



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 коммивояжёра | 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 коммивояжёра, 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 коммивояжёра;
- 2) разработать и реализовать алгоритм полного перебора;
- 3) описать математическую основу муравьиного алгоритма;
- 4) разработать и реализовать муравьиный алгоритм для задачи коммивояжёра с учётом ненулевой вероятности перехода в не посещённые города;
- 5) провести параметризацию муравьиного алгоритма по трём 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 коммивояжёра

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 рёбра (или дуги в ориентированном графе) имеют вес, определяющий расстояние, необходимое для перемещения между двумя городами. Цель состоит в минимизации общей длины (затрат) маршрута. Задача коммивояжёра принадлежит к числу NP-трудных задач, что означает, что для её 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 муравьёв. Каждый муравей ищет свой маршрут, опираясь на три основных фактора:

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

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

где D_{ij} — длина ребра $i \rightarrow j$.

2. **Память** — множество уже посещённых муравьём k в день t городов $J_k(t)$.

3. **Обоняние** — концентрация феромона $\theta_{ij}(t)$ на ребре $i \rightarrow j$ 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:

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

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

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

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

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

$$\Delta\theta_{ij}(t) = \sum_k \Delta\theta_{k,ij}(t),$$

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

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 рёбрах. Это реализуется добавлением дополнительной квоты $\Delta\theta_{elite,ij}(t)$, где лучшие маршруты выбираются на основе длины L_{best} .

Вывод

Была рассмотрена задача коммивояжёра, 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 муравьёв. Этот алгоритм адаптируется в процессе работы, усиливая феромонные следы на рёбрах, принадлежащих наиболее перспективным маршрутам. Таким образом, муравьиный алгоритм позволяет находить решение задачи коммивояжёра 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.

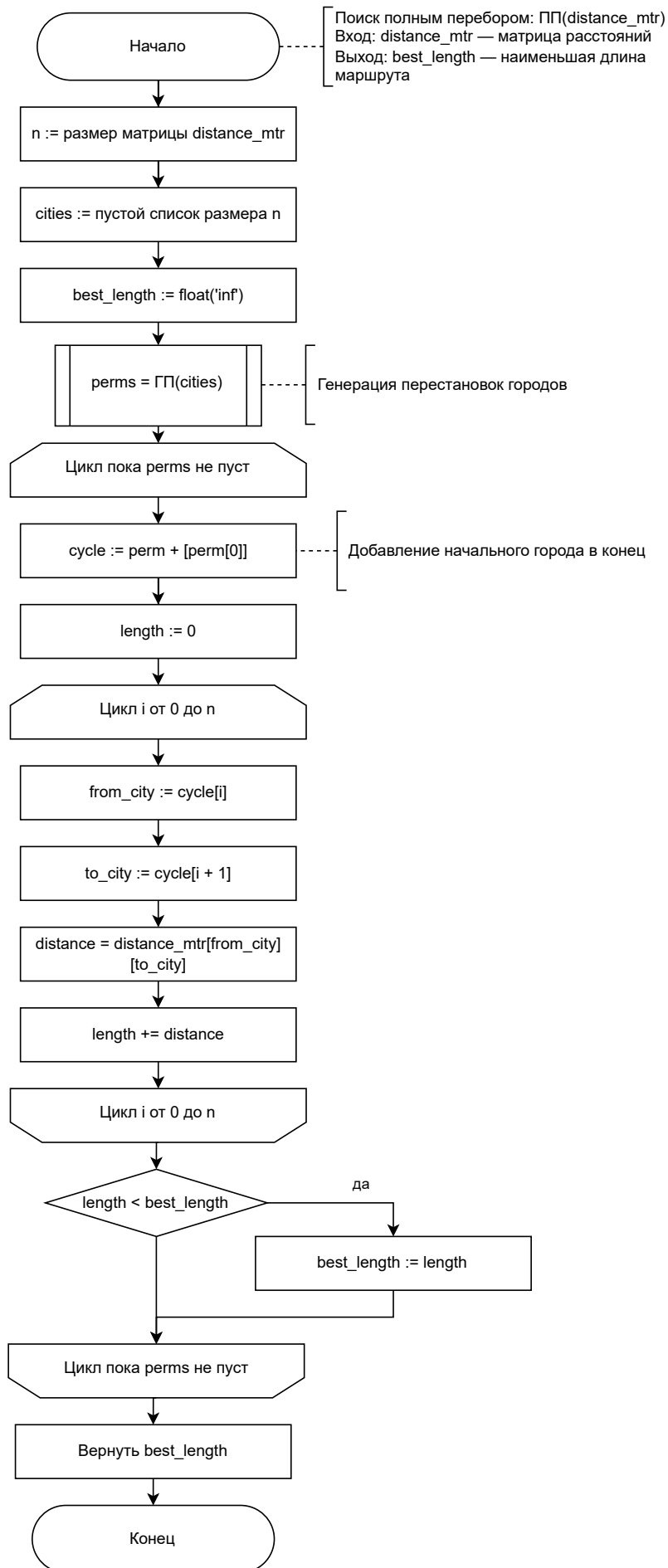


Рисунок 2.1 — Схема алгоритма решения полным перебором.

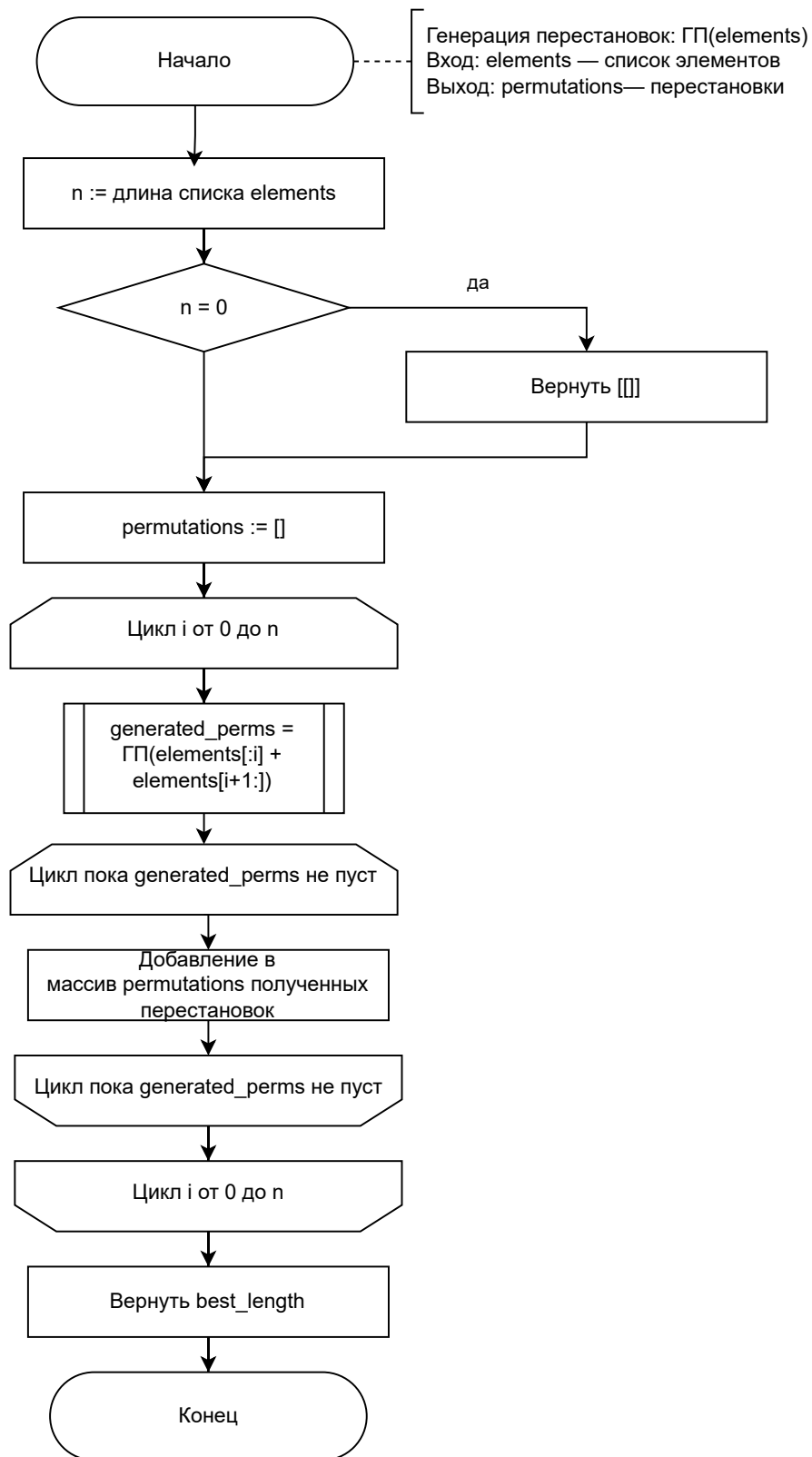


Рисунок 2.2 — Схема алгоритма генерации перестановок.

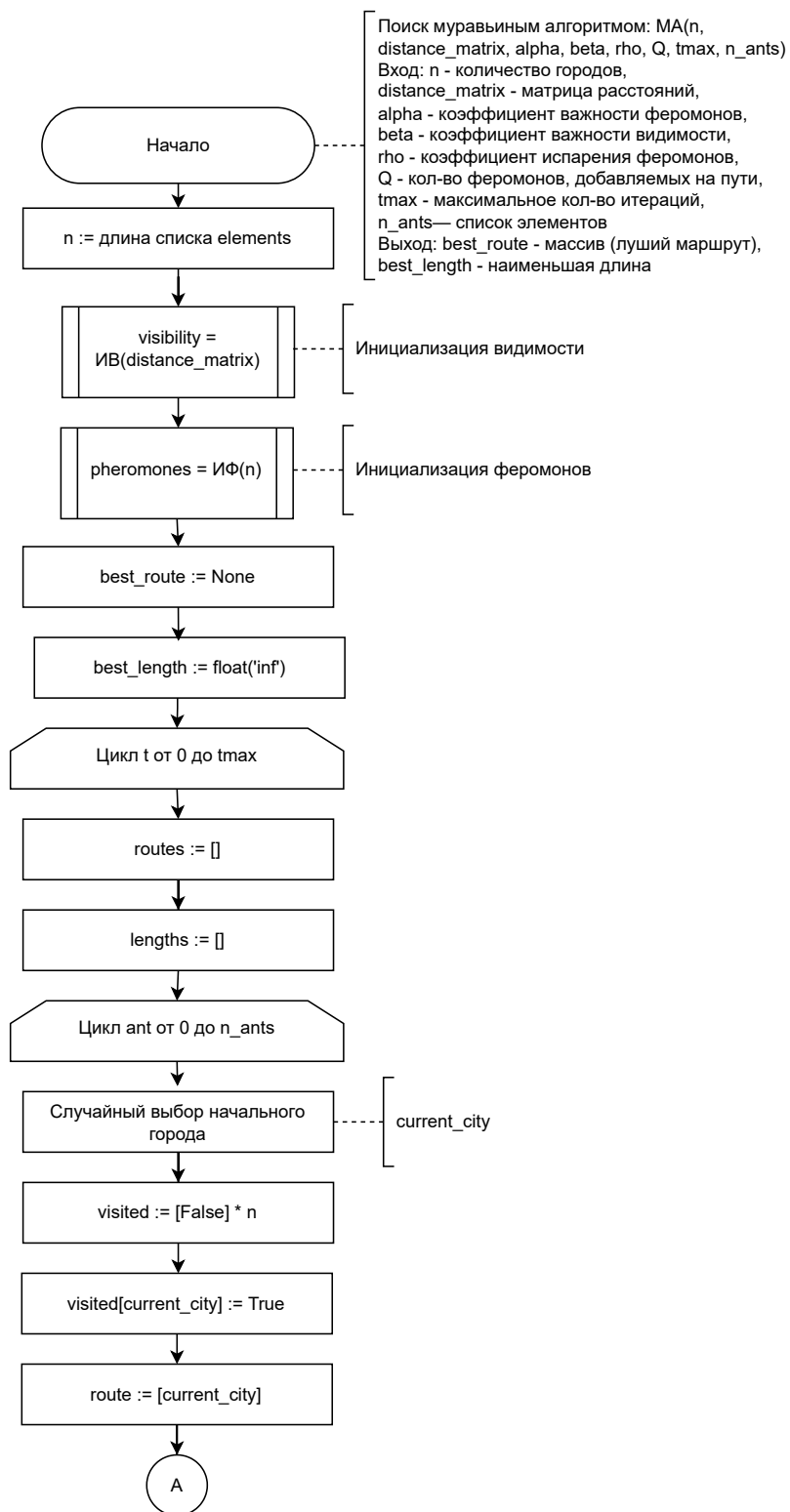


Рисунок 2.3 — Схема муравьиного алгоритма. Часть 2.

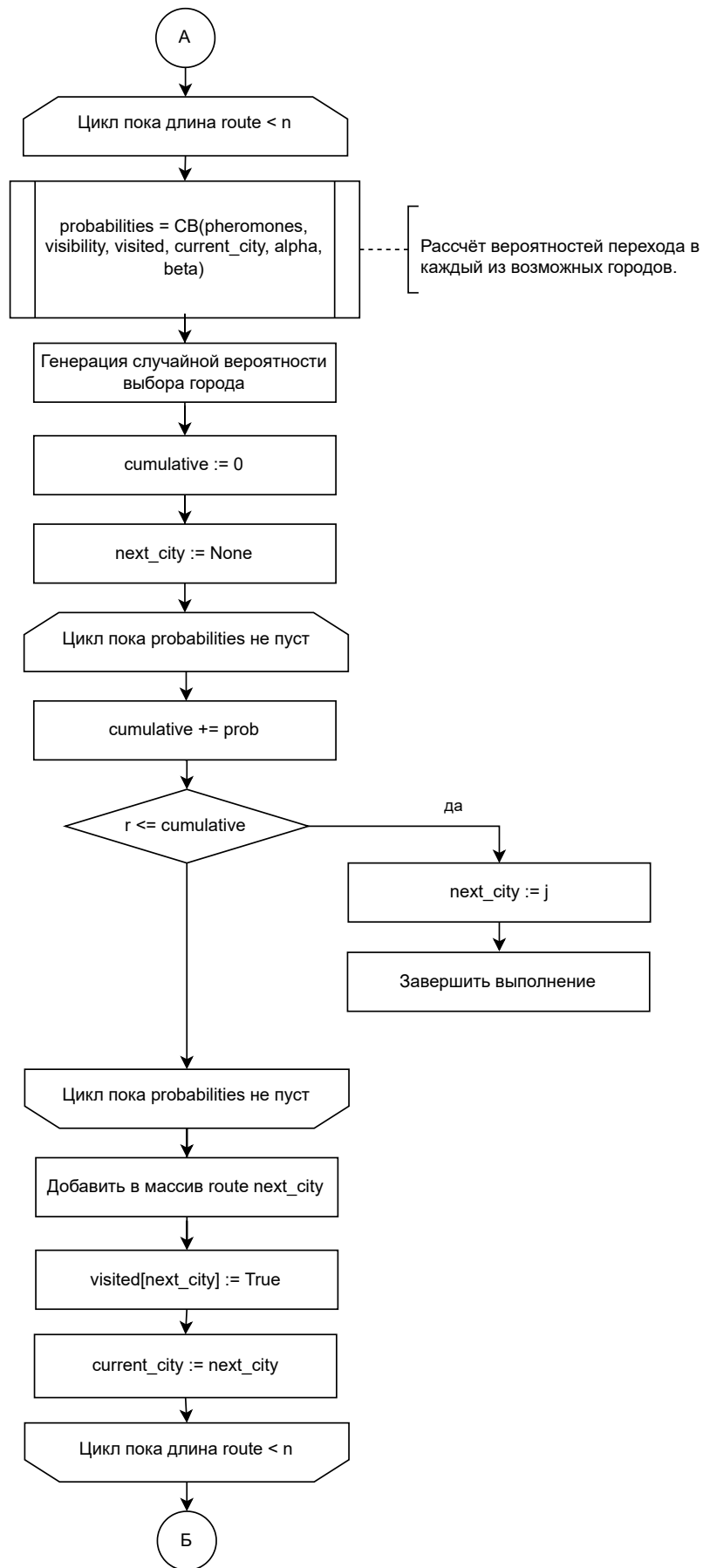


Рисунок 2.4 — Схема муравьиного алгоритма. Часть 2.

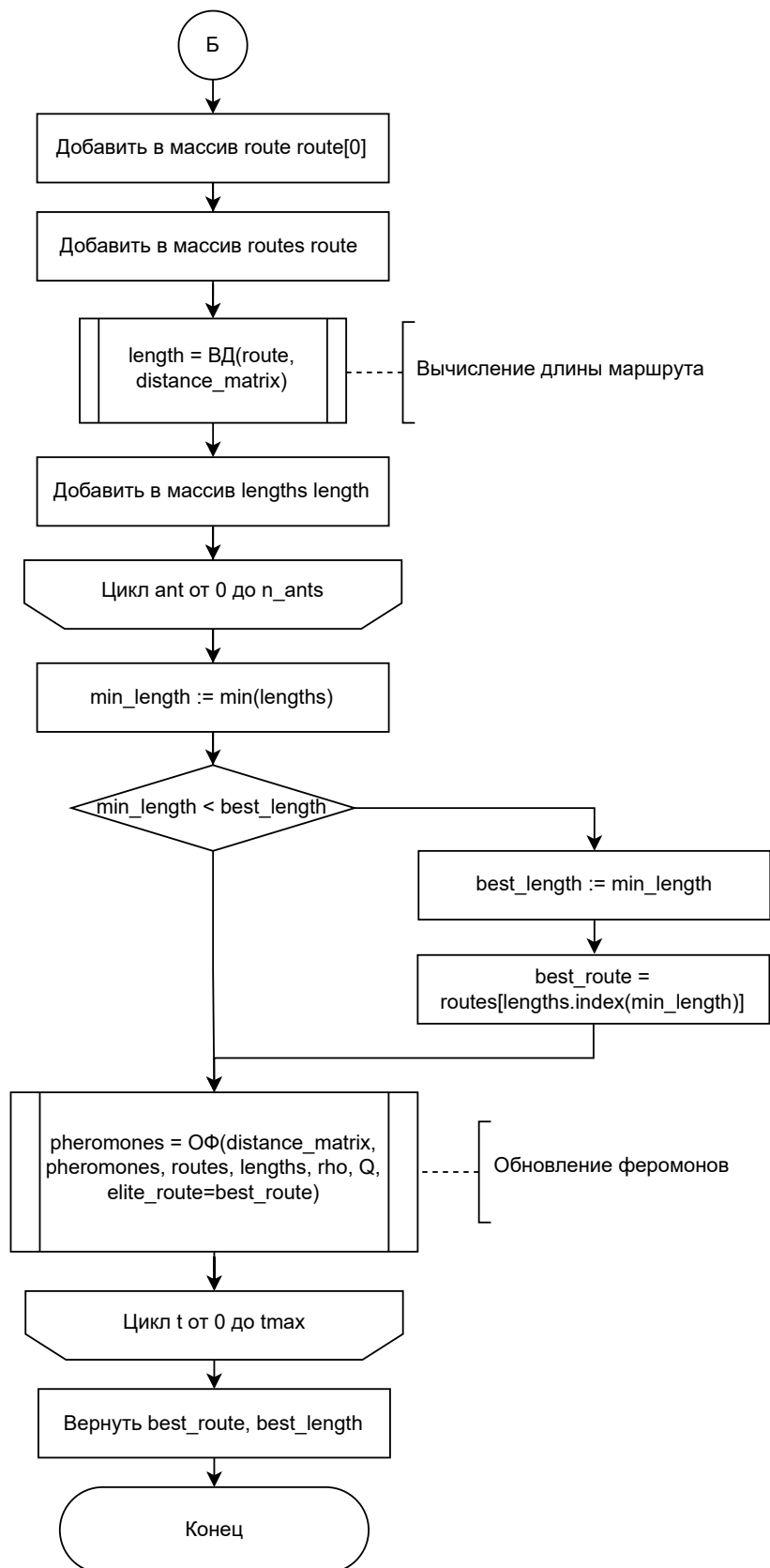


Рисунок 2.5 — Схема муравьиного алгоритма. Часть 3.

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)
```

```

3     if n == 0:
4         return [[]]
5
6     permutations = []
7     for i in range(n):
8         generated_permutations = generate_permutations(elements[:i] +
9             elements[i+1:])
10        for perm in generated_permutations:
11            permutations.append([elements[i]] + perm)
12    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.

```

1  def ant_colony_optimization(n, distance_matrix, alpha, beta, rho, Q, tmax,
2     n_ants):
3     visibility = initialize_visibility(distance_matrix)
4     pheromones = initialize_pheromones(n)
5
6     best_route = None
7     best_length = float('inf')
8
9     for t in range(tmax):
10        routes = []
11        lengths = []
12
13        for ant in range(n_ants):
14            current_city = random.randint(0, n - 1)
15            visited = [False] * n
16            visited[current_city] = True
17            route = [current_city]
18
19            while len(route) < n:
20                probabilities = calculate_probabilities(pheromones,
21                    visibility, visited, current_city, alpha, beta)
22                r = random.random()
23                cumulative = 0
24                next_city = None
25                for j, prob in enumerate(probabilities):
26                    cumulative += prob

```

```

25         if r <= cumulative + 1e-5:
26             next_city = j
27             break
28
29             route.append(next_city)
30             visited[next_city] = True
31             current_city = next_city
32
33             route.append(route[0])
34             routes.append(route)
35
36             length = calculate_route_length(route, distance_matrix)
37             lengths.append(length)
38
39             min_length = min(lengths)
40             if min_length < best_length:
41                 best_length = min_length
42                 best_route = routes[lengths.index(min_length)]
43
44             pheromones = update_pheromones(distance_matrix, pheromones, routes,
45                                             lengths, rho, Q, elite_route=best_route)
46
47     return best_route, best_length

```

Листинг 3.3 — Алгоритм поиска решения задачи коммивояжера муравьиным алгоритмом

В листинге 3.4 представлен алгоритм инициализации матрицы видимости.

```

1  def initialize_visibility(distance_matrix):
2      n = len(distance_matrix)
3      visibility = []
4
5      for i in range(n):
6          row = []
7          for j in range(n):
8              if distance_matrix[i][j] > 0:
9                  visibility_value = 1 / distance_matrix[i][j]
10             else:
11                 visibility_value = 0
12             row.append(visibility_value)
13         visibility.append(row)
14     return visibility

```

Листинг 3.4 — Алгоритм инициализации матрицы видимости

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

```
1 def calculate_probabilities(pheromones, visibility, visited, current_city,
2   alpha, beta):
3     n = len(pheromones)
4     probabilities = [0] * n
5     for j in range(n):
6         if not visited[j]:
7             probabilities[j] = (pheromones[current_city][j] ** alpha) *
8                 (visibility[current_city][j] ** beta)
9     total = sum(probabilities)
10    if total == 0:
11        probabilities = [1 if not visited[j] else 0 for j in range(n)]
12        total = sum(probabilities)
13    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.

```
1 def calculate_route_length(route, distance_matrix):
2     length = 0
3     for i in range(len(route) - 1):
4         from_city = route[i]
5         to_city = route[i + 1]
6         distance = distance_matrix[from_city][to_city]
7         length += distance
8     return length
```

Листинг 3.6 — Алгоритм вычисления вычисления длины маршрута

В листинге 3.7 представлен алгоритм обновления феромонов.

```
1 def update_pheromones(distance_matrix, pheromones, routes, lengths, rho, Q,
2   elite_route=None):
3     n = len(pheromones)
4     for i in range(n):
5         for j in range(n):
6             pheromones[i][j] *= (1 - rho)
7
8     for route, length in zip(routes, lengths):
9         for i in range(len(route) - 1):
10             pheromones[route[i]][route[i + 1]] += Q / length
```



```

11     if elite_route:
12         elite_length = sum(distance_matrix[elite_route[i]][elite_route[i +
13             1]] for i in range(len(elite_route) - 1))
14         for i in range(len(elite_route) - 1):
15             pheromones[elite_route[i]][elite_route[i + 1]] += Q / elite_length
16
17     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 коммивояжёра, 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 учётом особенностей перехода между городами;
- 4) проведена параметризация муравьиного алгоритма по параметрам α , ρ , и числу итераций, результаты оформлены в таблицы и графики;

В результате проведённого исследования установлено, что метод полного перебора обеспечивает точное решение задачи коммивояжёра, 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.