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 Муравьиный алгоритм

Студент Якуба Д. В.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

Цель лабораторной работы

Реализация муравьиного алгоритма и приобретение навыков параметризации методов на примере реализованного алгоритма, примененного к задаче коммивояжера.

Задачи лабораторной работы

1) изучить алгоритм полного перебора для решения задачи коммивояжера;
2) реализовать алгоритм полного перебора для решения задачи коммивояжера;
3) изучить муравьиный алгоритм для решения задачи коммивояжера;
4) реализовать муравьиный алгоритм для решения задачи коммивояжера;
5) провести параметризацию муравьиного алгоритма на двух классах данных;
6) провести сравнительный анализ скорости работы реализованных алгоритмов;
7) подготовить отчёт по проведенной работе.

1 Аналитическая часть

В данном разделе описаны задача коммивояжёра, идея муравьиного алгоритма и алгоритма полного перебора для решения этой задачи.

1.1 Задача коммивояжера

Коммивояжёр (фр. commis voyageur) — бродячий торговец. Задача коммивояжёра — важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок [1]. В описываемой задаче рассматривается несколько городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса.

1.2 Алгоритм полного перебора для решения задачи коммивояжера

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них.

Такой подход гарантирует точное решение задачи, однако, так как задача относится к числу трансвычислительных [2], то уже при небольшом числе городов решение за приемлемое время невозможно.

1.3 Муравьиный алгоритм для решения задачи коммивояжера

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев [3]. Колония представляет собой систему с очень простыми правилами автономного поведения особей.

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает, во время прохождения, каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Самоорганизация колонии является результатом взаимодействия следующих компонентов:

- случайность — муравьи имеют случайную природу движения;
- многократность — колония допускает число муравьев, достигающее от нескольких десятков до миллионов особей;
- положительная обратная связь — во время движения муравей откладывает феромон, позволяющий другим особям определить для себя оптимальный маршрут;
- отрицательная обратная связь — по истечении определенного времени феромон испаряется;
- целевая функция.

Пусть муравей обладает следующими характеристиками:

- зрение — определяет длину ребра;
- обоняние — чувствует феромон;
- память — запоминает маршрут, который прошел.

Введем целевую функцию $\eta_{ij} = 1/D_{ij}$, где D_{ij} — расстояние из текущего пункта i до заданного пункта j.

Посчитаем вероятности перехода в заданную точку по формуле (1.1):

$$P_{kij} = \begin{cases} \frac{t_{ij}^a \eta_{ij}^b}{\sum_{q=1}^{m} t_{iq}^a \eta_{iq}^b}, \text{вершина не была посещена ранее муравьем k,} \\ 0, \text{иначе} \end{cases} \tag{1.1}$$

где a, b – настраиваемые параметры, t - концентрация феромона, причем $a + b = const$, а при $a = 0$ алгоритм вырождается в жадный [**?**].

Когда все муравьи завершили движение происходит обновление феромона по формуле (1.2):

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

где

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

L_k — длина пути k-ого муравья, Q — настраивает концентрацию нанесения/испарения феромона, N — количество муравьев.

Вывод

Были рассмотрены задача коммивояжера, муравьиный алгоритм и алгоритм полного перебора для решения поставленной задачи.

В данной работе стоит задача реализации двух рассмотренных алгоритмов.

2 | Конструкторская часть

В данном разделе представлены схемы муравьиного алгоритма и алгоритма полного перебора для решения задачи коммивояжера.

2.1 Схема алгоритма полного перебора

Схема алгоритма полного перебора для решения задачи коммивояжера предоставлена на рисунке 2.1. Схема алгоритма нахождения всех перестановок в графе, использующаяся в алгоритме полного перебора, предоставлена на рисунке 2.2.

2.2 Схема муравьиного алгоритма

На рисунке 2.3 предоставлена схема реализации муравьиного алгоритма для решения задачи коммивояжера.

Вывод

Были представлены схемы алгоритма Брезенхема, а также реализации конвейерной обработки данных для данного алгоритма.

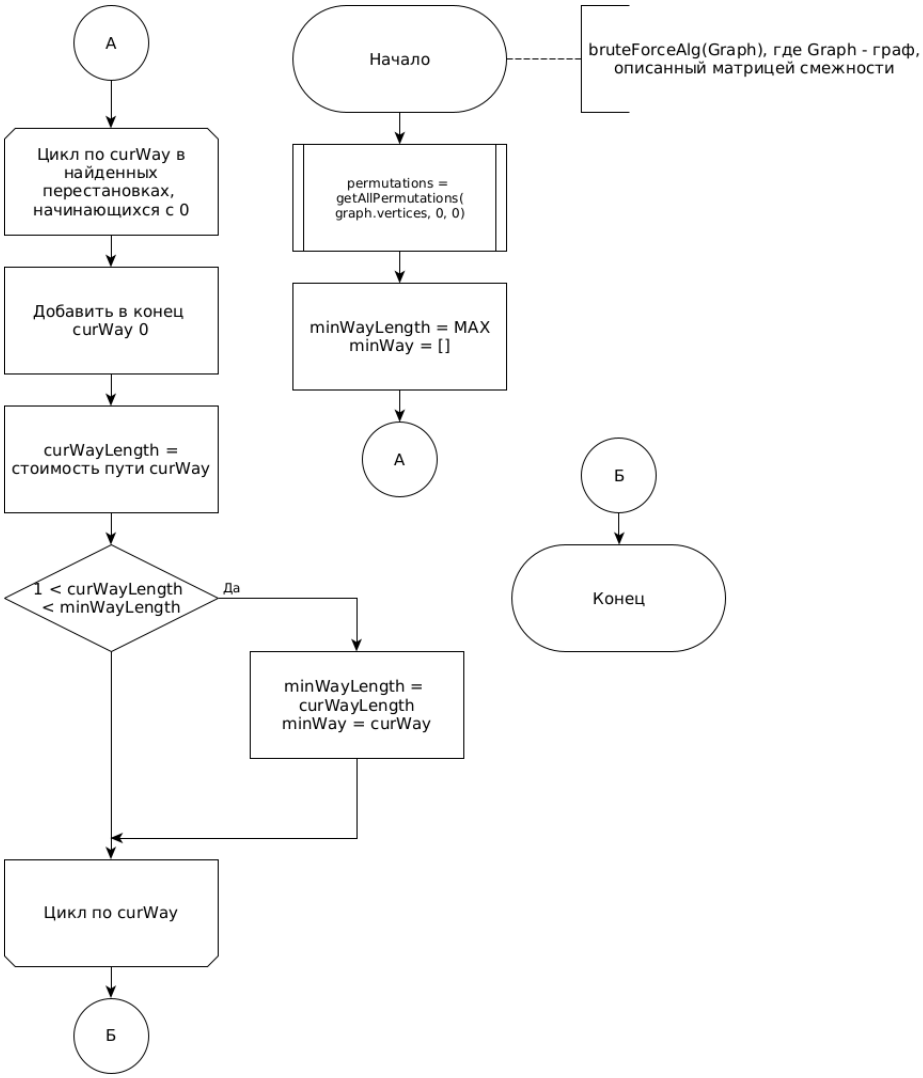

Рис. 2.1: Схема алгоритма полного перебора.

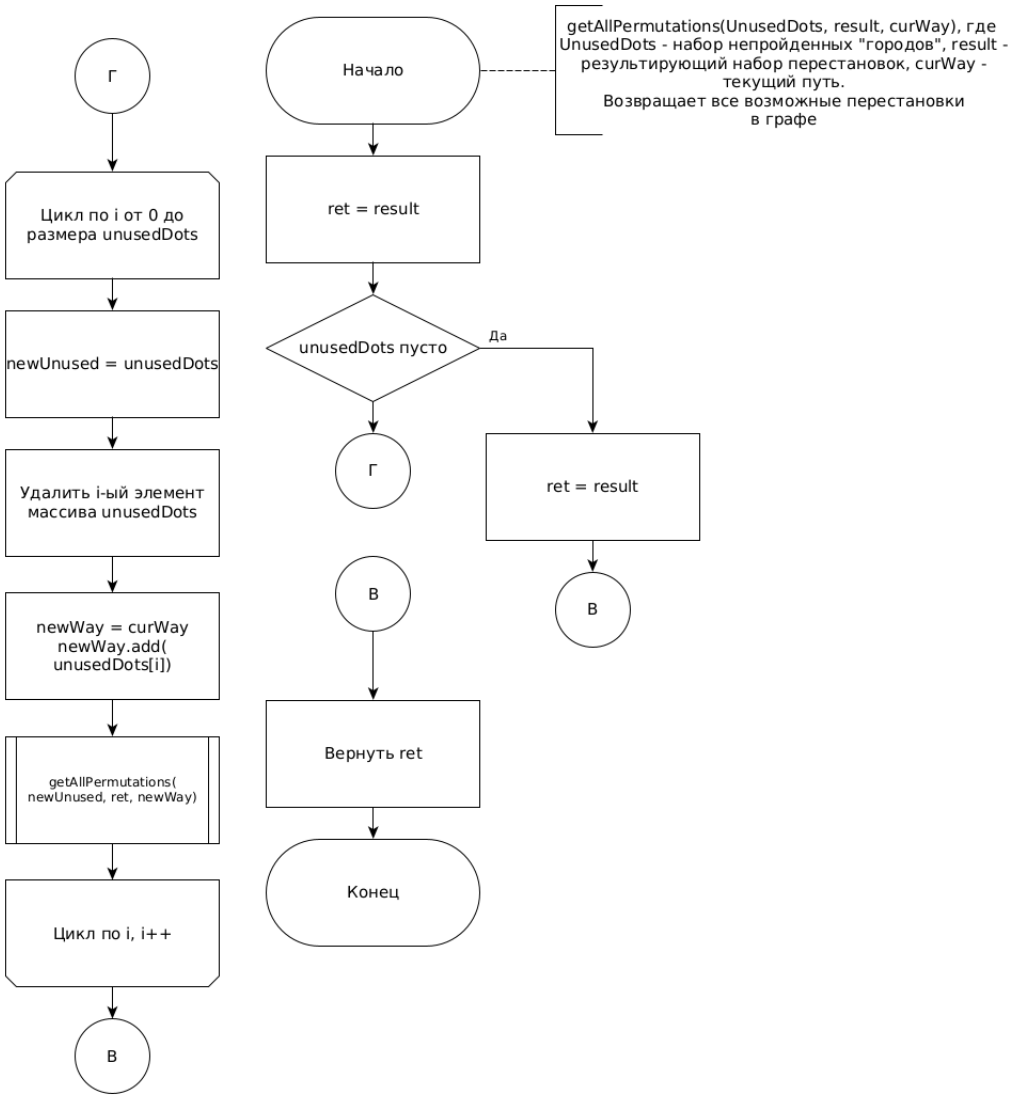

Рис. 2.2: Схема алгоритма нахождения всех перестановок в графе.

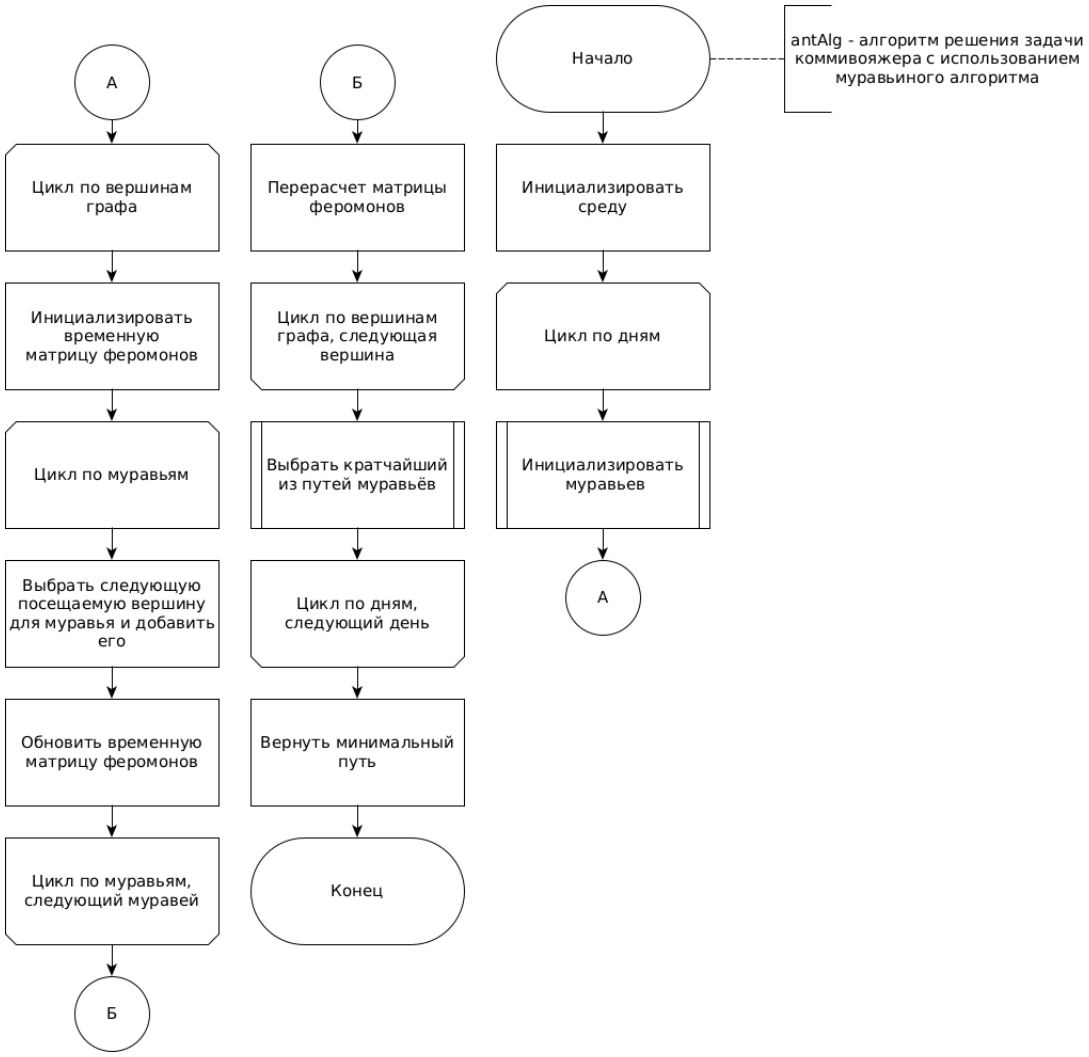

Рис. 2.3: Схема реализации муравьиного алгоритма.

3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации программного обеспечения, а также листинг кода.

3.1 Требования к программному обеспечению

- входные данные - матрицы смежности для сильно-связного графа;
- выходные данные - оптимальные пути, найденные с помощью алгоритма полного перебора и муравьиного алгоритма.

3.2 Средства реализации программного обеспечения

При написании программного продукта был использован язык программирования Kotlin [4].

Данный выбор обусловлен следующими факторами:

- Возможность портирования алгоритмов для работы с Android;
- Большое количество справочной литературы, связанной с ЯП Java.

Для тестирования производительности реализаций алгоритмов использовалась утилита measureTimedValue.

При написании программного продукта использовалась среда разработки IntelliJ IDEA [5].

Данный выбор обусловлен тем, что язык программирования Kotlin - это разработка компании JetBrains, поставляющей данную среду разработки.

3.3 Листинг кода

В листингах 3.1 и 3.3 предоставлены классы реализации рассматриваемых алгоритмов.

Листинг 3.1: Реализация класса графа и класса графа феромонов

```
const val randomStart = 1
const val randomEnd = 10

class Graph(size_ : Int)
{
    private var size = size_
    private var adjacencyMatrix = Array(size) { IntArray(size) }

    fun setWay(from : Int, to : Int, length : Int)
    {
        adjacencyMatrix[from][to] = length
        adjacencyMatrix[to][from] = length
    }

    fun getWay(from : Int, to : Int) : Int
    {
        return adjacencyMatrix[from][to]
    }

    fun getWayLength(way : MutableList<Int>) : Int
    {
        var length = 0
        for (i in 0 until way.size - 1)
        {
            val curLength = getWay(way[i], way[i + 1])
            length += curLength
        }
        return length
    }

    fun getSize() : Int
    {
        return size
    }

    fun generate()
    {
        for (i in 0 until size)
```

```
39              for (j in 0 until i)
40                  setWay(i, j, Random.nextInt(randomStart,
                        randomEnd))
41      }
42
43      fun getVertecies() : MutableList<Int>
44      {
45          val ret : MutableList<Int> = mutableListOf()
46          for (i in 0 until size)
47              ret.add(i)
48          return ret
49      }
50
51      fun print()
52      {
53          for (i in 0 until size)
54          {
55              for (j in 0 until size)
56                  print("%3d ".format(getWay(i, j)))
57              print('\n')
58          }
59      }
60  }
61
62  class PheromoneGraph(size_ : Int)
63  {
64      private var size = size_
65      private var adjacencyMatrix = Array(size) { DoubleArray(size)
            }
66
67      fun set(i : Int, j : Int, value : Double)
68      {
69          adjacencyMatrix[i][j] = value
70      }
71
72      fun get(i : Int, j : Int) : Double
73      {
74          return adjacencyMatrix[i][j]
75      }
76
77      fun getSize() : Int
78      {
79          return size
80      }
81
```

```
82      fun print()
83      {
84          for (i in 0 until size)
85          {
86              for (j in 0 until size)
87                  print("%5f ".format(get(i, j)))
88              print('\n')
89          }
90      }
91  }
```

Листинг 3.2: Класс реализации алгоритма полного перебора для решения задачи коммивояжера

```
1   class BruteForce
2   {
3       fun bruteForceAlg(graph: Graph)
4       {
5           val permutations = getAllPermutations(graph.getVertecies
                ())
6
7           var minWayLength = MAX_VALUE
8           var minWay: MutableList<Int> = mutableListOf()
9           for (curWay in permutations.filter { it[0] == 0 })
10          {
11              curWay.add(0)
12              val curWayLength = graph.getWayLength(curWay)
13              if (curWayLength in 1 until minWayLength)
14              {
15                  minWayLength = curWayLength
16                  minWay = curWay
17              }
18          }
19
20          println("Min way length is: $minWayLength")
21          println("Min way is: $minWay")
22      }
23
24      private fun getAllPermutations(unusedDots: MutableList<Int>,
25                                     result : MutableList<MutableList<Int>>? =
                                           null,
26                                     curWay : MutableList<Int>? = null) :
                                           MutableList<MutableList<Int>>
27      {
28          var ret = result
```

```
29          if (ret == null)
30              ret = mutableListOf()
31
32          if (unusedDots.size == 0)
33          {
34              ret.add(curWay!!)
35              return ret
36          }
37
38          for (i in 0 until unusedDots.size)
39          {
40              val newUnusedDots = unusedDots.toMutableList()
41              newUnusedDots.removeAt(i)
42              var newWay : MutableList<Int> = mutableListOf()
43              if (curWay != null)
44                  newWay = curWay.toMutableList()
45              newWay.add(unusedDots[i])
46
47              getAllPermutations(newUnusedDots, ret, newWay)
48          }
49          return ret
50      }
51  }
```

Листинг 3.3: Класс реализации муравьиного алгоритма для решения задачи коммивояжера

```
1   class Colony(graph_: Graph)
2   {
3       var graph = graph_
4
5       class Ant
6       {
7           var way : MutableList<Int> = mutableListOf()
8           var startVertex : Int = 0
9           var visitedVerticies = BooleanArray(0)
10
11          var graph = Graph(0)
12
13          constructor(graph_: Graph, startVertex_ : Int)
14          {
15              way = mutableListOf(startVertex_)
16              visitedVerticies = BooleanArray(graph_.getSize())
17              startVertex = startVertex_
18              visitedVerticies[startVertex_] = true
```

```
19
20              graph = graph_
21          }
22
23          fun visitVertex(vertex : Int)
24          {
25              visitedVertices[vertex] = true
26              way.add(vertex)
27          }
28
29          fun getWayLength(): Int
30          {
31              return graph.getWayLength(way)
32          }
33      }
34
35      val initValue = 0.1
36
37      fun antsInitialization() : MutableList<Ant>
38      {
39          val ants = mutableListOf<Ant>()
40          for (i in 0 until graph.getSize())
41              ants.add(Ant(graph, Random.nextInt(0, graph.getSize()
                    )))
42
43          return ants
44      }
45
46      fun antAlg(days : Int, alpha : Double, beta : Double, rho :
            Double, q : Double) : Pair<MutableList<Int>, Int>
47      {
48          var minWay = Int.MAX_VALUE
49          var min = mutableListOf<Int>()
50          var pheromoneGraph = PheromoneGraph(graph.getSize())
51
52          for (i in 0 until pheromoneGraph.getSize())
53          {
54              for (j in 0 until pheromoneGraph.getSize())
55                  pheromoneGraph.set(i, j, initValue)
56          }
57
58          for (day in 0 until days)
59          {
60              val ants = antsInitialization()
61
```

```
62              for (i in 0 until graph.getSize() - 1)
63              {
64                  val tempPheromoneGraph = PheromoneGraph(
                        pheromoneGraph.getSize())
65
66                  for (curAntNum in 0 until ants.size)
67                  {
68                      var curAnt = ants[curAntNum]
69                      var sumChance = 0.0
70                      var curVertex = curAnt.way.last()
71
72                      for (vertId in 0 until graph.getSize())
73                      {
74                          if (!curAnt.visitedVertices[vertId])
75                              sumChance += pheromoneGraph.get(
                                    curVertex, vertId).pow(alpha) *
76                                      (1.0 / graph.getWay(curVertex
                                          , vertId).toDouble()).pow
                                          (beta)
77                      }
78
79                      var coin = Random.nextDouble()
80                      var curChoice = 0
81                      while (coin > 0)
82                      {
83                          if (!curAnt.visitedVertices[curChoice])
84                          {
85                              var chance = pheromoneGraph.get(
                                    curVertex, curChoice).pow(alpha)
                                    *
86                                      (1.0 / graph.getWay(curVertex
                                          , curChoice).toDouble()).
                                          pow(beta) / sumChance
87                              coin -= chance
88                          }
89                          curChoice++
90                      }
91                      curChoice--
92
93                      ants[curAntNum].visitVertex(curChoice)
94                      tempPheromoneGraph.set(curVertex, curChoice,
95                              tempPheromoneGraph.get(curVertex,
                                    curChoice) +
96                                      q / graph.getWay(curVertex,
                                          curChoice).toDouble())
```

```
97                  }
98
99                  for (k in 0 until graph.getSize())
100                     for (j in 0 until graph.getSize())
101                         pheromoneGraph.set(k, j, (1 - rho) *
                                pheromoneGraph.get(k, j) +
                                tempPheromoneGraph.get(k, j))
102             }
103
104             for (ant in ants)
105             {
106                 ant.way.add(ant.way[0])
107                 val cur = ant.graph.getWayLength(ant.way)
108                 if (cur < minWay)
109                 {
110                     minWay = ant.getWayLength()
111                     min = ant.way
112                 }
113             }
114         }
115
116         return Pair(min, minWay)
117     }
118 }
```

3.4 Тестирование программного продукта

В таблице 3.1 приведены тесты для функции, реализующей алгоритм для решения задачи коммивояжера. Тесты пройдены успешно.

Вывод

Спроектированные алгоритмы были реализованы и протестированы.

Таблица 3.1: Тестирование функций

Матрица смежности	Ожидаемый наименьший путь
$\begin{pmatrix} 0 & 3 & 4 & 7 \\ 3 & 0 & 3 & 7 \\ 4 & 3 & 0 & 7 \\ 7 & 7 & 7 & 0 \end{pmatrix}$	20, [0, 1, 2, 3, 0]
$\begin{pmatrix} 0 & 7 & 4 & 3 \\ 7 & 0 & 4 & 5 \\ 4 & 4 & 0 & 5 \\ 3 & 5 & 5 & 0 \end{pmatrix}$	16, [0, 2, 1, 3, 0]
$\begin{pmatrix} 0 & 8 & 9 & 7 \\ 8 & 0 & 4 & 4 \\ 9 & 4 & 0 & 2 \\ 7 & 4 & 2 & 0 \end{pmatrix}$	21, [0, 1, 2, 3, 0]

4 | Исследовательская часть

4.1 Технические характеристики

Технические характеристики ЭВМ, на котором выполнялись исследования:

- ОС: Manjaro Linux 20.1.1 Mikah
- Оперативная память: 16 Гб
- Процессор: Intel Core i7-10510U

При проведении замеров времени ноутбук был подключен к сети электропитания.

4.2 Пример работы программного обеспечения

На рисунке 4.1 приведен пример работы программы.

```
  0   9   9   9   8   5   4   4   5   9
  9   0   1   4   9   1   7   2   2   8
  9   1   0   1   1   8   6   5   1   8
  9   4   1   0   1   9   5   8   1   2
  8   9   1   1   0   8   8   7   3   9
  5   1   8   9   8   0   1   9   9   9
  4   7   6   5   8   1   0   5   7   7
  4   2   5   8   7   9   5   0   6   1
  5   2   1   1   3   9   7   6   0   5
  9   8   8   2   9   9   7   1   5   0
By the Brute Force Algorithm:
Min way length is: 18
Min way is: [0, 6, 5, 1, 8, 2, 4, 3, 9, 7, 0]
By the Ant Algorithm:
Min way is: ([1, 5, 6, 0, 7, 9, 3, 4, 2, 8, 1], 18)

Process finished with exit code 0
```

Рис. 4.1: Пример работы ПО.

4.3 Исследование скорости работы алгоритмов

Алгоритмы тестировались на данных, сгенерированных случайным образом один раз.

Результаты замеров времени приведены в таблице 4.1. На рисунке 4.2 приведен графики зависимостей времени работы алгоритмов от размерности матрицы смежности.

Таблица 4.1: Замеры времени

Размерность матрицы	Полный перебор	Муравьиный алгоритм
3	7768810	6373786
4	7224319	6397801
5	7990770	7014032
6	12360077	8505435
7	18466313	12679791
8	54640264	13791822
9	186935706	16966893
10	1332144414	17774922

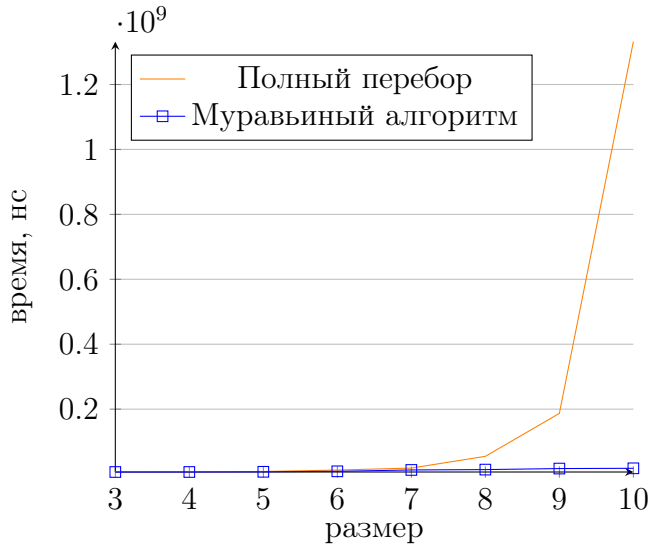

Рис. 4.2: Зависимость времени работы от размера матрицы смежности

4.4 Постановка эксперимента

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Рассмотрим два класса данных и подберем к ним параметры, при которых метод даст точный результат при минимальном количестве итераций.

Будем рассматривать матрицы размерности 10×10, так как иначе получение точного результата алгоритмом полного перебора слишком велико.

В качестве первого класса данных выделим матрицу смежности, в которой все значения незначительно отличаются друг от друга, например, в диапазоне $[1, 25]$. Вторым классом будут матрицы, где значения могут значительно отличаться, например $[1, 15000]$.

Будем запускать муравьиный алгоритм для всех значений $\alpha, P \in [0, 1]$, с шагом $= 0.1$, пока не будет найдено точное значение для каждого набора.

В результате тестирования будет выведена таблица со значениями α, β, ρ, $days$, $dist$, где $days$ — количество дней, данных для решения задачи, а α, β, p — настоечные параметры.

4.4.1 Класс данных 1

$$M = \begin{pmatrix} 0 & 3 & 14 & 2 & 22 & 20 & 2 & 14 & 18 & 23 \\ 3 & 0 & 12 & 12 & 8 & 24 & 19 & 4 & 20 & 12 \\ 14 & 12 & 0 & 23 & 8 & 3 & 21 & 16 & 6 & 5 \\ 2 & 12 & 23 & 0 & 22 & 19 & 1 & 13 & 15 & 8 \\ 22 & 8 & 8 & 22 & 0 & 8 & 2 & 8 & 11 & 7 \\ 20 & 24 & 3 & 19 & 8 & 0 & 15 & 12 & 18 & 9 \\ 2 & 19 & 21 & 1 & 2 & 15 & 0 & 13 & 18 & 21 \\ 14 & 4 & 16 & 13 & 8 & 12 & 13 & 0 & 7 & 23 \\ 18 & 20 & 6 & 15 & 11 & 18 & 18 & 7 & 0 & 19 \\ 23 & 12 & 5 & 8 & 7 & 9 & 21 & 23 & 19 & 0 \end{pmatrix} \tag{4.1}$$

В таблице 4.2 приведены результаты параметризации метода решения задачи коммивояжера на основании муравьиного алгоритма. Полный перебор определил оптимальную длину пути 44.

Таблица 4.2: Таблица коэффициентов для класса данных №1

α	β	ρ	days	длина пути
0	1	0	50	47
0	1	0.1	50	44
0	1	0.2	50	49
0	1	0.3	50	44
0	1	0.4	50	52
0	1	0.5	50	51
0	1	0.6	50	44
0	1	0.7	50	44
0	1	0.8	50	44
0	1	0.9	50	51
0	1	1	50	44
0.1	0.9	0	50	44
0.1	0.9	0.1	50	47
0.1	0.9	0.2	50	44
0.1	0.9	0.3	50	44
0.1	0.9	0.4	50	44
0.1	0.9	0.5	50	47
0.1	0.9	0.6	50	44
0.1	0.9	0.7	50	44
0.1	0.9	0.8	50	44
0.1	0.9	0.9	50	44
0.1	0.9	1	50	67
0.2	0.8	0	50	44
0.2	0.8	0.1	50	44
0.2	0.8	0.2	50	44
0.2	0.8	0.3	50	44
0.2	0.8	0.4	50	44
0.2	0.8	0.5	50	44
0.2	0.8	0.6	50	44
0.2	0.8	0.7	50	44
0.2	0.8	0.8	50	44
0.2	0.8	0.9	50	44
0.2	0.8	1	50	93
0.3	0.7	0	50	44
0.3	0.7	0.1	50	44
0.3	0.7	0.2	50	44
0.3	0.7	0.3	50	44
0.3	0.7	0.4	50	44
0.3	0.7	0.5	50	44
0.3	0.7	0.6	50	49
0.3	0.7	0.7	50	44
0.3	0.7	0.8	50	44
0.3	0.7	0.9	50	52
0.3	0.7	1	50	88
0.4	0.6	0	50	44
0.4	0.6	0.1	50	44
0.4	0.6	0.2	50	44
0.4	0.6	0.3	50	44
0.4	0.6	0.4	50	44
0.4	0.6	0.5	50	44
0.4	0.6	0.6	50	44
0.4	0.6	0.7	50	44
0.4	0.6	0.8	50	44
0.4	0.6	0.9	50	53
0.4	0.6	1	50	93
0.5	0.5	0	50	44
0.5	0.5	0.1	50	44
0.5	0.5	0.2	50	44
0.5	0.5	0.3	50	44
0.5	0.5	0.4	50	44
0.5	0.5	0.5	50	47
0.5	0.5	0.6	50	53
0.5	0.5	0.7	50	59
0.5	0.5	0.8	50	44
0.5	0.5	0.9	50	62
0.5	0.5	1	50	77
0.6	0.4	0	50	44
0.6	0.4	0.1	50	44
0.6	0.4	0.2	50	44
0.6	0.4	0.3	50	44
0.6	0.4	0.4	50	44
0.6	0.4	0.5	50	49
0.6	0.4	0.6	50	44
0.6	0.4	0.7	50	52
0.6	0.4	0.8	50	58
0.6	0.4	0.9	50	59
0.6	0.4	1	50	93
0.7	0.3	0	50	44
0.7	0.3	0.1	50	44
0.7	0.3	0.2	50	49
0.7	0.3	0.3	50	49
0.7	0.3	0.4	50	44
0.7	0.3	0.5	50	47
0.7	0.3	0.6	50	52
0.7	0.3	0.7	50	57
0.7	0.3	0.8	50	51
0.7	0.3	0.9	50	54
0.7	0.3	1	50	90

α	β	ρ	days	длина пути
0.8	0.2	0	50	44
0.8	0.2	0.1	50	44
0.8	0.2	0.2	50	44
0.8	0.2	0.3	50	44
0.8	0.2	0.4	50	44
0.8	0.2	0.5	50	52
0.8	0.2	0.6	50	60
0.8	0.2	0.7	50	60
0.8	0.2	0.8	50	72
0.8	0.2	0.9	50	70
0.8	0.2	1	50	95
0.9	0.1	0	50	44
0.9	0.1	0.1	50	44
0.9	0.1	0.2	50	44
0.9	0.1	0.3	50	52
0.9	0.1	0.4	50	80
0.9	0.1	0.5	50	44
0.9	0.1	0.6	50	55
0.9	0.1	0.7	50	44
0.9	0.1	0.8	50	64
0.9	0.1	0.9	50	62
0.9	0.1	1	50	93
1.0	0.0	0	50	44
1.0	0.0	0.1	50	49
1.0	0.0	0.2	50	44
1.0	0.0	0.3	50	44
1.0	0.0	0.4	50	64
1.0	0.0	0.5	50	64
1.0	0.0	0.6	50	60
1.0	0.0	0.7	50	44
1.0	0.0	0.8	50	53
1.0	0.0	0.9	50	60
1.0	0.0	1	50	93

4.4.2 Класс данных 2

$$M = \begin{pmatrix} 0 & 242 & 306 & 590 & 364 & 379 & 1202 & 344 & 505 & 560 \\ 242 & 0 & 465 & 1385 & 853 & 853 & 878 & 324 & 173 & 1006 \\ 306 & 465 & 0 & 1418 & 636 & 1036 & 553 & 1463 & 926 & 551 \\ 590 & 1385 & 1418 & 0 & 1198 & 828 & 11 & 1214 & 5 & 552 \\ 364 & 853 & 636 & 1198 & 0 & 1152 & 1282 & 807 & 1494 & 821 \\ 379 & 853 & 1036 & 828 & 1152 & 0 & 753 & 185 & 1440 & 1200 \\ 1202 & 878 & 553 & 11 & 1282 & 753 & 0 & 1228 & 967 & 308 \\ 344 & 324 & 1463 & 1214 & 807 & 185 & 1228 & 0 & 1140 & 1450 \\ 505 & 173 & 926 & 5 & 1494 & 1440 & 967 & 1140 & 0 & 533 \\ 560 & 1006 & 551 & 552 & 821 & 1200 & 308 & 1450 & 533 & 0 \end{pmatrix} \tag{4.2}$$

В таблице 4.3 приведены результаты параметризации метода решения задачи коммивояжера на основании муравьиного алгоритма. Полный перебор определил оптимальную длину пути 2936.

Таблица 4.3: Таблица коэффициентов для класса данных №2

α	β	ρ	days	длина пути
0	1	0	50	2936
0	1	0.1	50	2936
0	1	0.2	50	3148
0	1	0.3	50	2936
0	1	0.4	50	2936
0	1	0.5	50	2936
0	1	0.6	50	2936
0	1	0.7	50	2936
0	1	0.8	50	2936
0	1	0.9	50	2936
0	1	1	50	3148
0.1	0.9	0	50	2936
0.1	0.9	0.1	50	3148
0.1	0.9	0.2	50	3148
0.1	0.9	0.3	50	2936
0.1	0.9	0.4	50	2936
0.1	0.9	0.5	50	2936
0.1	0.9	0.6	50	3148
0.1	0.9	0.7	50	3148
0.1	0.9	0.8	50	2936
0.1	0.9	0.9	50	2936
0.1	0.9	1	50	5262
0.2	0.8	0	50	2936
0.2	0.8	0.1	50	2936
0.2	0.8	0.2	50	2936
0.2	0.8	0.3	50	2936
0.2	0.8	0.4	50	2936
0.2	0.8	0.5	50	2936
0.2	0.8	0.6	50	2936
0.2	0.8	0.7	50	2936
0.2	0.8	0.8	50	2936
0.2	0.8	0.9	50	3297
0.2	0.8	1	50	5262
0.3	0.7	0	50	2936
0.3	0.7	0.1	50	2936
0.3	0.7	0.2	50	2936
0.3	0.7	0.3	50	2936
0.3	0.7	0.4	50	2936
0.3	0.7	0.5	50	2936
0.3	0.7	0.6	50	2936
0.3	0.7	0.7	50	3148
0.3	0.7	0.8	50	3148
0.3	0.7	0.9	50	2936
0.3	0.7	1	50	5262
0.4	0.6	0	50	3148
0.4	0.6	0.1	50	2936
0.4	0.6	0.2	50	2936
0.4	0.6	0.3	50	2936
0.4	0.6	0.4	50	2936
0.4	0.6	0.5	50	2936
0.4	0.6	0.6	50	2936
0.4	0.6	0.7	50	2936
0.4	0.6	0.8	50	2936
0.4	0.6	0.9	50	3529
0.4	0.6	1	50	5262
0.5	0.5	0	50	2936
0.5	0.5	0.1	50	2936
0.5	0.5	0.2	50	2936
0.5	0.5	0.3	50	2936
0.5	0.5	0.4	50	2936
0.5	0.5	0.5	50	3602
0.5	0.5	0.6	50	2936
0.5	0.5	0.7	50	2936
0.5	0.5	0.8	50	2936
0.5	0.5	0.9	50	2936
0.5	0.5	1	50	5179
0.6	0.4	0	50	2936
0.6	0.4	0.1	50	2936
0.6	0.4	0.2	50	2936
0.6	0.4	0.3	50	2936
0.6	0.4	0.4	50	2936
0.6	0.4	0.5	50	2936
0.6	0.4	0.6	50	3148
0.6	0.4	0.7	50	3563
0.6	0.4	0.8	50	3642
0.6	0.4	0.9	50	3379
0.6	0.4	1	50	5262
0.7	0.3	0	50	3148
0.7	0.3	0.1	50	2936
0.7	0.3	0.2	50	2936
0.7	0.3	0.3	50	2936
0.7	0.3	0.4	50	3297
0.7	0.3	0.5	50	3379
0.7	0.3	0.6	50	2936
0.7	0.3	0.7	50	3529
0.7	0.3	0.8	50	3640
0.7	0.3	0.9	50	3639
0.7	0.3	1	50	5179

α	β	ρ	days	длина пути
0.8	0.2	0	50	3297
0.8	0.2	0.1	50	2936
0.8	0.2	0.2	50	3379
0.8	0.2	0.3	50	2936
0.8	0.2	0.4	50	3379
0.8	0.2	0.5	50	3148
0.8	0.2	0.6	50	4121
0.8	0.2	0.7	50	3727
0.8	0.2	0.8	50	3460
0.8	0.2	0.9	50	3604
0.8	0.2	1	50	5262
0.9	0.1	0	50	2936
0.9	0.1	0.1	50	2936
0.9	0.1	0.2	50	3297
0.9	0.1	0.3	50	2936
0.9	0.1	0.4	50	2936
0.9	0.1	0.5	50	3430
0.9	0.1	0.6	50	3604
0.9	0.1	0.7	50	3604
0.9	0.1	0.8	50	3297
0.9	0.1	0.9	50	3148
0.9	0.1	1	50	5262
1.0	0.0	0	50	2936
1.0	0.0	0.1	50	2936
1.0	0.0	0.2	50	3148
1.0	0.0	0.3	50	2936
1.0	0.0	0.4	50	3878
1.0	0.0	0.5	50	4073
1.0	0.0	0.6	50	3379
1.0	0.0	0.7	50	4581
1.0	0.0	0.8	50	5166
1.0	0.0	0.9	50	3710
1.0	0.0	1	50	5262

Вывод

Сравнительный анализ скорости работы двух реализованных алгоритмов указал на факт того, что на всех рассмотренных значениях размер-

ности матрицы смежности, муравьиный алгоритм преобладает по скорости над алгоритмом полного перебора. Из таблицы 4.1 можно увидеть, что на минимальной рассмотренной размерности матрицы муравьиный алгоритм работает быстрее алгоритма полного перебора на ≈ 22

Из графика, предоставленного на рисунке 4.2, видно, что муравьиный муравьиный алгоритм до размерности матрицы смежности, равной 6, сравним по скорости работы со вторым реализованным алгоритмом. Как таковое преобладание первого алгоритма над вторым начинается с матриц смежности, размерности которых превышают 6 элементов.

На основе проведенной параметризации для двух классов данных можно сделать следующие выводы:

- Для класса данных, предоставленного в уравнении 4.1 в качестве матрицы, содержащей приблизительно равные значения, наилучшими наборами стали ($\alpha = 0.2, \beta = 0.8, \rho = x$), где $\rho \neq 1$, так как они показали наиболее стабильные результаты, равные эталонному значению оптимального пути (44).

- Для класса данных, предоставленного в уравнении 4.2 в качестве матрицы, содержащей различные значения, наилучшими наборами стали ($\alpha = 0, 0.2, 0.5, \beta = 1, 0.8, 0.5, \rho = x$). При этих параметрах, количество найденных эталонных оптимальных путей составило 9 единиц.

Заключение

В ходе выполнения лабораторной работы была выполнена цель и следующие задачи:

1) был изучен алгоритм полного перебора для решения задачи коммивояжера;
2) был реализован алгоритм полного перебора для решения задачи коммивояжера;
3) был изучен муравьиный алгоритм для решения задачи коммивояжера;
4) был реализован муравьиный алгоритм для решения задачи коммивояжер;
5) была проведена параметризация муравьиного алгоритма на двух классах данных;
6) был проведен сравнительный анализ скорости работы реализованных алгоритмов;
7) был подготовлен отчёт по проведенной работе.

Исследования показали, что муравьиный алгоритм решения задачи коммивояжера в среднем преобладает по скорости на ≈ 192253274 наносекунд над алгоритмом полного перебора. На конечных значениях размерности матрицы смежности разрыв в скорости работы составляет $\approx 650\%$. Таким образом, был сделан вывод, что муравьиный алгоритм имеет преимущество над методом полного перебора за счет того, что способен работать с данными большего объема за меньшее время. Однако, в отличии от алгоритма полного перебора, муравьиный алгоритм не гарантирует, что найденный путь будет оптимальным.

Были подобраны параметры для оптимальной работы реализованного эвристического алгоритма для оптимальной работы метода на двух классах данных: с приблизительно равными значениями, с различными значениями.

Литература

[1] Perl. Примеры программ [Электронный ресурс]. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/ (дата обращения 09.12.2020).

[2] Решение задачи коммивояжера для поиска оптимального плана перевозок предприятия (на примере ООО «Фабрика еды») [Электронный ресурс]. Режим доступа: https://perm.hse.ru/data/2015/02/16/1092093493/%D0%A0%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8%20%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80%D0%B0.pdf (дата обращения 09.12.2020).

[3] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. ФИЗМАТЛИТ, 2008. с. 304.

[4] Kotlin language specification [Электронный ресурс]. Режим доступа: https://kotlinlang.org/spec/introduction.html (дата обращения 09.10.2020).

[5] IntelliJ IDEA documentation [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/ru-ru/idea/ (дата обращения 09.12.2020).