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 Методы решения задачи коммивояжера

Студент Вавилова В. Л.

Группа ИУ7-54Б

Преподаватели Волкова Л. Л., Строганов Ю. В.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Задача коммивояжера 4

1.2 Полный перебор 4

1.3 Муравьиный алгоритм 4

2 Конструкторская часть **6**

2.1 Алгоритм поиска полным перебором в массиве 6

3 Технологическая часть **13**

3.1 Средства реализации 13

3.1.1 Алгоритм поиска решения задачи коммивояжера полным перебором 13

3.1.2 Алгоритм поиска решения задачи коммивояжера муравьиным алгоритмом . . 14

3.2 Тестовые данные 17

4 Исследовательская часть **19**

ЗАКЛЮЧЕНИЕ **21**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **22**

Приложение А **23**

ВВЕДЕНИЕ

Цель лабораторной работы — исследование методов решения задачи коммивояжера, включая анализ их преимуществ и недостатков, а также реализация и исследование эффективности алгоритмов полного перебора и муравьиного алгоритма.

Задачи лабораторной работы:

1) описать метод полного перебора для решения задачи коммивояжера;
2) разработать и реализовать алгоритм полного перебора;
3) описать математическую основу муравьиного алгоритма;
4) разработать и реализовать муравьиный алгоритм для задачи коммивояжера с учетом необнуления вероятности перехода в не посещенные города;
5) провести параметризацию муравьиного алгоритма по трем его параметрам, используя серию запусков на полносвязных графах;
6) организовать результаты исследования и параметризации, представить данные в виде таблиц.

1 Аналитическая часть

1.1 Задача коммивояжера

Задача коммивояжера является классической задачей оптимизации и комбинаторики и заключается в нахождении оптимального маршрута, который проходит через все указанные пункты (города) хотя бы по одному разу с последующим возвращением в исходный пункт (город) [1].

В общем случае задача формулируется на графе, где вершины соответствуют городам, а ребра (или дуги в ориентированном графе) имеют вес, определяющий расстояние, необходимое для перемещения между двумя городами. Цель состоит в минимизации общей длины (затрат) маршрута. Задача коммивояжера принадлежит к числу NP-трудных задач, что означает, что для ее точного решения в общем случае требуется экспоненциальное время при увеличении количества городов. Требуемое для решения задачи время пропорционально $(n-1)!$, где n — количество пунктов. Это делает применение переборных методов непрактичным для больших графов. Например, для задачи с числом городов более 50 нахождение оптимального маршрута потребовало бы вычислительной мощности компьютеров всего мира [1].

1.2 Полный перебор

Метод полного перебора подразумевает последовательный перебор всевозможных маршрутов, соединяющих вершины графа и выборе среди них маршрута с минимальной длиной. Для графа из N городов общее число возможных маршрутов равно $(N-1)!$, так как первый город фиксируется, а остальные $N-1$ города могут быть упорядочены в любом порядке. Этот метод проверяет все возможные комбинации маршрутов, поэтому при увеличении числа городов полное перечисление становится вычислительно неэффективным из-за экспоненциального роста числа возможных маршрутов.

1.3 Муравьиный алгоритм

Муравьиный алгоритм использует подход, основанный на поведении колонии муравьев. Каждый муравей ищет свой маршрут, опираясь на три основных фактора:

1) **зрение** — привлекательность перехода из города i в город j, которая рассчитывается как:

$$(\nu)_{ij} = \frac{1}{D_{ij}}, \tag{1.1}$$

где D_{ij} — длина ребра $i \to j$;

2) **память** — множество уже посещенных муравьем k в день t городов $J_k(t)$;

3) **обоняние** — концентрация феромона $\theta_{ij}(t)$ на ребре $i \to j$ в день t.

На рассвете каждый муравей начинает маршрут из уникального города. Вероятность выбора

города j из города i в день t рассчитывается следующим образом:

$$P_{k,ij}(t) = \begin{cases} 0, & \text{если } j \in J_k(t), \\ \frac{((\nu)_{ij})^{\alpha} \cdot (\theta_{ij}(t))^{\beta}}{\sum_{q \notin J_k(t)} ((\nu)_{iq})^{\alpha} \cdot (\theta_{iq}(t))^{\beta}}, & \text{если } j \notin J_k(t). \end{cases} \tag{1.2}$$

Параметры α и β задают баланс между жадностью (α) и стадностью (β) в решении.

После завершения всех маршрутов фаза ночи включает обновление феромонов на ребрах графа:

$$\theta_{ij}(t+1) = \theta_{ij}(t) \cdot (1 - - - \rho) + \Delta\theta_{ij}(t), \tag{1.3}$$

где $\rho \in (0, 1)$ — коэффициент испарения, а $\Delta\theta_{ij}(t)$ — добавление феромонов, определяемое как:

$$\Delta\theta_{ij}(t) = \sum_{k} \Delta\theta_{k,ij}(t), \tag{1.4}$$

$$\Delta\theta_{k,ij}(t) = \begin{cases} \frac{Q}{L_k(t)}, & \text{если муравей } k \text{ использовал ребро } i \to j, \\ 0, & \text{иначе.} \end{cases} \tag{1.5}$$

Здесь Q — дневная квота феромона, $L_k(t)$ — длина маршрута муравья k.

Модификация с элитными муравьями

Элитные муравьи усиливают наиболее успешные маршруты, дополнительно увеличивая концентрацию феромонов на их ребрах. Это реализуется добавлением дополнительной квоты $\Delta\theta_{elite,ij}(t)$, где лучшие маршруты выбираются на основе длины L_{best}.

Вывод

Была рассмотрена задача коммивояжера, которая из-за экспоненциального роста вычислительной сложности становится трудной для точного решения при большом числе городов. Метод полного перебора требует проверки всех возможных маршрутов, из-за чего невозможно его применение для графов с числом вершин более нескольких десятков. Альтернативой является муравьиный алгоритм, который моделирует поведение колонии муравьев. Этот алгоритм адаптируется в процессе работы, усиливая феромонные следы на ребрах, принадлежащих наиболее перспективным маршрутам. Таким образом, муравьиный алгоритм позволяет находить решение задачи коммивояжера с меньшими вычислительными затратами по сравнению с методами полного перебора.

2 Конструкторская часть

2.1 Алгоритм поиска полным перебором в массиве

На рисунках 2.1, 2.2 представлена схема алгоритма решения задачи коммивояжера полным перебором.

На рисунках 2.3, 2.4, 2.4 представлена схема алгоритма решения задачи коммивояжера муравьиным алгоритмом.

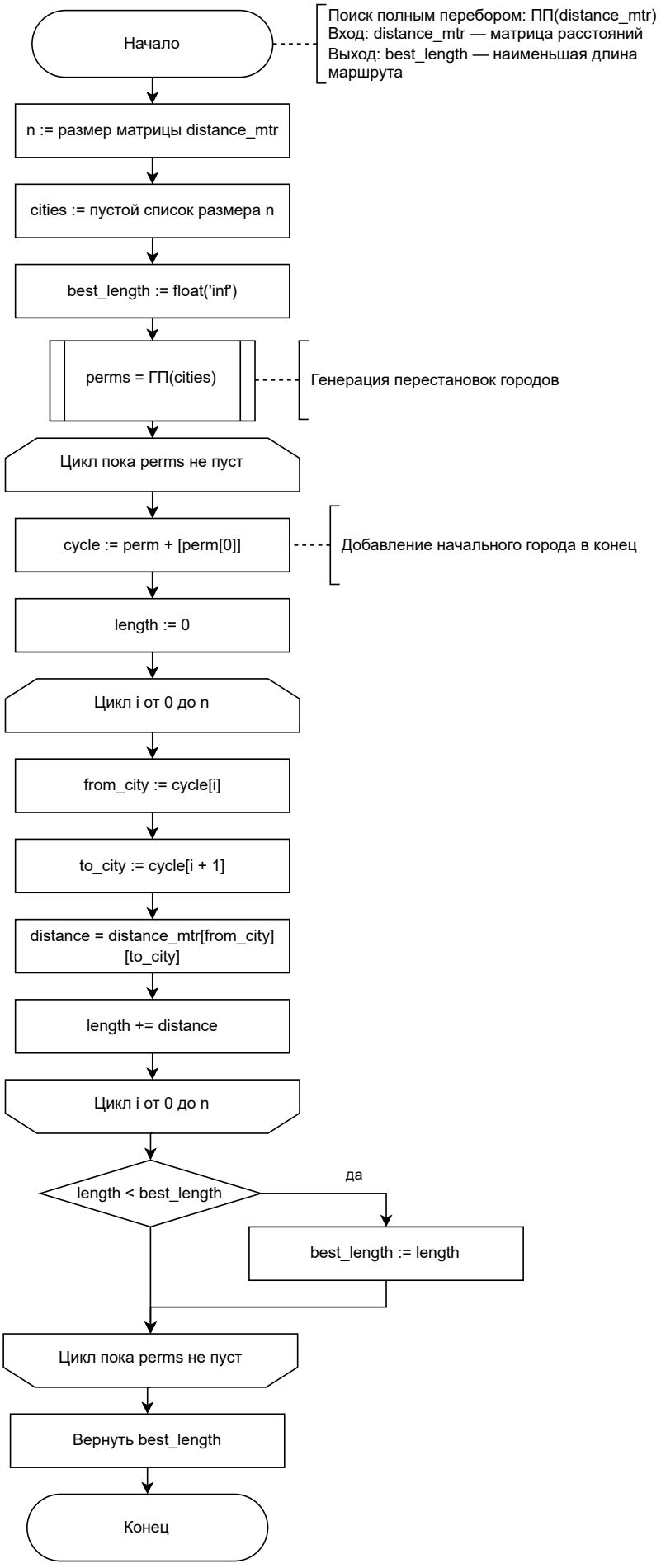

Рисунок 2.1 — Схема алгоритма решения полным перебором.

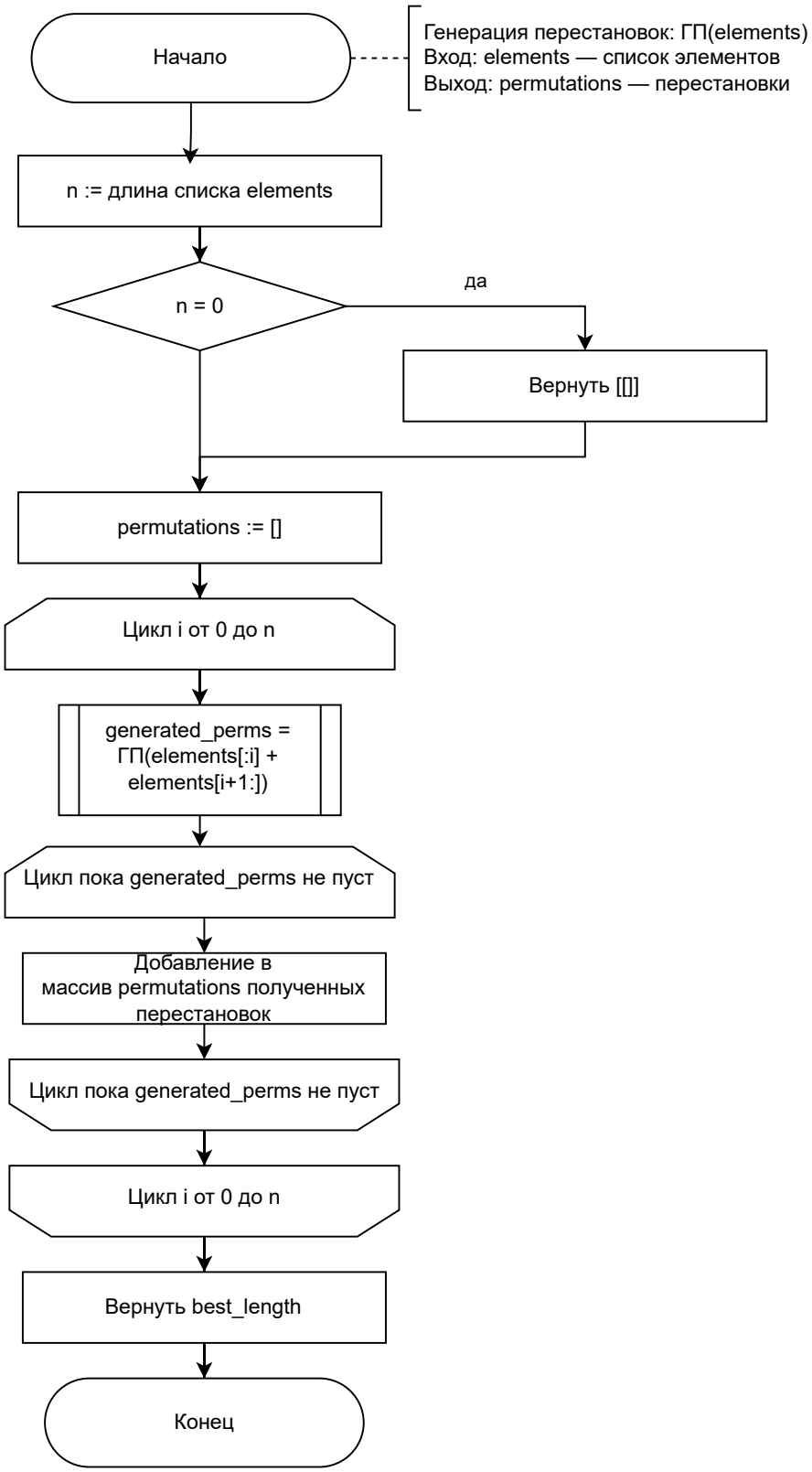

Рисунок 2.2 — Схема алгоритма генерации перестановок.

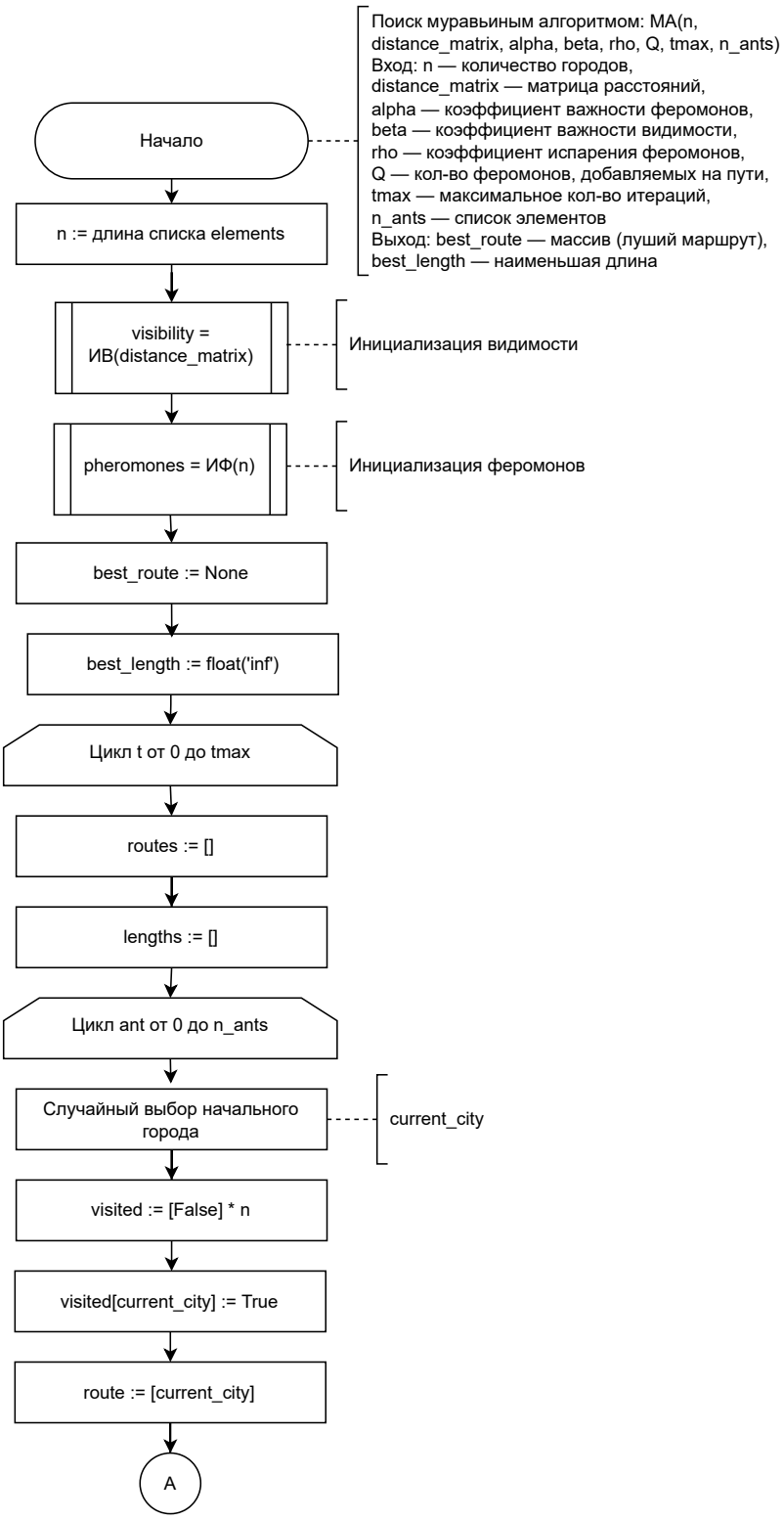

Рисунок 2.3 — Схема муравьиного алгоритма. Часть 2.

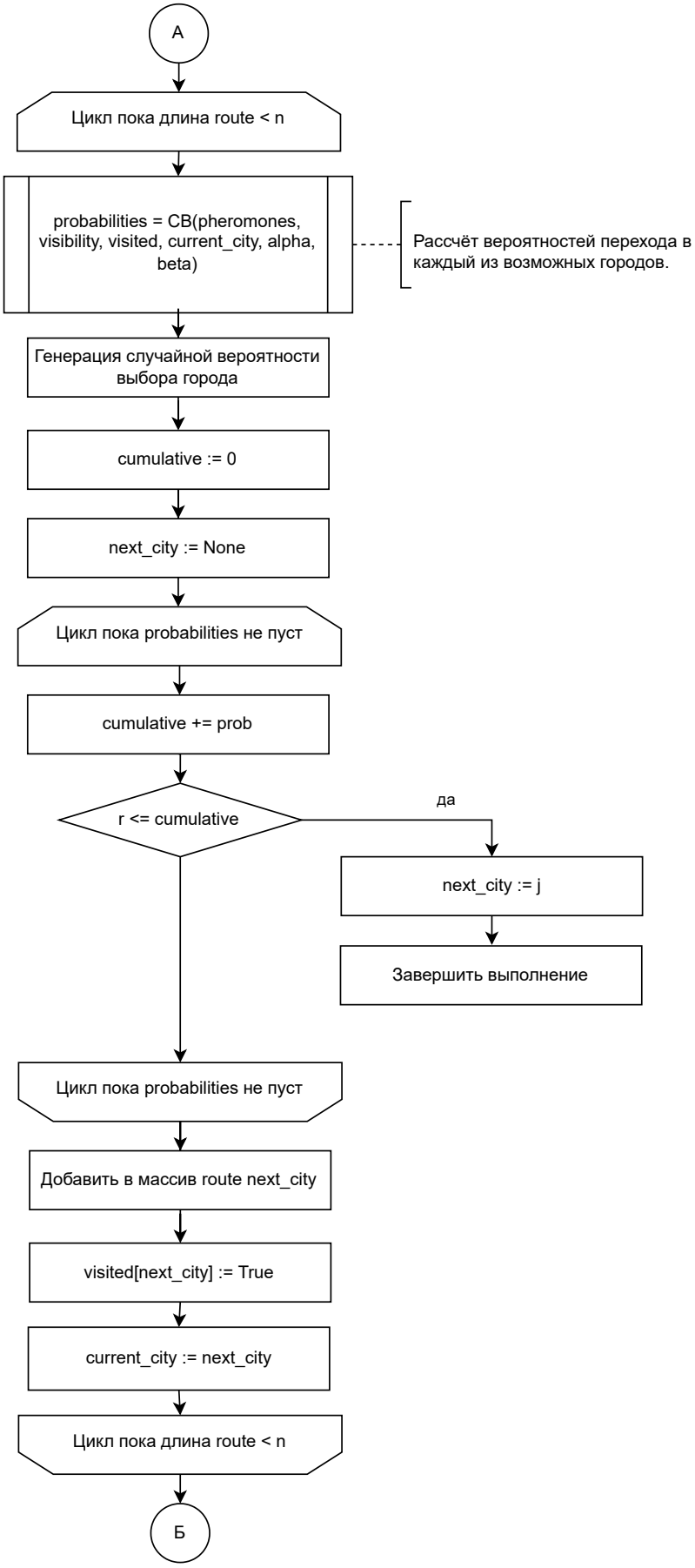

Рисунок 2.4 — Схема муравьиного алгоритма. Часть 2.

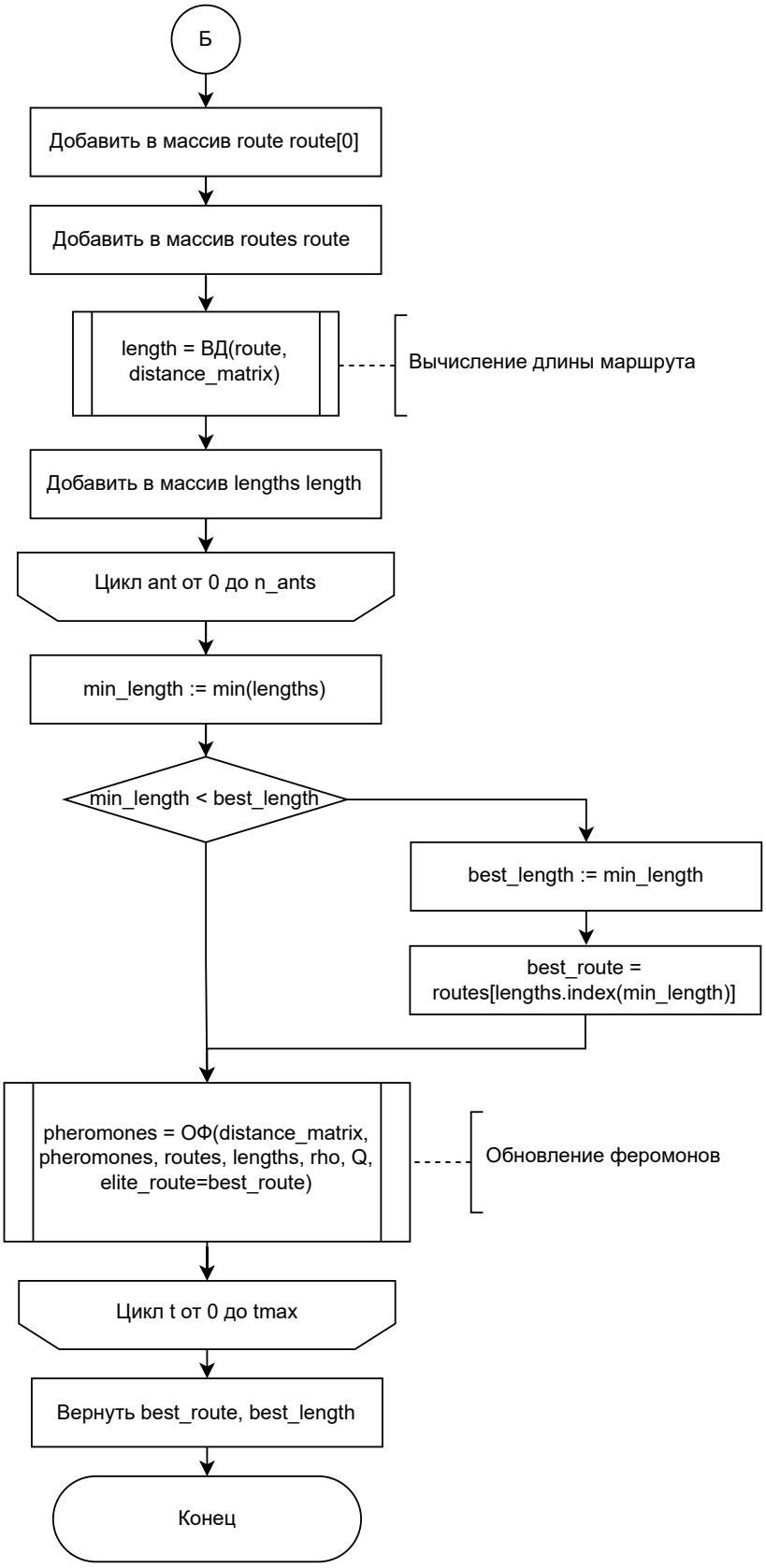

Рисунок 2.5 — Схема муравьиного алгоритма. Часть 3.

Вывод

На основе теоретических данных, полученных в аналитическом разделе, были разработаны схемы алгоритмов, необходимых для решения поставленных задач.

3 Технологическая часть

В данном разделе будут рассмотрены средства, использованные для разработки, а также приведены коды реализованных алгоритмов.

3.1 Средства реализации

Python был выбран, потому что он предлагает все необходимые инструменты для реализации алгоритмов.

3.1.1 Алгоритм поиска решения задачи коммивояжера полным перебором

В листинге 3.1 представлен алгоритм поиска решения задачи коммивояжера полным перебором.

```
1  def full_check(distance_matrix):
2      n = len(distance_matrix)
3      cities = list(range(n))
4      best_length = float('inf')
5
6      perms = generate_permutations(cities)
7      for perm in perms:
8          cycle = perm + [perm[0]]
9
10         length = 0
11         for i in range(n):
12             from_city = cycle[i]
13             to_city = cycle[i + 1]
14             distance = distance_matrix[from_city][to_city]
15             length += distance
16
17         if length < best_length:
18             best_length = length
19
20     return best_length
```

Листинг 3.1 — Алгоритм поиска решения задачи коммивояжера полным перебором

В листинге 3.2 представлен алгоритм генерации всех возможных перестановки элементов входного списка.

```
1  def generate_permutations(elements):
2      n = len(elements)
```

```
    if n == 0:
        return [[]]

    permutations = []
    for i in range(n):
        generated_permutations = generate_permutations(elements[:i] +
            elements[i+1:])
        for perm in generated_permutations:
            permutations.append([elements[i]] + perm)
    return permutations
```

Листинг 3.2 — Алгоритм генерации всех возможных перестановки элементов входного списка

3.1.2 Алгоритм поиска решения задачи коммивояжера муравьиным алгоритмом

В листинге 3.3 представлен алгоритм поиска решения задачи коммивояжера муравьиным алгоритмом.

```
def ant_colony_optimization(n, distance_matrix, alpha, beta, rho, Q, tmax,
    n_ants):
    visibility = initialize_visibility(distance_matrix)
    pheromones = initialize_pheromones(n)

    best_route = None
    best_length = float('inf')

    for t in range(tmax):
        routes = []
        lengths = []

        for ant in range(n_ants):
            current_city = random.randint(0, n - 1)
            visited = [False] * n
            visited[current_city] = True
            route = [current_city]

            while len(route) < n:
                probabilities = calculate_probabilities(pheromones,
                    visibility, visited, current_city, alpha, beta)
                r = random.random()
                cumulative = 0
                next_city = None
                for j, prob in enumerate(probabilities):
                    cumulative += prob
```

```
                    if r <= cumulative + 1e-5:
                        next_city = j
                        break

                route.append(next_city)
                visited[next_city] = True
                current_city = next_city

            route.append(route[0])
            routes.append(route)

            length = calculate_route_length(route, distance_matrix)
            lengths.append(length)

        min_length = min(lengths)
        if min_length < best_length:
            best_length = min_length
            best_route = routes[lengths.index(min_length)]

        pheromones = update_pheromones(distance_matrix, pheromones, routes,
            lengths, rho, Q, elite_route=best_route)

    return best_route, best_length
```

Листинг 3.3 — Алгоритм поиска решения задачи коммивояжера муравьиным алгоритмом

В листинге 3.4 представлен алгоритм инициализации матрицы видимости.

```
def initialize_visibility(distance_matrix):
    n = len(distance_matrix)
    visibility = []

    for i in range(n):
        row = []
        for j in range(n):
            if distance_matrix[i][j] > 0:
                visibility_value = 1 / distance_matrix[i][j]
            else:
                visibility_value = 0
            row.append(visibility_value)
        visibility.append(row)
    return visibility
```

Листинг 3.4 — Алгоритм инициализации матрицы видимости

В листинге 3.5 представлен алгоритм вычисления вероятности перехода из текущего города в другие города.

```
def calculate_probabilities(pheromones, visibility, visited, current_city,
    alpha, beta):
    n = len(pheromones)
    probabilities = [0] * n
    for j in range(n):
        if not visited[j]:
            probabilities[j] = (pheromones[current_city][j] ** alpha) *
                (visibility[current_city][j] ** beta)
    total = sum(probabilities)
    if total == 0:
        probabilities = [1 if not visited[j] else 0 for j in range(n)]
        total = sum(probabilities)
    return [p / total for p in probabilities]
```

Листинг 3.5 — Алгоритм вычисления вероятности перехода из текущего города в другие города

В листинге 3.6 представлен алгоритм вычисления длины маршрута.

```
def calculate_route_length(route, distance_matrix):
    length = 0
    for i in range(len(route) - 1):
        from_city = route[i]
        to_city = route[i + 1]
        distance = distance_matrix[from_city][to_city]
        length += distance
    return length
```

Листинг 3.6 — Алгоритм вычисления вычисления длины маршрута

В листинге 3.7 представлен алгоритм обновления феромонов.

```
def update_pheromones(distance_matrix, pheromones, routes, lengths, rho, Q,
    elite_route=None):
    n = len(pheromones)
    for i in range(n):
        for j in range(n):
            pheromones[i][j] *= (1 - rho)

    for route, length in zip(routes, lengths):
        for i in range(len(route) - 1):
            pheromones[route[i]][route[i + 1]] += Q / length

```

```
    if elite_route:
        elite_length = sum(distance_matrix[elite_route[i]][elite_route[i +
            1]] for i in range(len(elite_route) - 1))
        for i in range(len(elite_route) - 1):
            pheromones[elite_route[i]][elite_route[i + 1]] += Q / elite_length

    return pheromones
```

Листинг 3.7 — Алгоритм обновления феромонов

3.2 Тестовые данные

Используемые матрицы расстояний

1. **Матрица 1×1**

$$\text{distance\_matrix\_1} = \begin{bmatrix} 0 \end{bmatrix}$$

2. **Матрица 3×3**

$$\text{distance\_matrix\_3} = \begin{bmatrix} 0 & 10 & 15 \\ 10 & 0 & 20 \\ 15 & 20 & 0 \end{bmatrix}$$

3. **Матрица 5×5**

$$\text{distance\_matrix\_5} = \begin{bmatrix} 0 & 12 & 10 & 29 & 15 \\ 12 & 0 & 8 & 30 & 18 \\ 10 & 8 & 0 & 25 & 16 \\ 29 & 30 & 25 & 0 & 22 \\ 15 & 18 & 16 & 22 & 0 \end{bmatrix}$$

Результаты тестирования

Результаты тестирования приведены в таблице 3.1.

Таблица 3.1 — Результаты тестирования

№ теста	Размер матрицы	Алгоритм	Ожидаемый рез-т	Фактический рез-т
1	1×1	Полный перебор	0, [0]	0, [0]
2	1×1	Муравьиный алгоритм	0, [0]	0, [0]
3	3×3	Полный перебор	45, [0, 1, 2, 0]	45, [0, 1, 2, 0]
4	3×3	Муравьиный алгоритм	45, [0, 1, 2, 0]	45, [0, 1, 2, 0]
5	5×5	Полный перебор	88, [0, 2, 1, 4, 3, 0]	88, [0, 2, 1, 4, 3, 0]
6	5×5	Муравьиный алгоритм	88, [0, 2, 1, 4, 3, 0]	88, [0, 2, 1, 4, 3, 0]

Все тесты пройдены успешно.

Вывод

Были разработаны и протестированы реализованные алгоритмы.

4 Исследовательская часть

Технические характеристики устройства:

— процессор: Intel(R) Core(TM) i5-10300H с тактовой частотой 2500000000 герц;

— ядра: 4;

— логические процессоры: 8;

— оперативная система: Windows 10;

— оперативная память: 8 ГБ.

Параметризация была выполнена для одного набора данных, состоящего из трех матриц смежности размера 9x9. Матрицы состоят из координат городов древнего мира. Используемые города приведены в таблицах 4.1, 4.2 и 4.3.

Таблица 4.1 — Таблица городов 1

Название	**Широта**	**Долгота**
Вавилон	32.536	44.420
Мемфис	29.849	31.254
Фивы (Египет)	25.718	32.645
Иерусалим	31.768	35.213
Афины	37.984	23.728
Дамаск	33.513	36.292
Урарту	38.501	43.370
Мохенджо-Даро	27.327	68.132
Харран	36.867	39.031

Таблица 4.2 — Таблица городов 2

Название	**Широта**	**Долгота**
Рим	41.902	12.496
Карфаген	36.853	10.323
Александрия	31.200	29.918
Персеполь	29.934	52.891
Ниневия	36.360	43.150
Троя	39.957	26.238
Спарта	37.075	22.429
Сузиана	32.194	48.247
Кносс	35.299	25.162

Таблица 4.3 — Таблица городов 3

Название	**Широта**	**Долгота**
Эфес	37.941	27.341
Тир	33.273	35.196
Коринф	37.905	22.934
Пальмира	34.551	38.276
Угарит	35.595	35.782
Сидон	33.561	35.375
Гиза	29.978	31.134
Луксор	25.687	32.639
Тартесс	37.191	-6.929

Результаты параметризации приведены в таблице приложения А. Таблица 4.4 содержит результаты параметризации для оптимальных параметров.

Таблица 4.4 — Результаты исследования

			Граф 1			Граф 2			Граф 3		
α	p	**tmax**	**max**	**med**	**avg**	**max**	**med**	**avg**	**max**	**med**	**avg**
0.5	0.5	150	177	0	35	0	0	0	0	0	0
0.5	0.75	200	177	0	17	0	0	0	0	0	0
0.5	0.9	200	177	0	35	0	0	0	0	0	0
0.75	0.5	200	296	0	70	0	0	0	0	0	0
0.75	0.9	150	234	0	41	0	0	0	0	0	0
0.9	0.9	200	216	0	39	0	0	0	5	0	0

Вывод

В результате исследовательской части было проведено исследование производительности алгоритма на различных наборах параметров. Параметризация алгоритма позволила выделить оптимальные настройки. Лучшими значениями параметров для достижения высокой точности являются: $\alpha = 0.5$, $p = 0.75$, и максимальное количество итераций $t_{\max} = 200$, что подтверждается данными из таблицы 4.4.

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы была достигнута — исследованы методы решения задачи коммивояжера, а также выполнен анализ их преимуществ и недостатков.

В ходе работы были выполнены следующие задачи:

1) исследован и реализован метод полного перебора;
2) описана математическая основа муравьиного алгоритма;
3) разработан и реализован муравьиный алгоритм с учетом особенностей перехода между городами;
4) проведена параметризация муравьиного алгоритма по параметрам α, p, и числу итераций, результаты оформлены в таблицы и графики;

В результате проведенного исследования установлено, что метод полного перебора обеспечивает точное решение задачи коммивояжера, однако его применение ограничено из-за высокой вычислительной сложности. Муравьиный алгоритм, напротив, демонстрирует высокую эффективность на больших графах при условии правильной настройки параметров. Оптимальные параметры алгоритма были установлены путем расчетов и анализа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Володина Е. В., Студентова Е. А. Практическое применение алгоритма решения задачи коммивояжера // Электронный научный журнал «Инженерный вестник Дона». — 2007–2015. — № 2, ч. 2 (2015). — Режим доступа: ivdon.ru/ru/magazine/archive/n2p2y2015/3074, свободный.

Приложение А

Результаты параметризации муравьиного алгоритма.