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 «Анализы Алгоритмов»

на тему: «Методы решения задачи коммивояжера»

Студент группы ИУ7-56Б _____ (Подпись, дата) Мамврийский И. С. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Д. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Коммивояжер — задача поиска минимального по стоимости маршрута по всем вершинам без повторений на полном взвешанном графе с n вершинами. Вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояние(длины) или стоимость проезда. Эта задача является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность [1].

Для оптимизации поиска минимального маршрута используют муравьиный алгоритм, который базируется на моделировании поведения колонии муравьев [2].

Целью данной лабораторной работы является исследование метода решения задачи коммивояжера на базе муравьиного алгоритма.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать задачу коммивояжера;

— описать алгоритмы решения задачи коммивояжера – полный перебор и муравьиный алгоритм;

— разработать программное обеспечение, реализующее данные алгоритмы;

— измерить процессорное время работы данных алгоритмов;

— провести сравнительный анализ по времени реализованных алгоритмов.

1 Аналитическая часть

В этом разделе будет представлена информация о задаче коммивояжера, а также о путях ее решения – полным перебором и муравьиным алгоритмом.

1.1 Задача коммивояжера

Задача коммивояжера заключается в поиске кратчайшего замкнутого маршрута, проходящего через все города ровно 1 раз [2]. Если говорить формально, то необходимо найти гамильтонов цикл минимального веса во взвешенном полном графе.

1.2 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Данный алгоритм имеет высокую сложность $O(n!)$, что влечет большие затраты по времени даже при небольших значениях числа вершин в графе.

1.3 Муравьиный алгоритм

Муравьиный алгоритм основан на принципе поведения колонии муравьев.

Муравьи действуют, ощущая некий феромон. Каждый муравей, чтобы другие могли ориентироваться, оставляет его на своем пути. При этом феромон испаряется для исключения случая, когда все муравьи движутся одним и тем же субоптимальным маршрутом. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Суть в том, что отдельно взятый муравей мало, что может, поскольку способен выполнять только максимально простые задачи. Но при большом количестве муравьи могут самоорганизовываться в большие очереди для решения сложных задач.

Для каждого муравья переход из города i в город j зависит от трех составляющих:

— **память муравья** — список посещенных муравьем городов, заходить в которые еще раз нельзя. Используя этот список, муравей гарантированно не попадет в один и тот же город дважды.

— **видимость** — величина, обратная расстоянию: $\eta_{ij} = 1/D_{ij}$, где D_{ij} — расстояние между городами i и j. Эта величина выражает эвристическое желание муравья поседить город i из города j, чем ближе город, тем больше желание его посетить.

— **виртуальный след феромона** τ_{ij} на ребре (i, j) представляет подтвержденное муравьиным опытом желание посетить город j из города i.

Вероятность перехода k-ого муравья из города i в город j на t-й итерации определяется формулой 1.1:

$$\begin{cases} P_{ij,k}(t) = \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum\limits_{l \in J_{i,k}} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} & \text{если } j \in J_{i,k}, \\ P_{ij,k}(t) = 0 & \text{если } j \notin J_{i,k} \end{cases}, \tag{1.1}$$

где α, β – настраиваемые параметры, $J_{i,k}$ - список городов, которые надо посетить k-ому муравью, находящемуся в i-ом городе, τ - концентрация феромона, а при $\alpha = 0$ алгоритм вырождается в жадный.

После завершения движения всеми муравьями происходит обновление феромона. Если $p \in [0, 1]$ – коэффициент испарения феромона, то новое значения феромона на ребре (i, j) рассчитывается по формуле 1.2:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}, \quad \Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^{k} \tag{1.2}$$

где

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & \text{ребро посещено k-ым муравьем,} \\ 0, & \text{иначе} \end{cases} \tag{1.3}$$

L_k — длина пути k-ого муравья, Q — некоторая константа порядка длины путей, N — количество муравьев [2].

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также алгоритм полного перебор для ее решения и муравьиный алгоритм.

2 Конструкторская часть

В данном разделе будут разарботаны алгоритмы решения задачи коммивояжера: полный перебор и муравьиный алгоритм, также описана структура программы.

2.1 Реализация алгоритмов

На рисунках 2.1 – 2.9 представлена схема алгоритма метода полного перебора.

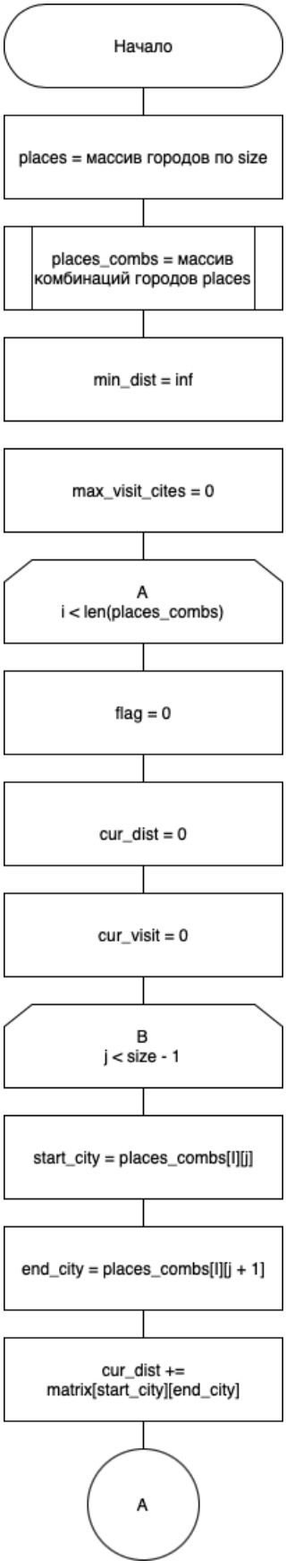

Рисунок 2.1 – Схема алгоритма полного перебора

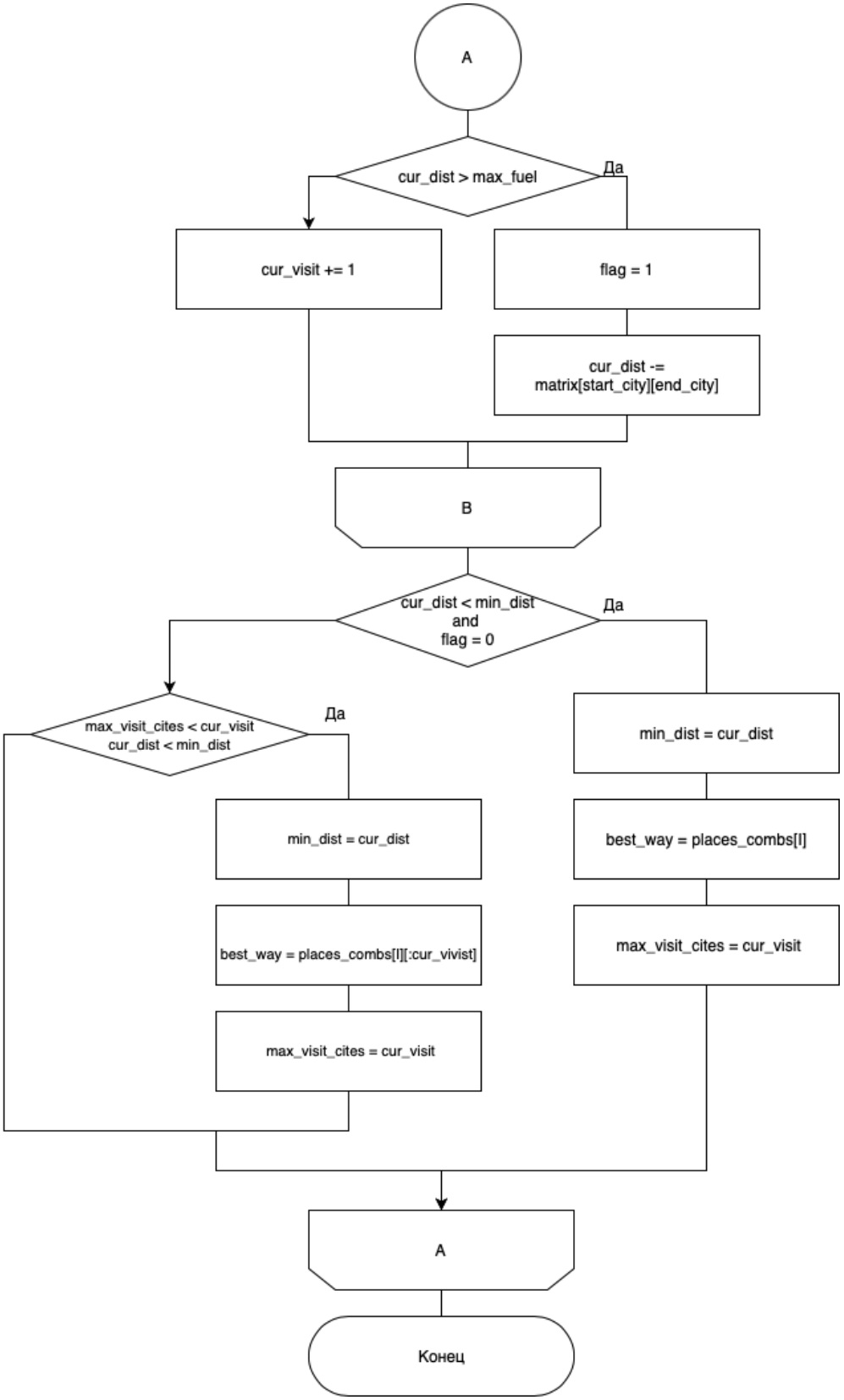

Рисунок 2.2 – Схема алгоритма полного перебора

Рисунок 2.3 – Схема муравьиного алгоритма

Рисунок 2.4 – Схема муравьиного алгоритма

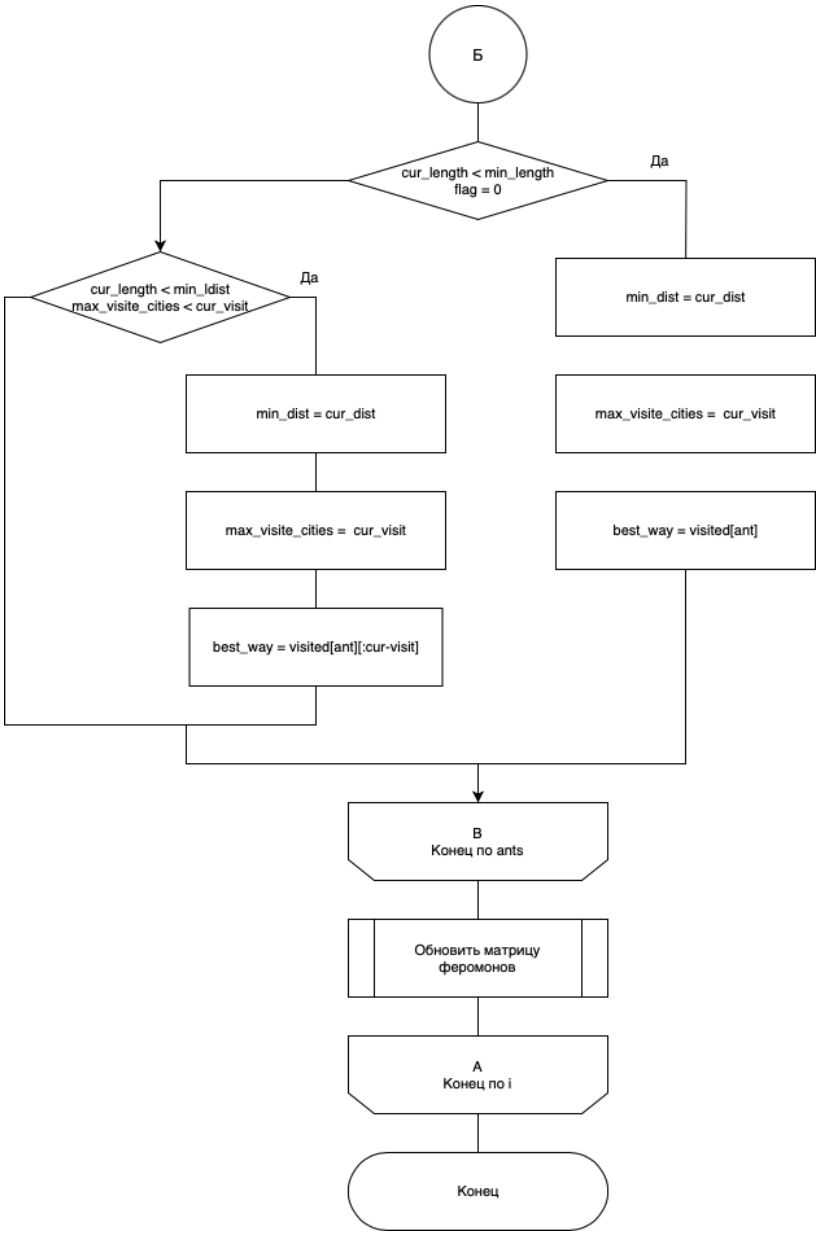

Рисунок 2.5 – Схема муравьиного алгоритма

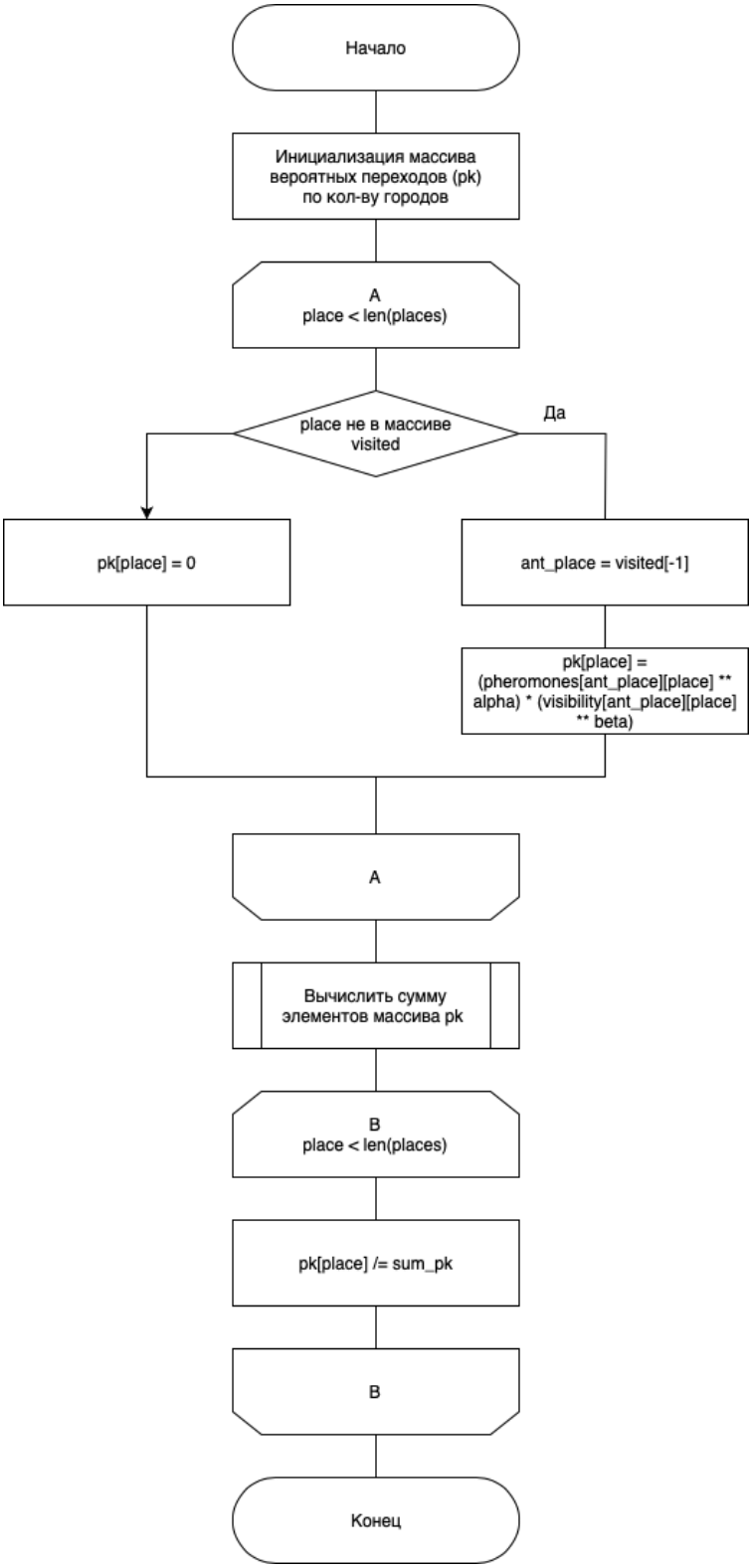

Рисунок 2.6 – Схема алгоритма нахождения массива вероятностных переходов в непосещенные города

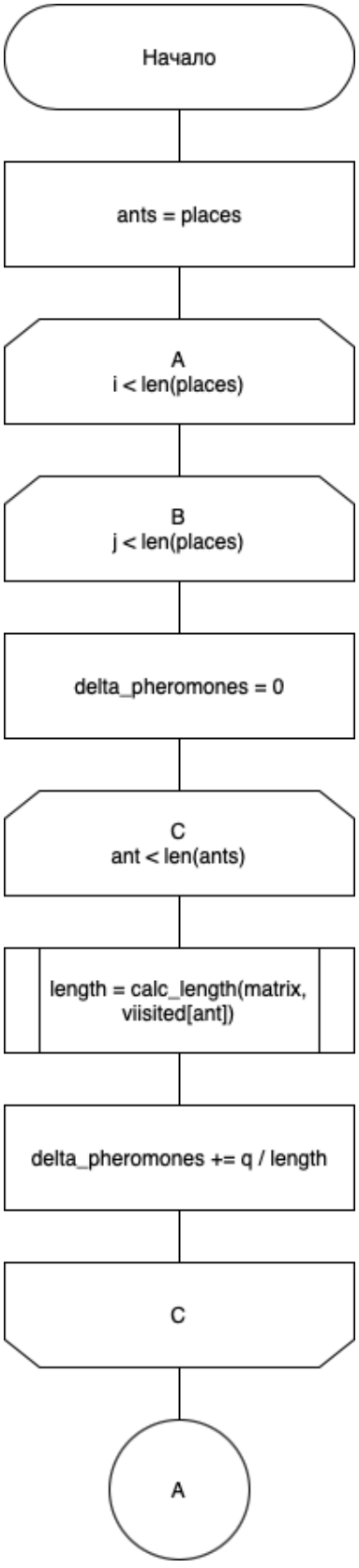

Рисунок 2.7 – Схема алгоритма обновления матрицы феромонов

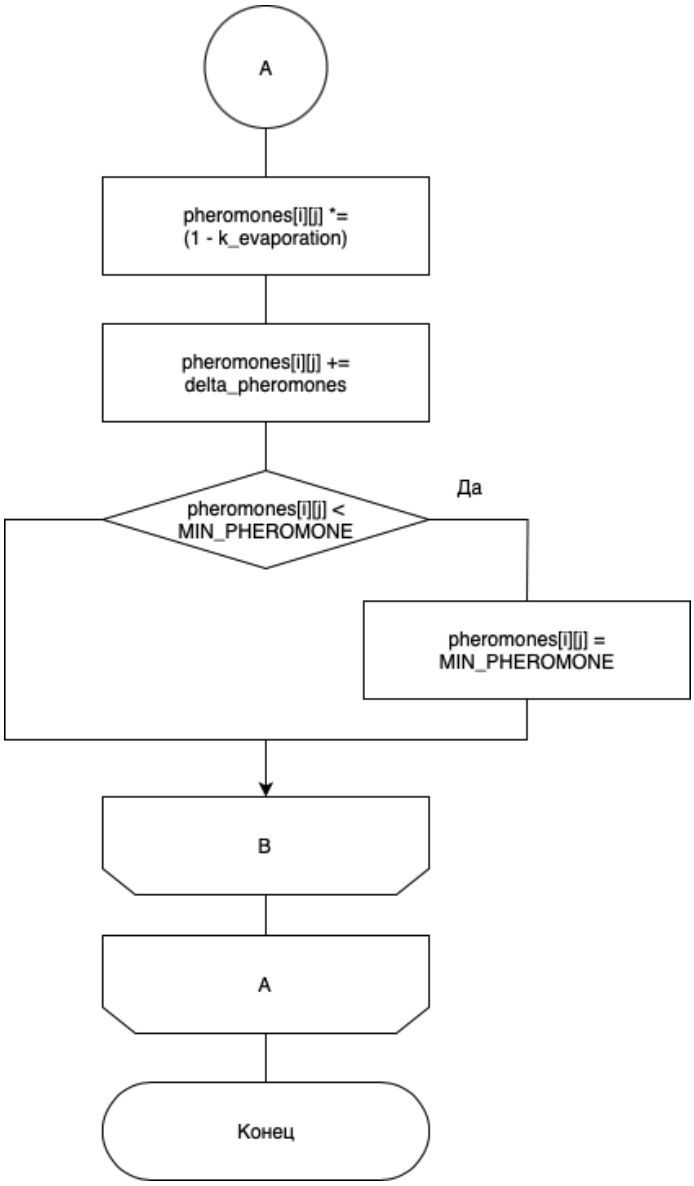

Рисунок 2.8 – Схема алгоритма обновления матрицы феромонов

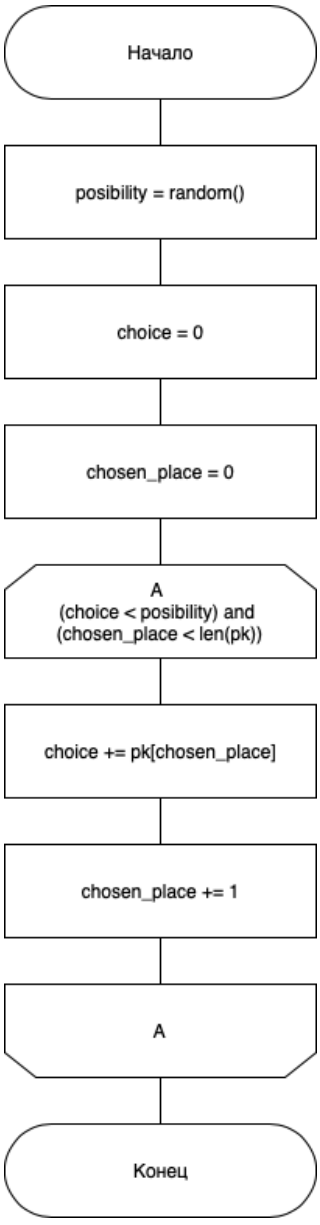

Рисунок 2.9 – Схема алгоритма нахождения следующего города на основании рандома

2.2 Структура разрабатываемого программного обеспечения

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

Вывод

В данном разделе были разработаны алгоритмы решения задачи коммивояжера, также была описана структура разрабатываемого программного обеспечения.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

— ввод имени файла, хранящего матрицу смежности графа;

— ввод коэффициентов для муравьиного алгоритма;

— ввод максимального количества топлива;

— вывод искомого пути и минимальной затраты;

— подсчет времени работы алгоритмов на различных значениях количества вершин;

— параметризация муравьиного алоритма с оценкой точности при заданных параметрах.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[3], так как 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* [4].

3.3 Реализация алгоритмов

В листинге 3.1 представлен алгоритм полного перебора путей, а в листингах 3.2-3.5 – муравьиный алгоритм и дополнительные к нему функции.

Листинг 3.1 – Релизация алгоритма полного пербора

```
1     def full_combinations(matrix, size, max_fuel):
2     places = np.arange(size)
3     places_combs = list()
4
5     for combination in it.permutations(places):
6         comb_arr = list(combination)
7         places_combs.append(comb_arr)
8
9     min_dist = float("inf")
10    max_visit_cites = 0
11
12    for i in range(len(places_combs)):
13        flag = 0
14        cur_dist = 0
15        cur_visit = 0
16
17        for j in range(size - 1):
18            start_city = places_combs[i][j]
19            end_city = places_combs[i][j + 1]
20
21            cur_dist += matrix[start_city][end_city]
22            if cur_dist > max_fuel:
23                flag = 1
24                cur_dist -= matrix[start_city][end_city]
25                break
26
27            cur_visit += 1
28
29        if (cur_dist < min_dist and flag == 0):
30            min_dist = cur_dist
31            best_way = places_combs[i]
32            max_visit_cites = cur_visit
33        elif (max_visit_cites < cur_visit and cur_dist <
              min_dist):
34            min_dist = cur_dist
```

```
35              max_visit_cites = cur_visit
36              best_way = places_combs[i][ : cur_visit]
37
38      return min_dist, best_way
```

Листинг 3.2 – Релизация муравьиного алгоритма

```
1       def ant_algorithm(matrix, places, alpha, beta,
            k_evaporation, days, max_fuel):
2       q = calc_q(matrix, places)
3
4       best_way = []
5       min_dist = float("inf")
6       max_visit_cites = 0
7
8       pheromones = get_pheromones(places)
9       visibility = get_visibility(matrix, places)
10
11      ants = places
12
13      for _ in range(days):
14          route = np.arange(places)
15          visited = get_visited_places(route, ants)
16
17          for ant in range(ants):
18              while len(visited[ant]) != places:
19                  pk = find_posibilities(pheromones, visibility,
                        visited, places, ant, alpha, beta)
20                  chosen_place =
                        choose_next_place_by_posibility(pk)
21                  visited[ant].append(chosen_place - 1)
22
23
24              cur_dist, cur_visit, flag  =
                    res_calc_length(matrix, visited[ant], max_fuel)
25
26              if (cur_dist < min_dist and flag == 0):
27                  min_dist = cur_dist
28                  best_way = visited[ant]
29                  max_visit_cites = cur_visit
30              elif (max_visit_cites < cur_visit and cur_dist <
                    min_dist):
```

```
31                  min_dist = cur_dist
32                  max_visit_cites = cur_visit
33                  best_way = visited[ant][ : cur_visit]
34
35          pheromones = update_pheromones(matrix, places, visited,
                pheromones, q, k_evaporation)
36
37      return min_dist, best_way
```

Листинг 3.3 – Алгоритм нахождения массива вероятностных переходов в непосещенные города

```
1 def find_posibilities(pheromones, visibility, visited, places,
      ant, alpha, beta):
2     pk = [0] * places
3
4     for place in range(places):
5         if place not in visited[ant]:
6             ant_place = visited[ant][-1]
7
8             pk[place] = pow(pheromones[ant_place][place],
                  alpha) * \
9                         pow(visibility[ant_place][place], beta)
10        else:
11            pk[place] = 0
12
13    sum_pk = sum(pk)
14
15    for place in range(places):
16        pk[place] /= sum_pk
17
18    return pk
```

Листинг 3.4 – Алгоритм нахождения следующего города на основании рандома

```
def choose_next_place_by_posibility(pk):
    posibility = random()
    choice, chosen_place = 0, 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

```
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
```

3.4 Функциональные тесты

В таблице 3.1 представлены функциональные тесты для алгоритма полного перебора и муравьиного алгоритма. Входными данными являются матрица смежностей и количество топлива, данное для осуществления маршрута. Выходными данными являются маршрут, который удалось пройти с минимальными затратами, и количество топлива, потраченного на данный маршрут.

Таблица 3.1 – Функциональные тесты

Входные данные	**Вывод**	**Ожидаемый результат**
$\begin{pmatrix} 0 & 4 & 2 & 1 & 7 \\ 4 & 0 & 3 & 7 & 2 \\ 2 & 3 & 0 & 10 & 3 \\ 1 & 7 & 10 & 0 & 9 \\ 7 & 2 & 3 & 9 & 0 \end{pmatrix}$, 16	8, [0, 2, 4, 1, 3]	8, [0, 2, 4, 1, 3]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$, 10	4, [0, 1, 2]	4, [0, 1, 2]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$, 30	27, [0, 1, 3]	27, [0, 1, 3]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$, 50	44, [0, 1, 2, 3]	44, [0, 1, 2, 3]

Алгоритмы, реализованные в данной лабораторной работе, функциональные тесты прошли успешно.

Вывод

В данном разделе были реализованы алгоритмы решения задачи комивояжера: полный перебор и муравьиный алгоритм. Также были написаны функциональные тесты для проверки правильности работы программы.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

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

— Процессор — 2 ГЦ 4-ядерный процессор Intel Core i5;

— Оперативная память — 16 ГБайт;

— Операционная система — macOS Venura 13.5.2.

4.2 Пример работы

В данном подразделе представлен пример 4.1 работы программы:

Листинг 4.1 – Демонстрация работы алгоритма

```
ivanmamvriyskiy@MacBook-Pro-Ivan-2 main % python3 main.py

    Меню
1. Полный перебор
2. Муравьиный алгоритм
3. Параметризация
4. Замерить время
0. Выход
Выбор:     1

Выберите файл: data/m1.txt
data/m1.txt
Введите максимально количетво топлива: 20

Минимально затраченное топливо = 8
Путь:  [1, 4, 2, 0, 3]
```

```
17
18          Меню
19      1. Полный перебор
20      2. Муравьиный алгоритм
21      3. Параметризация
22      4. Замерить время
23      0. Выход
24      Выбор:      2
25
26      Выберите файл: data/m1.txt
27      data/m1.txt
28
29      Введите коэффициент alpha: 0.7
30      Введите коэффициент evaporation: 0.5
31      Введите кол-во дней: 22
32      Введите максимально количетво топлива: 20
33
34      Минимально затраченное топливо = 8
35      Путь:  [1, 4, 2, 0, 3]
```

4.3 Время выполнения алгоритмов

В таблице 4.1 представлены замеры времени выполнения алгоритма полного перебора и муравьиного алгоритма. График зависимостей времени выполнения от размерности показан на рисунке 4.1. В таблице 4.2 представлена параметризация

Таблица 4.1 – Замеры времени выполнения алгоритма полного перебора и муравьиного алгоритма (с)

Размер	Полный перебор	Муравьиный алгоритм
2	0.000136	0.010161
3	0.000072	0.020081
4	0.000083	0.034927
5	0.002009	0.068276
6	0.001902	0.123055
7	0.015340	0.199969
8	0.137180	0.323262
9	1.402360	0.460636
10	15.593300	0.660567

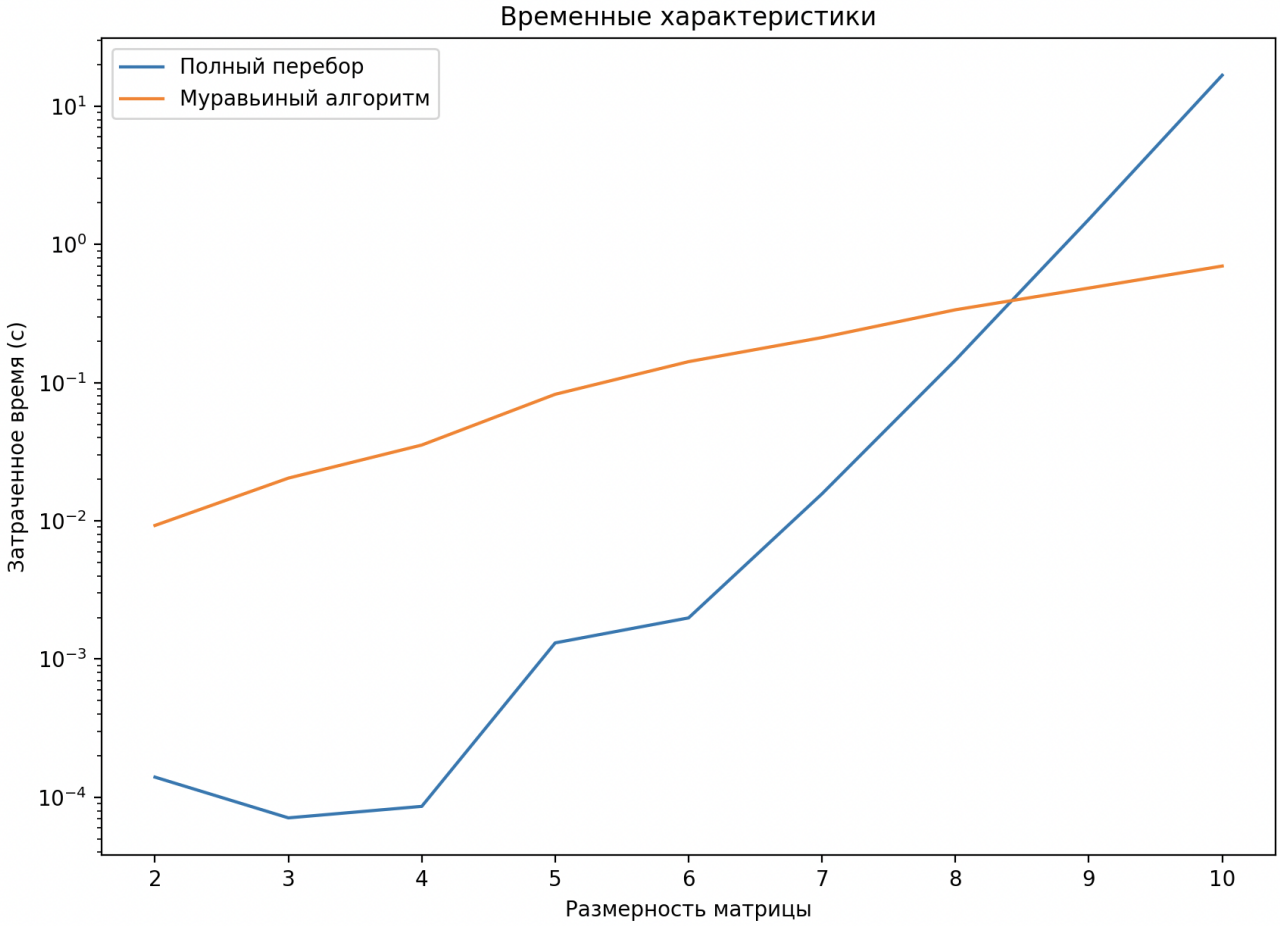

Рисунок 4.1 – График зависимости времени от размерности матрицы смежностей

Класс данных для параметризации представляет собой матрицу смежности с размерностью 9 4.1.

$$K_1 = \begin{pmatrix} 0 & 9271 & 8511 & 2010 & 1983 & 7296 & 7289 & 3024 & 1011 \\ 9271 & 0 & 7731 & 4865 & 5494 & 6812 & 4755 & 7780 & 7641 \\ 8511 & 7731 & 0 & 1515 & 9297 & 7506 & 5781 & 5804 & 7334 \\ 2010 & 4865 & 1515 & 0 & 3662 & 9597 & 2876 & 8188 & 9227 \\ 1983 & 5494 & 9297 & 3662 & 0 & 8700 & 4754 & 7445 & 3834 \\ 7296 & 6812 & 7506 & 9597 & 8700 & 0 & 4216 & 5553 & 8215 \\ 7289 & 4755 & 5781 & 2876 & 4754 & 4216 & 0 & 4001 & 4715 \\ 3024 & 7780 & 5804 & 8188 & 7445 & 5553 & 4001 & 0 & 9522 \\ 1011 & 7641 & 7334 & 9227 & 3834 & 8215 & 4715 & 9522 & 0 \end{pmatrix} \quad (4.1)$$

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

Таблица 4.2 – Параметры для класса данных K_1

Альфа	Бетта	Дней	Оптимальный	Ошибка
0.1	0.2	500	27523	0
0.1	0.4	300	27523	0
0.1	0.8	500	27523	0
0.2	0.3	500	27523	0
0.2	0.6	100	27523	0
0.2	0.8	50	27523	0
0.3	0.1	300	27523	0
0.3	0.4	500	27523	0
0.3	0.6	300	27523	0
0.3	0.6	500	27523	0
0.3	0.7	500	27523	0
0.3	0.8	50	27523	0
0.3	0.8	500	27523	0
0.4	0.1	300	27523	0
0.4	0.5	300	27523	0
0.5	0.1	500	27523	0
0.5	0.2	300	27523	0
0.5	0.4	500	27523	0
0.5	0.5	300	27523	0

4.4 Сравнение алгоритмов

В таблице 4.3 представлен сравнительный анализ алгоримта полного перебора и муравьиного перебора.

Таблица 4.3 – Сравнение алгоритма полного перебора с муравьиным алгоритмом

Методы / Критерии	Точность решения	Трудоемкость
Метод полного перебора	Гарантирует	O(n!)
Муравьиный алгоритм	Не гарантирует, так как присутствует случайный выбор начального маршрута и вероятностный выбор в процессе работы.	$3n^2 + n^4 + +m(3n + n^2 * *(n^2 + n^3))$

4.5 Вывод

По результатам проделанных экспериментов можно сделать следующие выводы:

1) при количестве вершин в графе до 8 алгоритм полного перебора работает быстрее муравьного алгоритма;

2) при количестве вершин в графе более 8 время выполнения алгоритма полного перебора резко возрастает, в то время как у муравьиного алгоритма при переходе на следующее количество вершин время возрастает не более чем в 1.5 раза;

3) лучшие результаты работы муравьиного алгоритма налюдались на парах значений:

   — $\alpha = 0.1, \beta = 0.2, 0.4, 0.8$;

   — $\alpha = 0.2, \beta = 0.3, 0.6, 0.8$;

   — $\alpha = 0.3, \beta = 0.1, 0.4, 0.6, 0.7, 0.8$;

- $\alpha = 0.4, \beta = 0.1, 0.5$;
- $\alpha = 0.5, \beta = 0.1, 0.2, 0.4, 0.5$.

Заключение

В результате исследования было определено, что при малых значениях количества вершин, до 8, следует использовать алгоримт полного перебора, так как он в любом случае дает точный результат. При количестве вершин больше 8 следует использовать муравьиный алгоритм, так как алгоритм полного перебора при таких значениях работает неопределенно долго.

При использовании муравьиного алгоритма необходимо выбирать оптимальные по результатам параметризации значения параметров.

Цель, которая заключается в исследовании метода решения задачи коммивояжера на базе муравьиного алгоритма, выполнена, также в ходе выполнения лабораторной работы были решены следующие задачи:

— описана задача коммивояжера;

— описаны алгоритмы решения задачи коммивояжера — полный перебор и муравьиный алгоритм;

— разработано программное обеспечение, реализующее данные алгоритмы;

— измеренно процессорное время работы данных алгоритмов;

— проведен сравнительный анализ по времени реализованных алгоритмов.

Список используемых источников

1. М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. ФИЗМАТЛИТ, 2008. с. 203.

2. Штовба С. Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. 2003. № 4. С. 70–75. URL: https://www.researchgate.net/publication/279535061_Stovba_SD_Muravinye_algoritmy_Exponenta_Pro_Matematika_v_prilozeniah_-_2003_-_No4_-_S_70-75(дата обращения: 05.12.2023).

3. Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 23.11.2021).

4. time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 27.11.2021).