схеме:

1) Каждый муравей имеет список запретов — список уже посещенных городов (вершин графа).

2) Муравьиной зрение — отвечает за желание посетить вершину.

3) Муравьиное обоняние — отвечает за ощущение феромона на определенном пути (ребре). При этом количество феромона на пути (ребре) в момент времени t обозначается как $\tau_{i,j}(t)$.

4) После прохождения определенного ребра муравей откладывает на нем некотрое количество феромона, которое показывает оптимальность сделанного выбора (это количество вычисляется по формуле (1.5))

1.4 Вариация муравьиного алгоритма с элитными муравьями

Из общего числа муравьев выделяются так называемые «элитные муравьи». По результатам каждой итерации алгоритма производится усиление лучших маршрутов путём прохода по данным маршрутам элитных муравьев и, таким образом, увеличение количества феромона на данных маршрутах. В такой системе количество элитных муравьев является дополнительным параметром, требующим определения. Так, для слишком большого числа элитных муравьев алгоритм может «застрять» на локальных экстремумах [3].

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также полный перебор для ее решения и муравьиный алгоритм и его модификация с элитными муравьями.

2 Конструкторская часть

В этом разделе будет представлено описание используемых типов данных, а также схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

1) программа должна получать на вход матрицу смежности, для которой можно будет выбрать один из алгоритмов поиска оптимальных путей (полным перебором или муравьиным алгоритмом);

2) программа должна давать возможность получить минимальную сумму пути, а также сам путь, используя один из алгоритмов;

3) программа должна иметь возможность провести параметризацию муравьиного алгоритма, а также замерить процессорное время работы реализаций алгоритмов.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора путей, а на рисунках 2.2 и 2.3 схема муравьиного алгоритма поиска путей. Также на рисунках 2.4 и 2.5 представлены схемы вспомогательных функций для муравьиного алгоритма.

Вывод

В данном разделе были представлены схемы алгоритмов, рассматриваемых в лабораторной работе и требования к программному обеспечению.

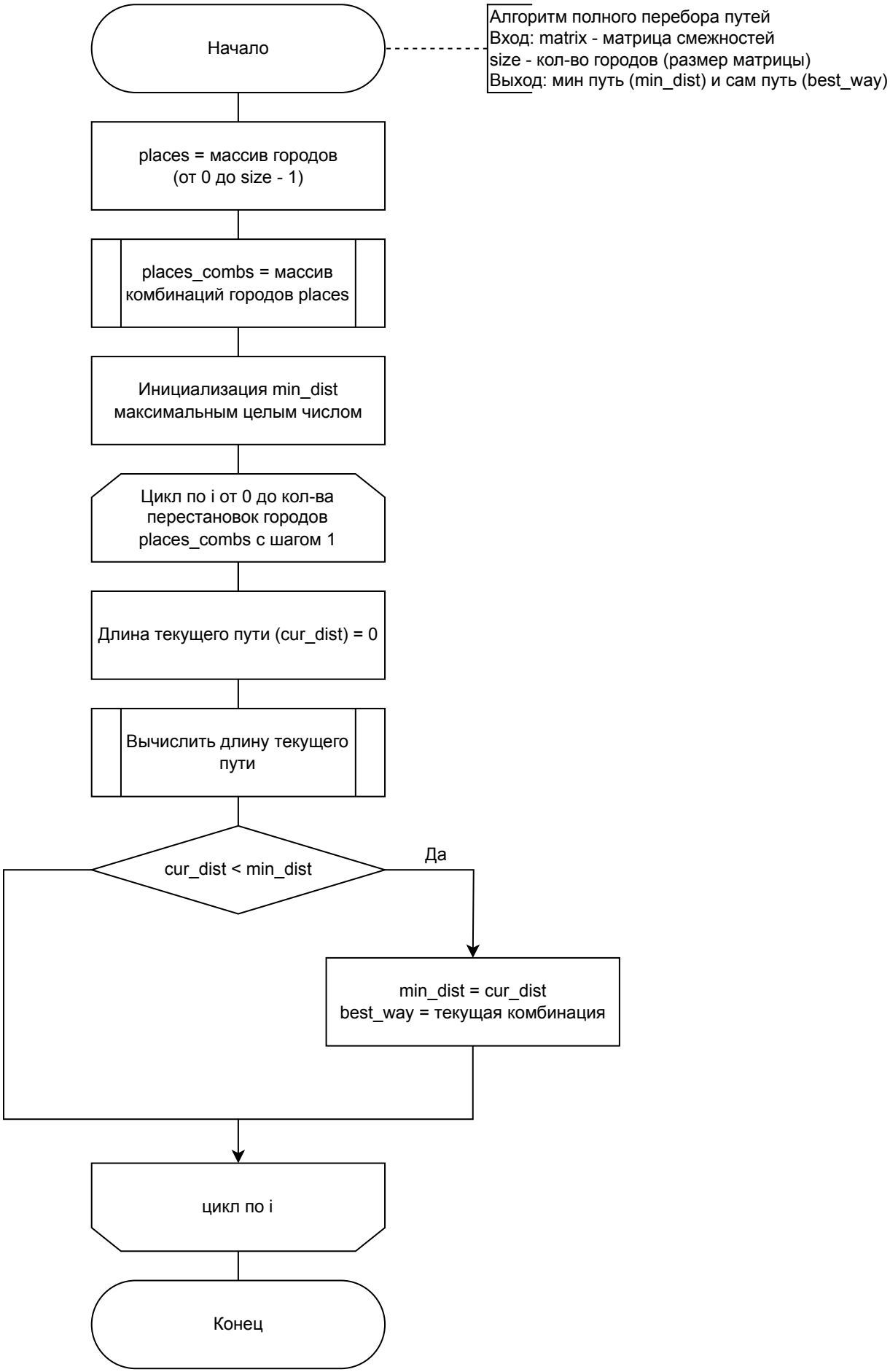

Рисунок 2.1 – Схема алгоритма полного перебора путей

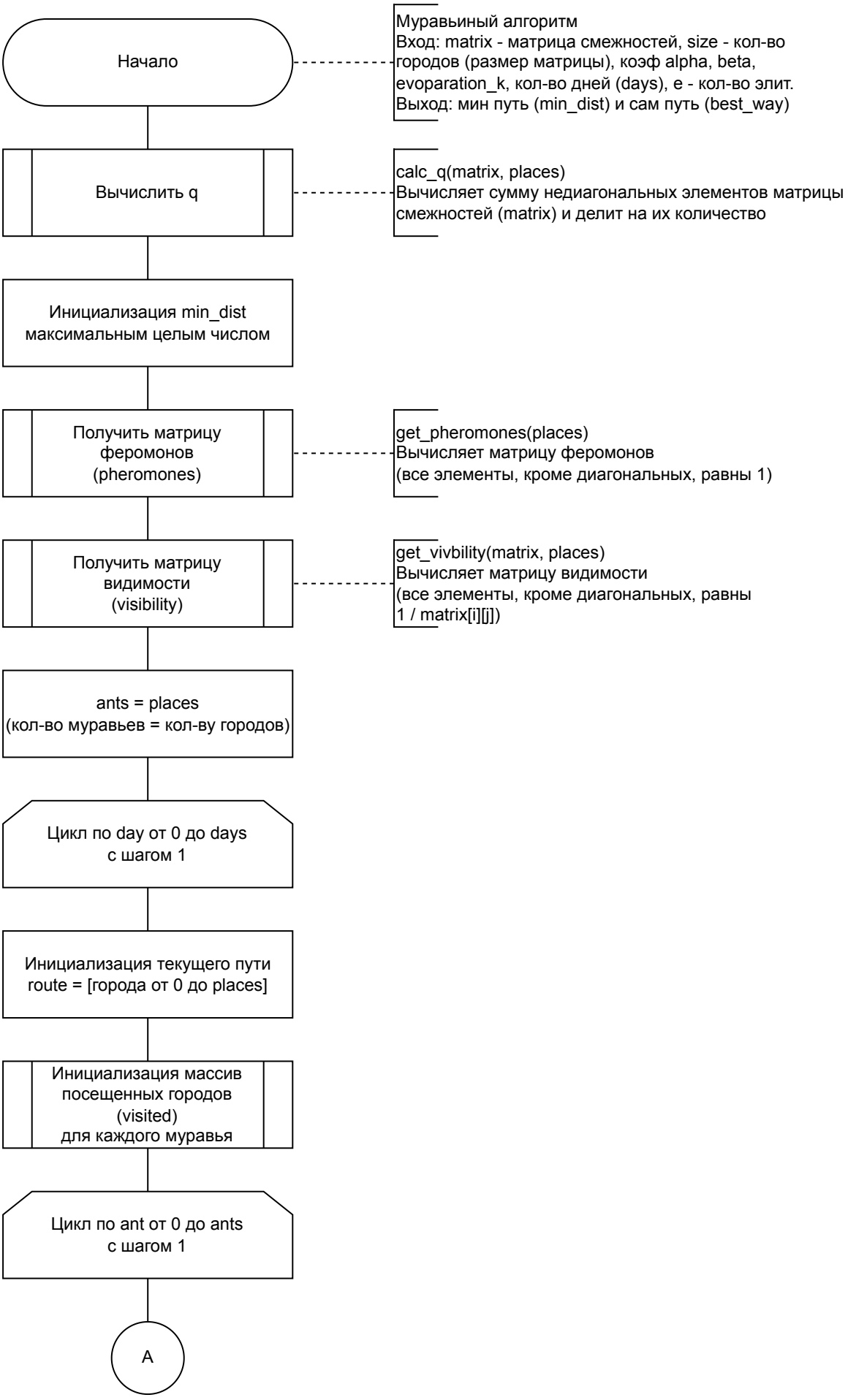

Рисунок 2.2 – Схема муравьиного алгоритма (часть 1)

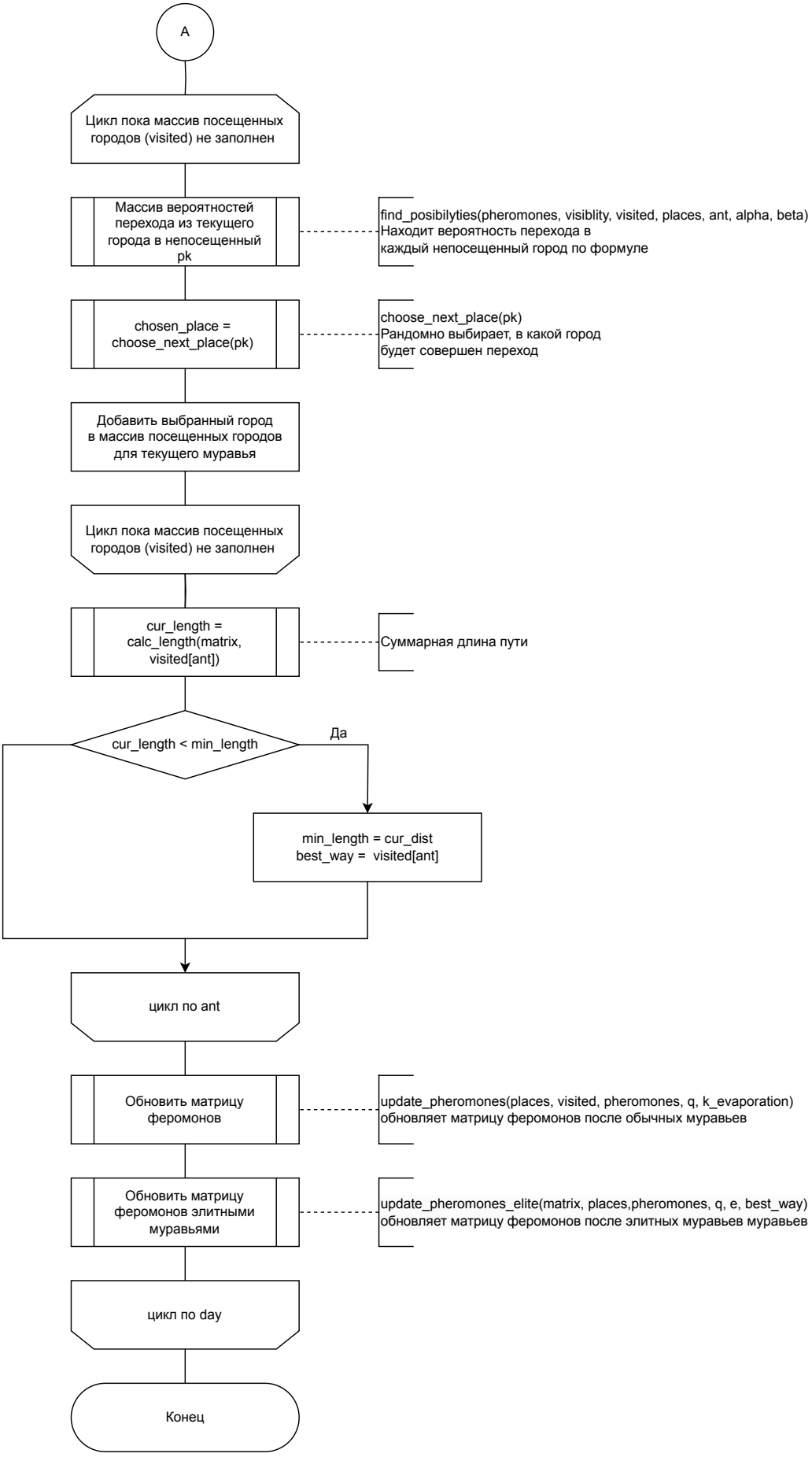

Рисунок 2.3 – Схема муравьиного алгоритма (часть 2)

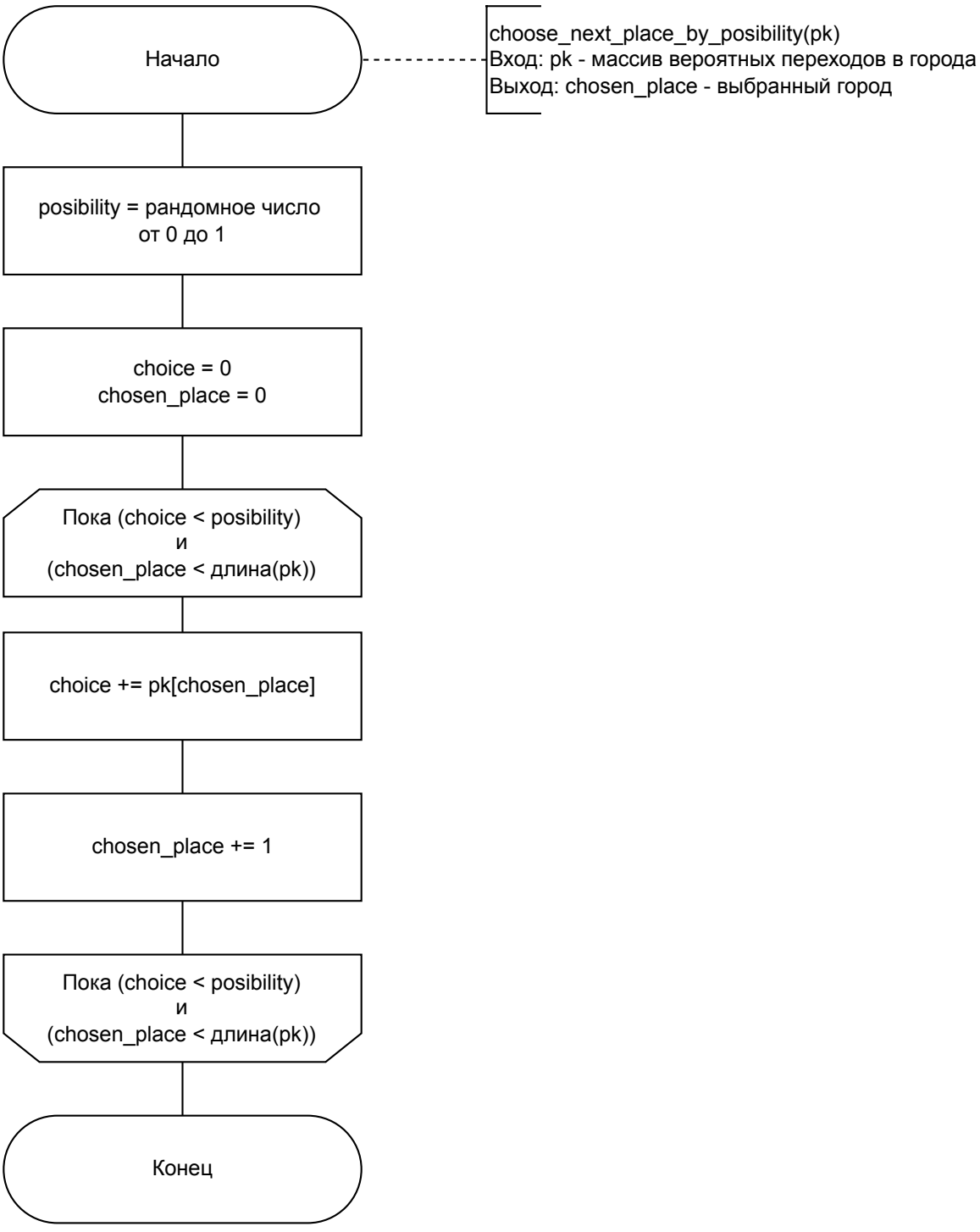

Рисунок 2.4 – Схема алгоритма нахождения массива вероятностных переходов в непосещенные города

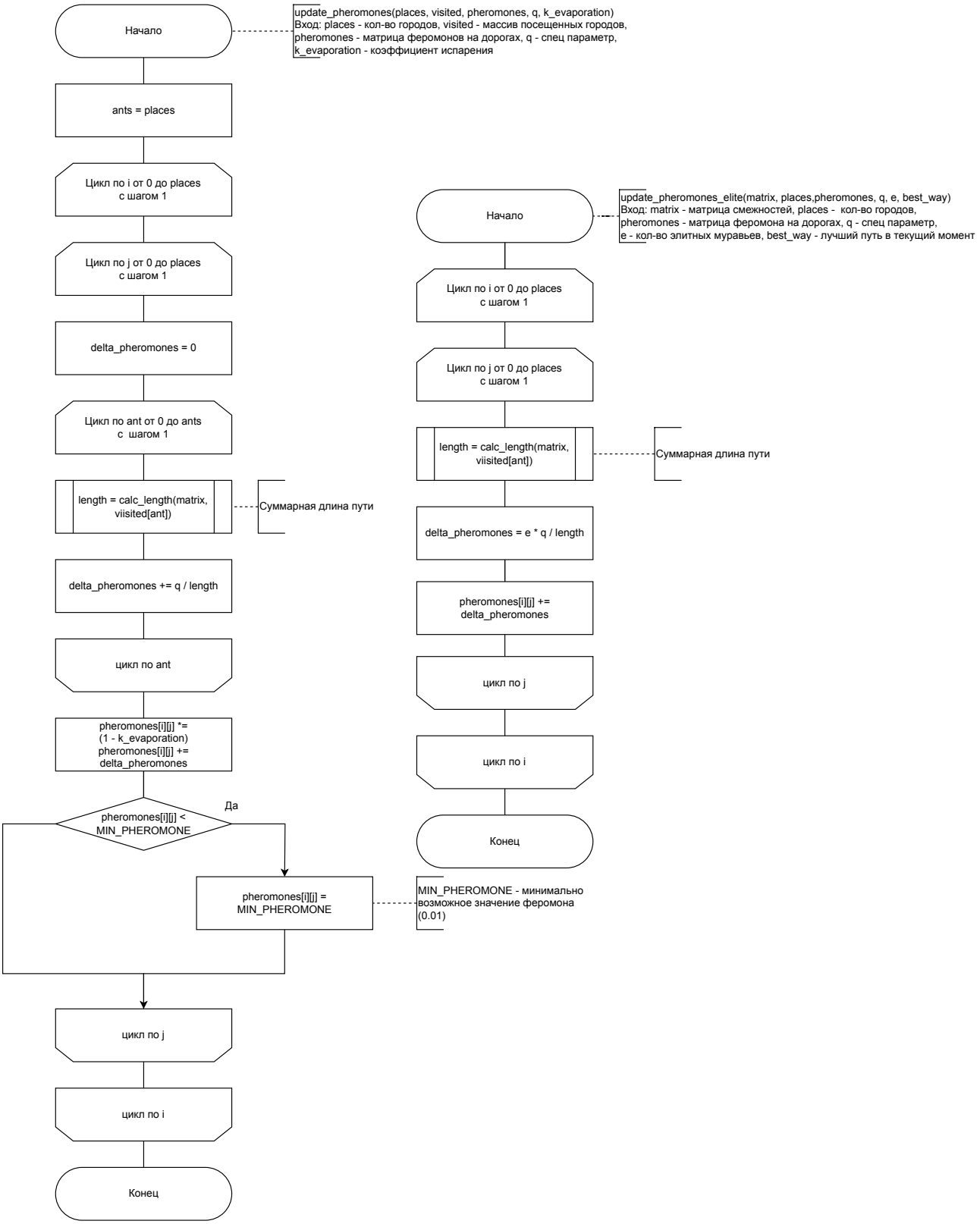

Рисунок 2.5 – Схема алгоритмов обновления матрицы феромонов

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритмов полного перебора путей и муравьиного алгоритма.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [4]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time* [5].

3.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

1) размер матрицы смежности — целое число типа *int*;
2) название файла — строка типа *str*;
3) коэффициент α, β, *evaporation_koef* — числа типа *float*;
4) коэффициент e — число типа *int*;
5) матрица смежностей — матрица типа *int* для хранения длины путей между городами.

3.3 Листинги кода

В листинге 3.1 представлен алгоритм полного перебора путей, а в листингах 3.2–3.6 — муравьиный алгоритм и дополнительные к нему функции.

Листинг 3.1 – Алгоритм полного перебора путей

```
def full_combinations(matrix, size):
    places = np.arange(size)
    places_combs = list()
    for combination in it.permutations(places):
        comb_arr = list(combination)
        places_combs.append(comb_arr)
    min_dist = float("inf")
    for i in range(len(places_combs)):
        cur_dist = 0
        for j in range(size - 1):
            start_city = places_combs[i][j]
            end_city = places_combs[i][j + 1]
            cur_dist += matrix[start_city][end_city]
        if (cur_dist < min_dist):
            min_dist = cur_dist
            best_way = places_combs[i]
    return min_dist, best_way
```

Листинг 3.2 – Алгоритм нахождения массива вероятностных переходов в непосещенные города

```
def find_posibilyties(pheromones, visibility, visited, places, ant,
   alpha, beta):
    pk = [0] * places
    for place in range(places):
        if place not in visited[ant]:
            ant_place = visited[ant][-1]
            pk[place] = pow(pheromones[ant_place][place], alpha) * \
            pow(visibility[ant_place][place], beta)
        else:
            pk[place] = 0
    sum_pk = sum(pk)
    for place in range(places):
        pk[place] /= sum_pk
    return pk
```

Листинг 3.3 – Муравьиный алгоритм

```
def ant_algorythm(matrix, places, days, alpha, beta, k_evaporation,
    e):
    q = calc_q(matrix, places)
    best_way = []
    min_length = float("inf")
    pheromones = get_pheromones(places)
    visibility = get_visibility(matrix, places)
    ants = places
    for day in range(days):
        route = np.arange(places)
        visited = get_visited_places(route, ants)
        for ant in range(ants):
            while (len(visited[ant]) != ants):
                pk = find_posibilyties(pheromones, visibility,
                    visited, places, ant, alpha, beta)
                chosen_place = choose_next_place_by_posibility(pk)
                visited[ant].append(chosen_place - 1)
            cur_length = calc_length(matrix, visited[ant])
            if (cur_length < min_length):
                min_length = cur_length
                best_way = visited[ant]
        pheromones = update_pheromones(matrix, places, visited,
            pheromones, q, k_evaporation, e)
        pheromones = update_pheromones_elite(matrix, places,
            pheromones, q, e, best_way)
    return min_length, best_way
```

Листинг 3.4 – Алгоритм нахождения следующего города на основании рандома

```
def choose_next_place_by_posibility(pk):
    posibility = random()
    choice = 0
    chosen_place = 0
    while ((choice < posibility) and (chosen_place < len(pk))):
        choice += pk[chosen_place]
        chosen_place += 1
    return chosen_place
```

Листинг 3.5 – Алгоритм обновления матрицы феромонов для обычных муравьев

```
def update_pheromones(matrix, places, visited, pheromones, q,
    k_evaporation):
    ants = places
    for i in range(places):
        for j in range(places):
            delta_pheromones = 0
            for ant in range(ants):
                length = calc_length(matrix, visited[ant])
                delta_pheromones += q / length
            pheromones[i][j] *= (1 - k_evaporation)
            pheromones[i][j] += delta_pheromones
            if (pheromones[i][j] < MIN_PHEROMONE):
                pheromones[i][j] = MIN_PHEROMONE
    return pheromones
```

Листинг 3.6 – Алгоритм обновления матрицы феромонов для элитных муравьев

```
def update_pheromones_elite(matrix, places, pheromones, q, e,
    best_way):
    for i in range(places):
        for j in range(places):
            length = calc_length(matrix, best_way)
            delta_pheromones = e * q / length
            pheromones[i][j] += delta_pheromones
    return pheromones
```

3.4 Сведения о модулях программы

Программа состоит из двух модулей:

1) *main.py* — файл, содержащий меню программы, а также весь служебный код;

2) *algorythms.py* — файл, содержащий реализацию алгоритма полного перебора и муравьиного алгоритма.

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций программы. Тесты *для всех функций* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 3 & 8 & 3 & 9 \\ 7 & 0 & 5 & 2 & 8 \\ 5 & 7 & 0 & 7 & 6 \\ 7 & 5 & 5 & 0 & 9 \\ 10 & 4 & 5 & 3 & 0 \end{pmatrix}$	16, [0, 1, 3, 2, 4]	16, [0, 1, 3, 2, 4]
$\begin{pmatrix} 0 & 6 & 4 \\ 3 & 0 & 5 \\ 3 & 6 & 0 \end{pmatrix}$	7, [1, 0, 2]	7, [1, 0, 2]
$\begin{pmatrix} 0 & 27 & 12 & 21 \\ 26 & 0 & 16 & 24 \\ 28 & 11 & 0 & 11 \\ 14 & 20 & 25 & 0 \end{pmatrix}$	37, [3, 0, 2, 1]	37, [3, 0, 2, 1]

Вывод

В данном разделе была приведена информация о выбранных средствах для реализации алгоритмов, листинги реализаций всех алгоритмов — полного перебора и муравьиного и сведения о модулях программы, а так же проведено функциональное тестирование.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных и проведена параметризация реализации муравьиного алгоритма.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент представлены далее:

1) операционная система — Ubuntu 22.04.3 [6] Linux x86_64;

2) память — 16 Гб;

3) процессор — Intel® Core™ i5-1135G7 @ 2.40 ГГц.

При эксперименте ноутбук не был включен в сеть электропитания.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен пример работы программы для обоих алгоритмов — полного перебора и муравьиного.

```
      1. Полный перебор
      2. Муравьиный алгоритм
      3. Оба алгоритма
      4. Параметризация
      5. Замерить время
      6. Обновить данные
      7. Распечатать матрицу
      0. Выход
          Выбор:     1


Доступные файлы:
1. mat10.csv
2. mat3.csv
3. mat4.csv
4. mat5.csv

Выберите файл: 4


Минимальная сумма пути =  15
Путь:  [4, 2, 0, 1, 3]


       Меню:

      1. Полный перебор
      2. Муравьиный алгоритм
      3. Оба алгоритма
      4. Параметризация
      5. Замерить время
      6. Обновить данные
      7. Распечатать матрицу
      0. Выход
          Выбор:     2


Доступные файлы:
1. mat10.csv
2. mat3.csv
3. mat4.csv
4. mat5.csv

Выберите файл: 4


Введите коэффициент alpha: 0.5
Введите коэффициент evaporation: 0.5
Введите кол-во дней: 200
Введите кол-во элитных муравьев: 3


Минимальная сумма пути =  15
Путь:  [4, 2, 0, 1, 3]
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализаций алгоритмов

Как было сказано выше, используется функция замера процессорного времени *process_time(...)* из библиотеки *time* на *Python*. Функция возвращает процессорное время типа float в секундах.

Функция используется дважды: перед началом выполнения алгоритма и после завершения, затем из конечного времени вычитается начальное, чтобы получить результат.

Замеры проводились для разного размера матриц, чтобы определить, когда наиболее эффективно использовать муравьиный алгоритм.

Результаты замеров приведены в таблице 4.1 (время в с).

Таблица 4.1 – Результаты замеров времени

Размер	Полный перебор	Муравьиный алгоритм
2	0.0000	0.0023
3	0.0000	0.0072
4	0.0001	0.0184
5	0.0002	0.0385
6	0.0014	0.0718
7	0.0113	0.1202
8	0.1090	0.1873
9	1.1419	0.2825
10	12.7923	0.4079

Также на рисунке 4.2 приведены графические результаты замеров.

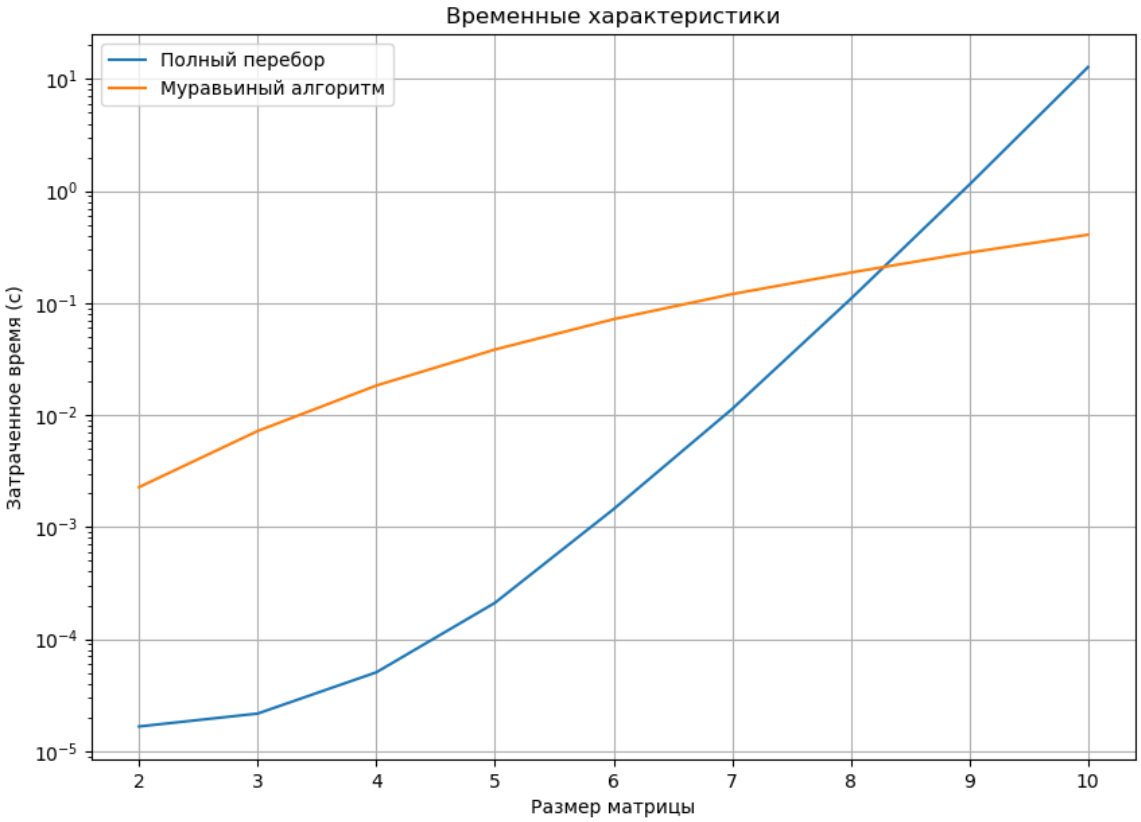

Рисунок 4.2 – Сравнение по времени алгоритмов полного перебора путей и муравьиного на разных размерах матрицы

4.4 Постановка эксперимента

Автоматическая параметризация была проведена на трех классах данных — 4.1 – 4.3. Алгоритм будет запущен для всех значений $\alpha, \rho \in (0, 1), days = \{1, 3, 5, 10, 50, 100, 300, 500\}, e = \{3, 10, 20\}$.

Итоговая таблица значений параметризации будет состоять из следующих колонок:

- α — изменяющийся параметр;
- ρ — изменяющийся параметр;
- $days$ — кол-во дней, изменяющийся параметр;
- e — кол-во элитных муравьев;

- *Result* — эталонный результат;
- *Mistake* — разность получившегося значения и эталонного значения на данных значениях параметров.

Цель эксперимента — определить комбинацию параметров, которые решают задачу наилучшим образом. Качество зависит от двух факторов:

- количества дней;
- погрешности измерений.

4.4.1 Класс данных 1

Класс данных 1 представляет собой матрицу смежности размером 10 элементов, которая представлена далее.

$$K_1 = \begin{pmatrix} 0 & 14 & 1 & 3 & 4 & 26 & 23 & 20 & 21 & 15 \\ 21 & 0 & 25 & 19 & 19 & 16 & 7 & 15 & 8 & 7 \\ 24 & 30 & 0 & 24 & 28 & 5 & 10 & 20 & 25 & 15 \\ 20 & 2 & 23 & 0 & 26 & 20 & 21 & 2 & 17 & 23 \\ 27 & 24 & 17 & 18 & 0 & 17 & 27 & 15 & 23 & 6 \\ 13 & 19 & 1 & 20 & 22 & 0 & 20 & 14 & 21 & 6 \\ 12 & 10 & 10 & 19 & 19 & 26 & 0 & 12 & 24 & 16 \\ 4 & 8 & 17 & 10 & 19 & 5 & 3 & 0 & 22 & 29 \\ 10 & 9 & 20 & 21 & 21 & 28 & 7 & 19 & 0 & 19 \\ 23 & 13 & 29 & 9 & 15 & 1 & 2 & 22 & 5 & 0 \end{pmatrix} \tag{4.1}$$

Для данного класса данных приведена таблица с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.2 – Параметризация для класса данных 1

α	ρ	Days	e	Result	Mistake
0.1	0.1	500	3	45	0

0.1	0.2	500	20	45	0
0.1	0.3	500	3	45	0
0.1	0.3	500	10	45	0
0.1	0.5	100	20	45	0
0.1	0.5	500	20	45	0
0.1	0.8	500	10	45	0
0.2	0.1	500	20	45	0
0.2	0.2	500	10	45	0
0.2	0.3	300	20	45	0
0.2	0.4	500	3	45	0
0.2	0.5	300	10	45	0
0.2	0.7	100	10	45	0
0.2	0.8	300	3	45	0
0.2	0.8	300	10	45	0
0.3	0.3	300	10	45	0
0.3	0.3	300	20	45	0
0.3	0.5	500	20	45	0
0.4	0.2	300	20	45	0
0.4	0.8	500	3	45	0
0.5	0.4	300	10	45	0
0.5	0.5	500	10	45	0
0.5	0.6	50	3	45	0
0.6	0.2	500	20	45	0
0.6	0.4	500	3	45	0
0.6	0.7	50	10	45	0
0.7	0.5	300	3	45	0
0.8	0.2	300	10	45	0
0.8	0.4	100	20	45	0
0.8	0.5	300	20	45	0

4.4.2 Класс данных 2

Класс данных 2 представляет собой матрицу смежности размером 10 элементов, которая представлена далее.

$$K_1 = \begin{pmatrix} 0 & 9271 & 8511 & 2010 & 1983 & 7296 & 7289 & 3024 & 1011 \\ 9271 & 0 & 7731 & 4865 & 5494 & 6812 & 4755 & 7780 & 7641 \\ 8511 & 7731 & 0 & 1515 & 9297 & 7506 & 5781 & 5804 & 7334 \\ 2010 & 4865 & 1515 & 0 & 3662 & 9597 & 2876 & 8188 & 9227 \\ 1983 & 5494 & 9297 & 3662 & 0 & 8700 & 4754 & 7445 & 3834 \\ 7296 & 6812 & 7506 & 9597 & 8700 & 0 & 4216 & 5553 & 8215 \\ 7289 & 4755 & 5781 & 2876 & 4754 & 4216 & 0 & 4001 & 4715 \\ 3024 & 7780 & 5804 & 8188 & 7445 & 5553 & 4001 & 0 & 9522 \\ 1011 & 7641 & 7334 & 9227 & 3834 & 8215 & 4715 & 9522 & 0 \end{pmatrix} \quad (4.2)$$

Для данного класса данных приведена таблица с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.3 – Параметризация для класса данных 2

α	ρ	Days	e	Result	Mistake
0.1	0.1	500	10	46	0
0.1	0.2	100	20	46	0
0.1	0.2	500	3	46	0
0.1	0.3	100	3	46	0
0.1	0.4	500	20	46	0
0.1	0.5	100	10	46	0
0.1	0.5	500	3	46	0
0.1	0.6	500	20	46	0
0.1	0.7	300	3	46	0
0.1	0.7	500	10	46	0
0.1	0.8	500	10	46	0
0.2	0.1	500	10	46	0

0.2	0.2	100	10	46	0
0.2	0.2	500	3	46	0
0.2	0.4	300	3	46	0
0.2	0.6	500	10	46	0
0.2	0.7	50	3	46	0
0.2	0.8	50	3	46	0
0.2	0.8	50	20	46	0
0.3	0.1	300	20	46	0
0.3	0.1	500	3	46	0
0.3	0.2	500	10	46	0
0.3	0.2	500	20	46	0
0.3	0.3	50	20	46	0
0.3	0.3	300	10	46	0
0.3	0.7	100	3	46	0
0.3	0.8	500	20	46	0
0.4	0.3	300	10	46	0
0.4	0.5	100	3	46	0
0.5	0.5	500	3	46	0
0.6	0.4	300	3	46	0
0.6	0.4	300	20	46	0
0.6	0.6	500	10	46	0
0.6	0.8	500	3	46	0
0.7	0.7	300	10	46	0
0.8	0.7	500	3	46	0

4.4.3 Класс данных 3

Класс данных 3 представляет собой матрицу смежности размером 10 элементов, которая представлена далее.

$$K_1 = \begin{pmatrix} 0 & 20 & 2 & 24 & 18 & 23 & 27 & 24 & 17 & 28 \\ 25 & 0 & 10 & 1 & 29 & 18 & 3 & 26 & 6 & 20 \\ 1 & 12 & 0 & 14 & 16 & 28 & 9 & 15 & 24 & 30 \\ 15 & 12 & 27 & 0 & 20 & 9 & 25 & 24 & 26 & 30 \\ 1 & 6 & 11 & 2 & 0 & 8 & 12 & 21 & 12 & 5 \\ 7 & 10 & 14 & 5 & 9 & 0 & 5 & 15 & 5 & 17 \\ 15 & 29 & 11 & 18 & 20 & 3 & 0 & 17 & 23 & 28 \\ 2 & 16 & 22 & 8 & 26 & 11 & 8 & 0 & 29 & 25 \\ 15 & 6 & 5 & 14 & 20 & 9 & 25 & 8 & 0 & 11 \\ 8 & 17 & 4 & 2 & 24 & 11 & 7 & 2 & 14 & 0 \end{pmatrix} \tag{4.3}$$

Для данного класса данных приведена таблица с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.4 – Параметризация для класса данных 3

α	ρ	Days	e	Result	Mistake
0.1	0.2	500	3	35	0
0.1	0.3	300	3	35	0
0.1	0.3	500	3	35	0
0.1	0.3	500	20	35	0
0.1	0.4	100	10	35	0
0.1	0.4	300	10	35	0
0.1	0.6	50	10	35	0
0.1	0.6	300	3	35	0
0.1	0.8	100	20	35	0
0.1	0.8	500	3	35	0
0.1	0.8	500	10	35	0
0.2	0.2	300	3	35	0
0.2	0.4	500	3	35	0
0.3	0.1	300	10	35	0
0.3	0.3	500	10	35	0
0.3	0.3	500	20	35	0

0.3	0.4	500	3	35	0
0.3	0.4	500	20	35	0
0.3	0.5	300	20	35	0
0.3	0.5	500	3	35	0
0.3	0.8	500	3	35	0
0.4	0.2	300	10	35	0
0.7	0.5	300	3	35	0
0.7	0.6	500	10	35	0

Вывод

В результате эксперимента было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так при размере матрицы, равном 4, муравьиный алгоритм медленее алгоритма полного перебора в 184 раза, а при размере матрицы, равном 9, муравьиный алгоритм быстрее алгоритма полного перебора в 4 раза, а при размере в 10 – уже в 31 раз. Следовательно, при размерах матриц больше 8 следует использовать муравьиный алгоритм, но стоит учитывать, что он может давать погрешности вычислений.

Также при проведении эксперимента с классами данных было получено, что при хотя бы 2 из 3 классов данных муравьиный алгоритм лучше всего показывает себя при параметрах:

1) $\alpha = 0.1, \rho = 0.2, 0.3, 0.4, 0.6, 0.8$;

2) $\alpha = 0.2, \rho = 0.1, 0.2, 0.3, 0.4, 0.8$;

3) $\alpha = 0.3, \rho = 0.3, 0.5, 0.8$;

4) $\alpha = 0.4, \rho = 0.2$;

5) $\alpha = 0.5, \rho = 0.5$;

6) $\alpha = 0.6, \rho = 0.4$.

7) $\alpha = 0.7, \rho = 0.5$;

Также во время исследования было замечено — чем меньше параметр α, тем меньше погрешностей возникает. Количество дней также сильно влияет на отсутствие погрешностей: чем больше значение параметра $Days$, тем меньше погрешностей.

Заключение

Результаты эксперимента показали, что при размере матрицы, равном 4, реализация муравьиного алгоритма работает в 184 раз медленнее, чем реализация алгоритма полного перебора. Однако, при размере матрицы, равном 9, реализация муравьиного алгоритма в 4 раза быстрее реализации алгоритма полного перебора, а при размере 10 — уже в 31 раз. Следовательно, для матриц размером более 8, рекомендуется использовать муравьиный алгоритм, но стоит учесть возможные погрешности вычислений.

Также при исследовании было обнаружено, что чем меньше значение параметра α, тем меньше возникает погрешностей. Кроме этого, количество дней (параметр $Days$) оказывает значительное влияние на отсутствие погрешностей. Чем больше значение $Days$, тем меньше погрешностей возникает.

В ходе лабораторной работы были выполнены следующие задачи:

1) описан муравьиный алгоритм и алгоритм полного перебора для решения задачи коммивояжера;

2) реализованы эти алгоритмы;

3) проведена параметризация муравьиного алгоритма и проведено сравнение времени работы реализаций муравьиного алгоритма и алгоритма полного перебора;

4) описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Все задачи были решены, следовательно, поставленная цель лабораторной работы была достигнута.

Список используемых источников

[1] Задача коммивояжёра [Электронный ресурс]. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения: 08.12.2023).

[2] Решаем задачу коммивояжёра простым перебором [Электронный ресурс]. Режим доступа: https://thecode.media/path-js/ (дата обращения: 08.12.2023).

[3] С.Д. Штовба. Муравьиные алгоритмы. Exponenta Pro. Математика в приложениях. 2003.

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 08.12.2023).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 08.12.2023).

[6] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 08.12.2023).

[7] Linux [Электронный ресурс]. Режим доступа: https://www.linux.org/forums/#linux-tutorials.122 (дата обращения: 08.12.2023).

[8] Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html (дата обращения: 08.12.2023).