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 Задача коммивояжера

Студент Равашдех Ф.Х.

Группа ИУ7-55Б

Преподаватели Строганов Ю.В.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы — сравнение методов решения задачи коммивояжера — полным перебором и на основе муравьиного алгоритма. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) реализовать алгоритмы указанных методов;
2) провести замеры процессорного времени выполнения реализаций алгоритмов от размерности матриц;
3) провести сравнительный анализ реализаций алгоритмов по полученным экспериментальным данным;
4) описать результаты в отчете.

1 Аналитическая часть

В этом разделе будет представлена информация о задаче коммивояжера, а также о способах её решения — методом полного перебора и методом на основе муравьиного алгоритма.

1.1 Задача коммивояжера

Задача о коммивояжере (англ. *Traveling Salesman Problem*), TSP — задача, в которой коммивояжер должен посетить N городов, побывав в каждом из них ровно по одному разу и завершив путешествие в том городе, с которого он начал. В какой последовательности ему нужно обходить города, чтобы общая длина его пути была наименьшей? [1]

1.2 Полный перебор

Можно решить задачу перебором всевозможных перестановок. Для этого нужно сгенерировать все $N!$ всевозможных перестановок вершин исходного графа, подсчитать для каждой перестановки длину маршрута и выбрать минимальный из них. Но тогда задача оказывается неосуществимой даже для достаточно небольших N [1].

1.3 Муравьиный алгоритм

В основе муравьиного алгоритма лежит идея моделирования поведения колонии муравьев. Каждый муравей определяет свой маршрут на основе оставленных другими муравьями феромонов, а также сам оставляет оставляет феромоны, чтобы последующие муравьи ориентировались по ним. В результате при прохождении каждым муравьем своего маршрута наибольшее число феромонов остается на самом оптимальном пути. Временная сложность алгоритма была оценена как $683 - (42,467N) + (1,0696N^2)$ [2]. Однако главный недостаток алгоритма заключается в том, что, по сравнению с алгоритмом полного перебора, он дает приближенное решение задачи, а не точное.

Вывод

В данном разделе рассмотрены методы решения задачи коммивояжера.

2 Конструкторская часть

2.1 Полный перебор

Схема алгоритма полным перебором решения задачи коммивояжера представлена на рисунке 2.1.

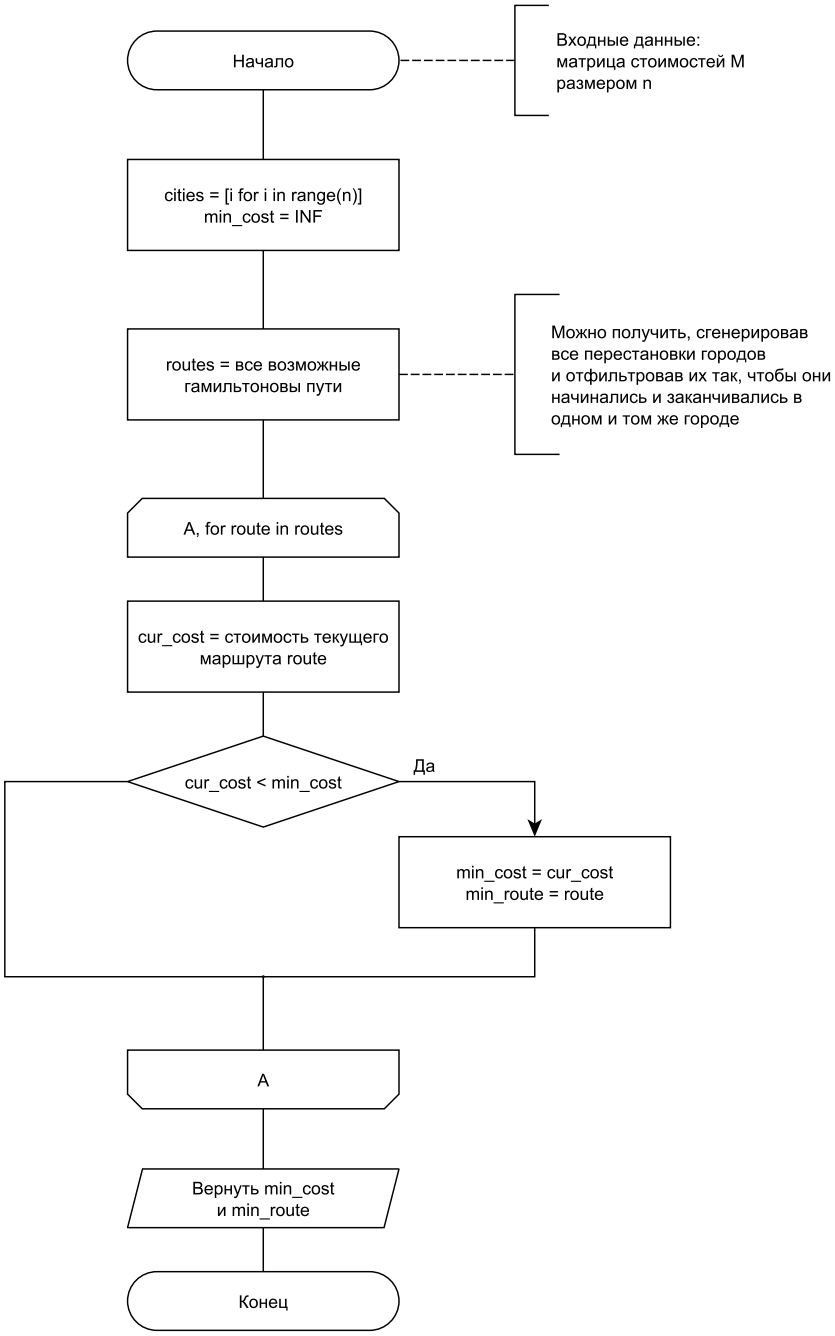

Рисунок 2.1 — Схема алгоритма полным перебором

2.2 Муравьиный алгоритм

Схема муравьиного алгоритма решения задачи коммивояжера представлена на рисунке 2.2. Схема алгоритма вычисления вероятности выбора следующей точки маршрута представлена на рисунке 2.3. Схема алгоритма обновления феромонов представлена на рисунке 2.4.

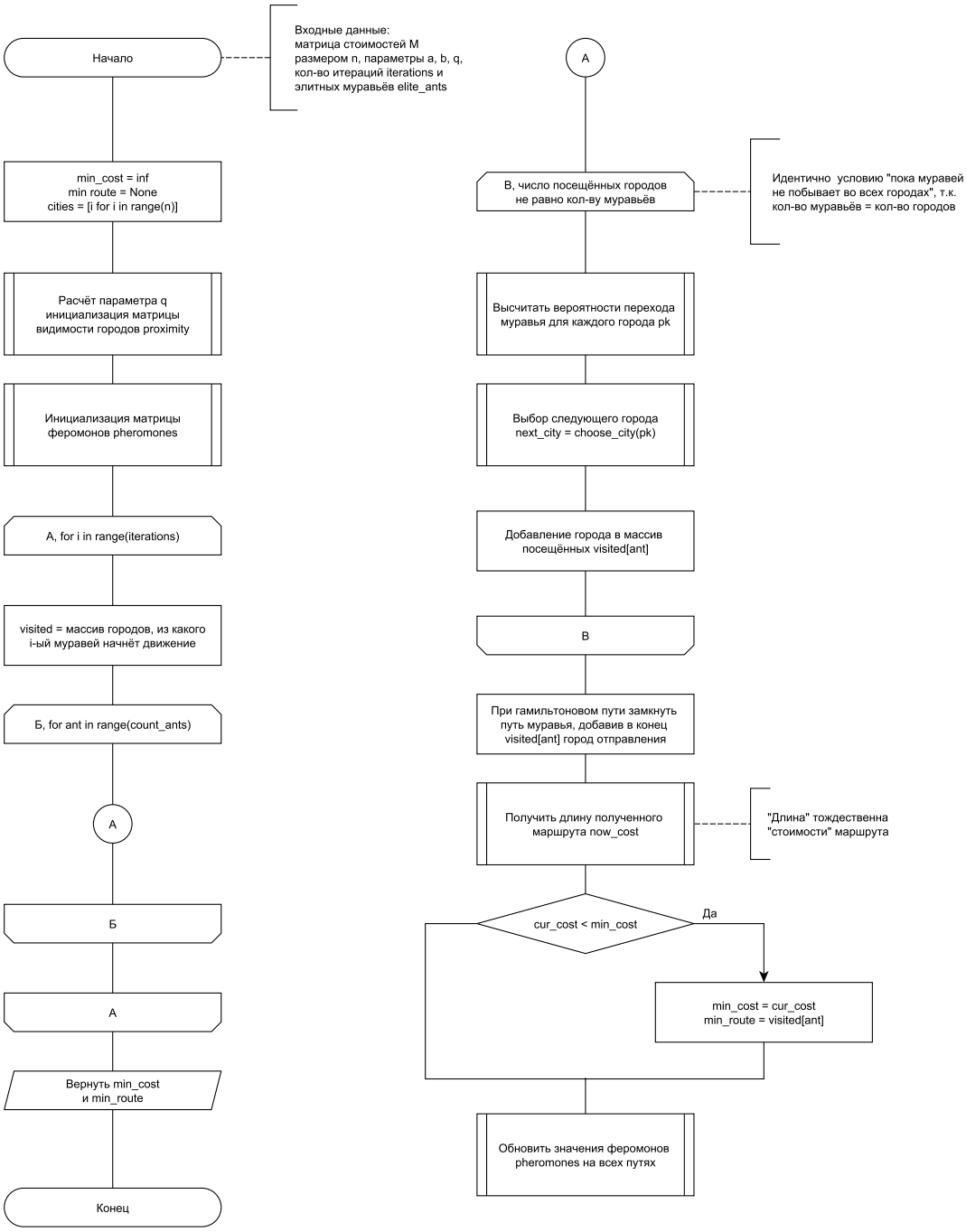

Рисунок 2.2 — Схема муравьиного алгоритма

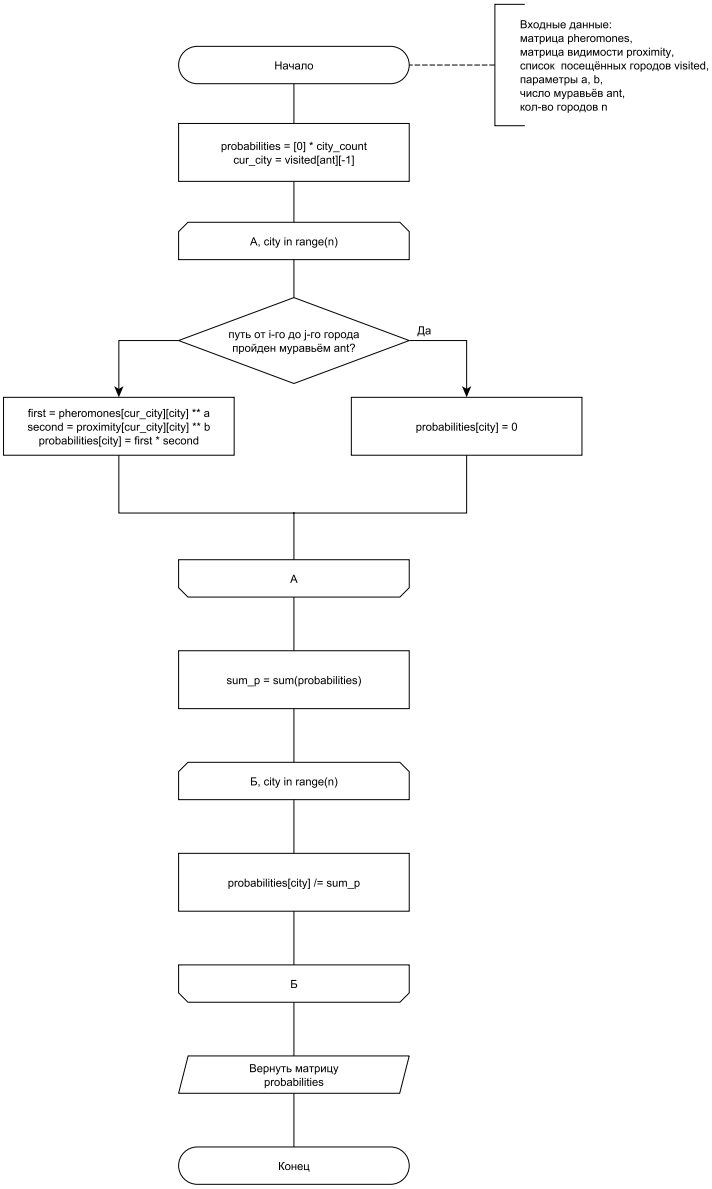

Рисунок 2.3 — Вычисление вероятности выбора следующей точки маршрута

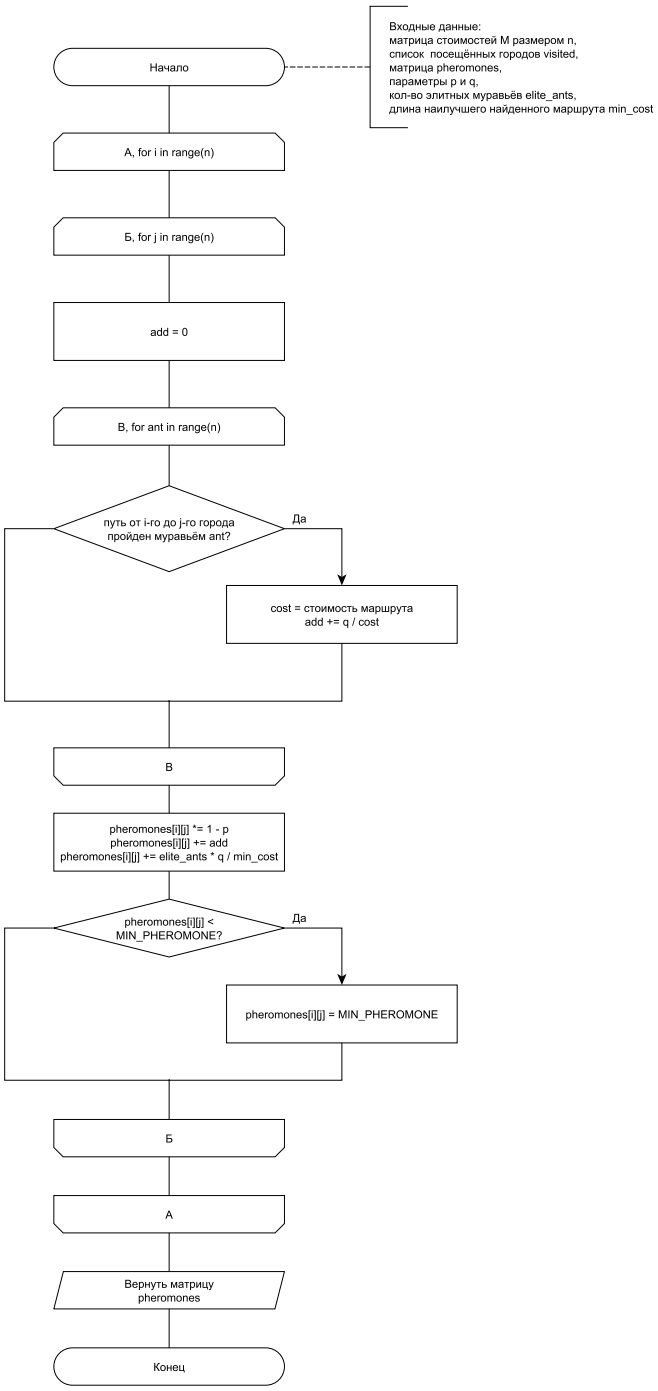

Рисунок 2.4 — Обновление феромонов

Вывод

В данной части работы были описаны алгоритм полного перебора и муравьиный алгоритм решения задачи коммивояжера.

3 Технологическая часть

3.1 Требования к программному обеспечению

Входные данные: матрица расстояний между городами и размер матрицы, для муравьиного алгоритма коэффициенты.

Выходные данные: минимальная стоимость маршрута и сам маршрут.

3.2 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке $Python$ [3], который имеет библиотеку для визуализации данных $Matplotlib$ [4].

3.3 Реализация алгоритмов

Реализация алгоритмов методом полного перебора и методом на основе муравьиного алгоритма представлена в листингах 3.1 и 3.2.

Листинг 3.1 — Функция решения задачи коммивояжера методом полного перебора

```
def bruteforce(src_matrix):
  routes = itertools.permutations(range(len(src_matrix)))
  min_cost = float('inf')
  min_route = None
  for route in routes:
    current_cost = sum(src_matrix[start_city][end_city] for start_city,
        end_city in zip(route, route[1:]))
    if current_cost < min_cost:
      min_cost = current_cost
      min_route = route
  return min_route, min_cost
```

Листинг 3.2 — Функции решения задачи коммивояжера методом на основе муравьиного алгоритма

```
def get_cost(matrix, route):
  cur_cost = 0
  for num in range(len(route) - 1):
    start_city = route[num]
    end_city = route[num + 1]
    cur_cost += matrix[start_city][end_city]
  return cur_cost

def get_all_routes(cities):
  routes = []
  for route in itertools.permutations(cities, len(cities)):
    route = list(route)
    route.append(route[0])
    routes.append(route)
  return routes

def get_q(matrix, size):
  q = 0
  count = 0
  for i in range(size):
    for j in range(i + 1, size):
      q += matrix[i][j] * 2
      count += 2
  return q / count

def get_proximity(matrix, size):
  proximity = [[0 for i in range(size)] for j in range(size)]
  for i in range(size):
    for j in range(i):
      proximity[i][j] = 1 / matrix[i][j]
      proximity[j][i] = proximity[i][j]
  return proximity

def get_visited_cities(cities, count_ants):
  visited = [[] for _ in range(count_ants)]
  for ant in range(count_ants):
    visited[ant].append(cities[ant])
  return visited
```

```
def get_probability(pheromones, proximity, visited, a, b, ant,
   city_count):
  probabilities = [0] * city_count
  now_city = visited[ant][-1]
  for city in range(city_count):
    if city not in visited[ant]:
      probabilities[city] = (pheromones[now_city][city] ** a) * (
         proximity[now_city][city] ** b)
    else:
      probabilities[city] = 0
  sum_p = sum(probabilities)
  for city in range(city_count):
    probabilities[city] /= sum_p
  return probabilities

def choose_city(probabilities):
  num = random()
  value = 0
  for i in range(len(probabilities)):
    value += probabilities[i]
    if value > num:
      return i

def check_route(i, j, route):
  for city_i, city in enumerate(route):
    if city == i and city_i + 1 < len(route) and route[city_i + 1] == j
       :
      return True
  return False

def update_pheromone(visited, matrix, pheromones, city_count, p, q,
   elite_ants_count, min_cost):
  for i in range(city_count):
    for j in range(city_count):
    add = 0
    for ant in range(city_count):
      if check_route(i, j, visited[ant]):
        cost = get_cost(matrix, visited[ant])
        add += q / cost
    pheromones[i][j] *= (1 - p)
    pheromones[i][j] += add + elite_ants_count * q / min_cost
```

```
    if pheromones[i][j] < MIN_PHEROMONE:
      pheromones[i][j] = MIN_PHEROMONE
  return pheromones

def ant_algorithm(matrix, size, a, b, p, iterations, elite_ants_count):
  cities = [i for i in range(size)]
  min_cost = inf
  min_route = None
  q = get_q(matrix, size)
  proximity = get_proximity(matrix, size)
  pheromones = [[START_PHEROMON for i in range(size)] for j in range(
     size)]
  count_ants = size
  for _ in range(iterations):
    visited = get_visited_cities(cities, count_ants)
    for ant in range(count_ants):
      while len(visited[ant]) != count_ants:
        pk = get_probability(pheromones, proximity, visited, a, b, ant,
            size)
        next_city = choose_city(pk)
        visited[ant].append(next_city)
      visited[ant].append(visited[ant][0])
      cur_cost = get_cost(matrix, visited[ant])
      if cur_cost < min_cost:
        min_cost = cur_cost
        min_route = visited[ant]
    pheromones = update_pheromone(visited, matrix, pheromones, size, p,
        q, elite_ants_count, min_cost)
  return min_route, min_cost
```

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, используемые средства реализации и приведена реализация алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики устройства, на котором выполнялись замеры:

1) операционная система — macOS Sonoma 14.1 (23B2073);
2) процессор — Apple M3;
3) оперативная память — 16 Гб.

4.2 Время выполнения реализаций алгоритмов

Замеры времени работы реализаций алгоритмов для каждого размера матриц проводились 100 раз, значение времени усреднялось. Данные замеров представлены в таблице 4.1.

Таблица 4.1 — Таблица замеров времени работы алгоритмов

Размер	Полным перебором	Муравьиный алгоритм
2	0.00011	0.00011
3	0.00027	0.00027
4	0.00104	0.00104
5	0.00581	0.00581
6	0.03724	0.03724
7	0.27912	0.27912
8	2.41039	2.41039
9	23.09638	23.09638
10	243.90210	243.90210

На рисунке 4.1 показаны графики зависимости времени работы реализаций алгоритмов решения задачи коммивояжера от размера матриц.

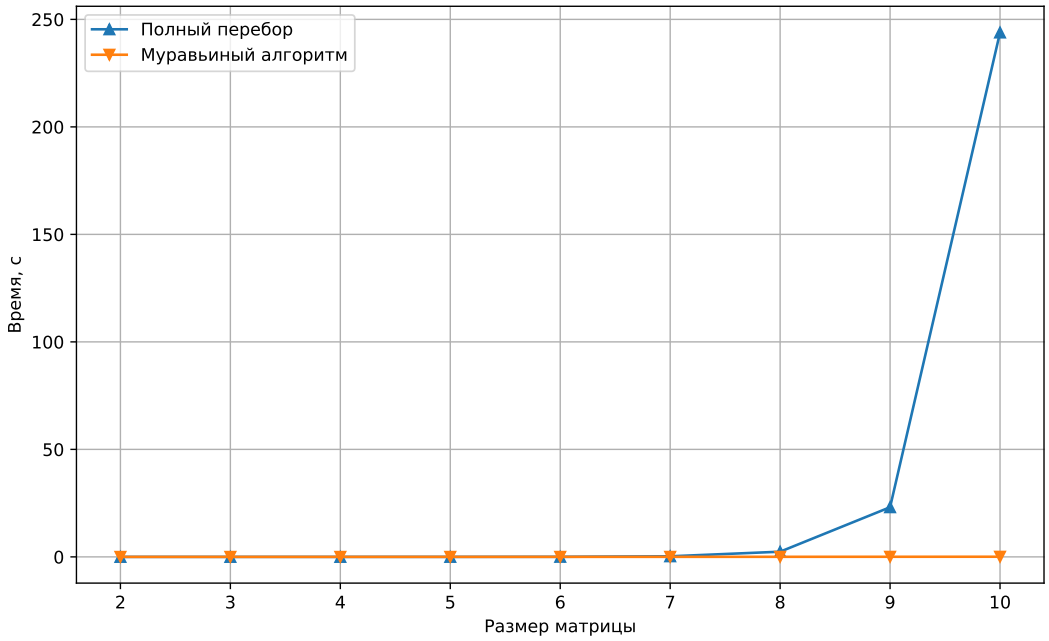

Рисунок 4.1 — Графики сравнения выполнения реализаций алгоритмов по времени

4.3 Класс данных

В качестве классов данных для параметризации используются три графа, построенных на городах Африки. В качестве стоимостей ребер использовались расстояния между городами по прямой.

4.4 Результаты параметризации

Таблица 4.2 — Результаты параметризации муравьиного алгоритма

Параметры			Граф 1			Граф 2			Граф 3		
α	ρ	Дни	min	max	avg	min	max	avg	min	max	avg
0.1	0.3	400	38	38	38	5	5	5	101	101	101
0.1	0.3	500	38	38	38	5	5	5	101	101	101
0.1	0.4	400	38	38	38	5	5	5	101	101	101
0.1	0.4	500	38	38	38	5	5	5	101	101	101
0.1	0.5	400	38	38	38	5	5	5	101	101	101
0.1	0.6	100	38	38	38	5	5	5	101	101	101
0.1	0.7	500	38	38	38	5	5	5	101	101	101

0.1	0.8	300	38	38	38	5	5	5	101	101	101
0.2	0.3	400	38	38	38	5	5	5	101	101	101
0.2	0.3	500	38	38	38	5	5	5	101	101	101

4.5 Вывод

В результате исследования было получено, что на графе с количеством вершин менее 8 муравьиный алгоритм и алгоритм полного перебора решают задачу за примерно одинаковое время, а при количестве вершин большем или равном 8 алгоритм полного перебора начинает проигрывать в скорости муравьиному алгоритму.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: сравнение алгоритмов решения задачи коммивояжера методом полного перебора и методом на основе муравьиного алгоритма было проведено.

В ходе выполнения лабораторной работы были решены все задачи:

1) реализованы алгоритмы указанных методов;

2) провести замеры процессорного времени выполнения реализаций алгоритмов от размерности матриц;

3) проведен сравнительный анализ реализаций алгоритмов по полученным экспериментальным данным;

4) описаны результаты в отчете.

Муравьиный алгоритм справляется с решением задачи коммивояжера быстрее алгоритма полного перебора, но не всегда дает наилучшее решение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задача о коммивояжере [Электронный ресурс]. Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Задача_коммивояжера,_ДП_по_подмножествам (дата обращения 22.01.25)
2. Штовба С. Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях, 2003
3. Python3 documentation [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (дата обращения 13.10.24)
4. Библиотека Matplotlib [Электронный ресурс]. Режим доступа: https://matplotlib.org/ (дата обращения 13.10.24)