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 Криков А.В.

Группа ИУ7-53Б

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

Москва — 2021 г.

Содержание

Введение

Одна из самых известных и важных задач транспортной логистики (и комбинаторной оптимизации) – задача коммивояжёра или "задача о странствующем торговце". Суть задачи сводится к поиску оптимального (кратчайшего, быстрейшего или самого дешевого) пути, проходящего через промежуточный пункты по одному разу и возвращающегося в исходную точку. К примеру, нахождение наиболее выгодного маршрута, позволяющего коммивояжёру посетить со своим товаром определенные города по одному разу и вернуться обратно. Мерой выгодности маршрута может быть минимальное время поездки, минимальные расходы на дорогу или минимальная длина пути. В наше время, когда стоимость доставки часто бывает сопоставима со стоимостью самого товара, а скорость доставки - один из главных приоритетов, задача нахождения оптимального маршрута приобретает огромное значение.

Муравьиный алгоритм - один из эффективных полиномиальных алгоритмов для нахождения приближенных решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

Целью данной работы является изучение следующих алгоритмов решения задачи коммивояжёра:

- муравьиный алгоритм;
- наивный алгоритм;

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить подходы к решению задачи коммивояжёра;
- привести схемы реализации алгоритмов решения задачи коммивояжёра;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;

- протестировать разработанное ПО;
- привести сведения о модулях программы;
- определить требования к ПО;
- провести экспериментальные замеры временных характеристик реализованных алгоритмов;
- на основании проделанной работы сделать выводы и подготовить отчет.

1 Аналитическая часть

В данном разделе будут описаны задача коммивояжёра, а также алгоритм полного перебора и муравьиный алгоритм для решения данной задачи.

1.1 Постановка задачи

Задача коммивояжёра – важная задача транспортной логистики [1], отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута.

1.2 Описание алгоритмов

1.2.1 Алгоритм полного перебора

Метод полного перебора, по-другому именуемый методом грубой силы, является простым, логичным и широко используемым математическим методом. Он применим во многих, если не во всех, областях математики: задача коммивояжера также не является исключением. Идея brute force предельно проста: перебираются всевозможные решения и из них выбирается решение (или множество решений) отвечающее условию задачи.

В задаче коммивояжера, соответственно, требуется из всевозможных вариантов объезда пунктов выбрать маршрут, занимающий кратчайшее время (или минимальный по стоимости маршрут).

Огромным преимуществом метода полного перебора перед другими методами решения задачи коммивояжера является гарантированность нахож-

дения наилучшего маршрута. Другие методы советуют лишь «хороший» маршрут, который совсем не обязательно является лучшим. Кроме того, к достоинствам метода относится простота его программной реализации.

Однако, в связи с наличием огромного недостатка, метод полного перебора крайне редко используется на практике. Этим недостатком является временная сложность алгоритма. Асимметричная задача коммивояжера с n посещаемых пунктов требует при полном переборе рассмотрения (n-1)! туров, а факториал растет невероятно быстро. Поэтому метод полного перебора может применяться только для задач малой размерности (при рассмотрении до двух десятков посещаемых пунктов).

1.2.2 Муравьиный алгоритм

В то время как простой метод перебора всех вариантов чрезвычайно неэффективный при большом количестве городов, эффективными признаются решения, гарантирующие получение ответа за время, ограниченное полиномом от размерности задачи.

В основе алгоритма лежит поведение муравьиной колонии [2] – маркировка более удачных путей большим количеством феромона. Рассмотрим биологическую модель поведения такой колонии.

В реальном мире муравьи (первоначально) ходят в случайном порядке и по нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым, и, как следствие, плотность феромонов остаётся высокой. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие

муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути. Этапы работы муравьиной колонии представлены на рис. 1.1.

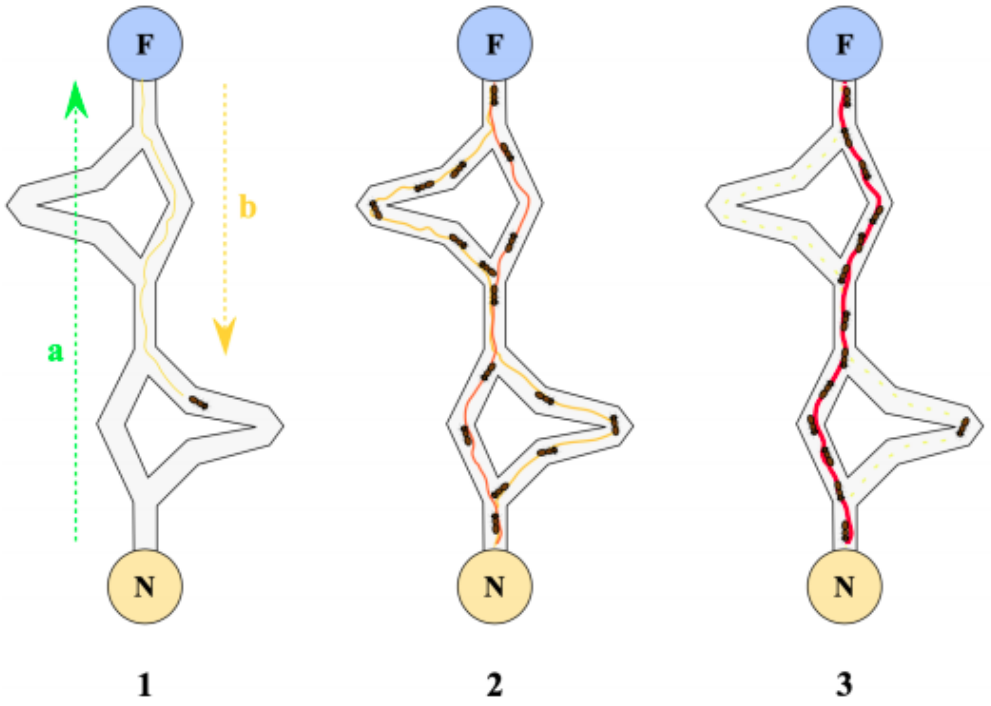

Рис. 1.1: Работа муравьиной колонии

Процесс поиска кратчайшего пути от колонии до источника питания (рис. 1.1):

1. первый муравей находит источник пищи (F) любым способом (a), а затем возвращается к гнезду (N), оставив за собой тропу из феромонов (b);

2. затем муравьи выбирают один из четырёх возможных путей, затем укрепляют его и делают привлекательным;

3. муравьи выбирают кратчайший маршрут, так как феромоны с более длинных путей быстрее испаряются.

Вероятность перехода из вершины i в вершину j определяется по формуле 1.1.

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где $\tau_{i,j}$— расстояние от города i до j;
$\eta_{i,j}$—количество феромонов на ребре ij;
α— параметр влияния длины пути;
β— параметр влияния феромона.
Уровень феромона обновляется в соответствии с формулой 1.2

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{1.2}$$

где ρ - доля феромона, которая испарится;
$\tau_{i,j}$ - количество феромона на дуге ij;
$\Delta\tau_{i,j}$ - количество отложенного феромона, вычисляется по формуле 1.3.

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \tag{1.3}$$

где k - количество муравьев в вершине графа с индексами i и j.
Описание поведения муравьев при выборе пути.

- Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через J_{ik} список городов, которые необходимо посетить муравью k, находящемуся в городе i.

- Муравьи обладают «зрением» - видимость есть эвристическое желание посетить город j , если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами.

- Муравьи обладают «обонянием» - они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i,j) в момент времени t обозначим через $\tau_{i,j}(t)$.

- Пройдя ребро (i,j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда

откладываемое количество феромона может быть задано формулой 1.4.

$$\Delta\tau_{i,j}^{k} = \begin{cases} Q/L_k, & \text{если k-ый мурваей прошел по ребру ij;} \\ 0, & \text{иначе} \end{cases} \tag{1.4}$$

где Q - количество феромона, переносимого муравьем.

Вывод

В данной работе стоит задача реализации алгоритмов для решения задачи коммивояжёра. Входными данными будет являтся матрица M расстояний между городами размера $n * n$, где n - количество строк и столбцов, а $M_{i,j}$ - расстояние из города с индексом i в город с индексом j. Выходными данными будет являтся целое число - кратчайший маршрут, а также массив содержащий номера городов кратчайшего маршрута. В связи с ограничениями накладываемыми на ПО, входное число n должно быть корректным, а также все расстояния должны быть целочисленными. Необходимо дать теоретическую оценку алгоритма полного перебора и муравьиного алгоритма.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы вышеизложенных алгоритмов, описаны способы тестирования и определены структуры данных.

2.1 Разработка алгоритмов

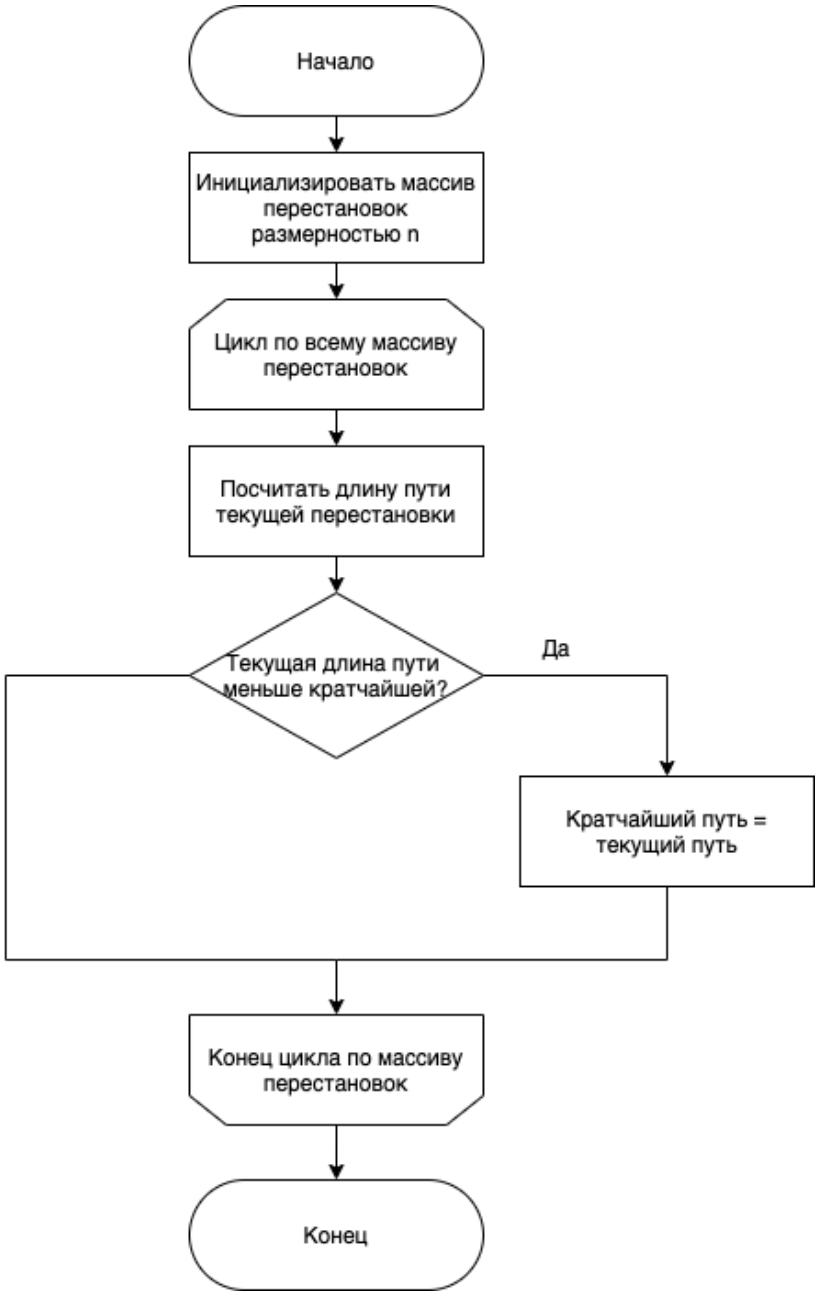

Рис. 2.1: Схема наивного алгоритма

Рис. 2.2: Схема муравьиного алгоритма

2.2 Описание структур данных

Исходя из условия задачи приходим к выводу о том, что для хранения информации о расстояниях между городами наиболее удобно использовать структуру данных - двумерный массив. Для хранения минимального пути

необходимо использовать целочисленный одномерный массив.

2.3 Структура ПО

ПО будет состоять из следующих модулей:

- Основной модуль;
- Модуль включающий в себя реализацию наивного алгоритма решения задачи коммивояжёра;
- Модуль включающий в себя реализацию муравьиного алгоритма;
- Модуль для работы с входными и выходными данными;
- Модуль вспомогательных функций;

2.4 Описание способов тестирования

В рамках данной лабораторной работы можно выделить следующие классы эквивалентности:

- входными данными является ациклический ориентированный взвешенный граф;
- входными данными является циклический ориентированный взвешенный граф;

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы двух алгоритмов решения задачи коммивояжёра. Приведена структура ПО, описаны структуры данных.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг алгоритмов.

3.1 Средства реализации

Для разработки ПО был выбран язык Java, поскольку он предоставляет разработчику широкий спектр возможностей и позволяет разрабатывать кроссплатформенные приложени, а также предоставляет большой набор инструментов для работы с многопоточностью. В качестве среды разработки использовалась Visual Studio Code. [3]

3.2 Листинг кода

Листинг 3.1: Реализация алгоритма полного перебора

```
1   private static ArrayList<ArrayList<Integer>> dists;
2
3   public static void swap(ArrayList<Integer> list, int i, int j) {
4       int t = list.get(i);
5       list.set(i, list.get(j));
6       list.set(j, t);
7   }
8
9   public static void generatePermutations(ArrayList<Integer> arrayList, int leftIndex)
        {
10      if (leftIndex >= arrayList.size()) {
11          dists.add((ArrayList<Integer>) arrayList.clone());
12          return;
13      }
14
15      generatePermutations(arrayList, leftIndex + 1);
16      for (int i = leftIndex + 1; i < arrayList.size(); i++) {
17          swap(arrayList, leftIndex, i);
18          generatePermutations(arrayList, leftIndex + 1);
19          swap(arrayList, leftIndex, i);
20      }
21  }
22
```

```
23      public static void bruteForce(ArrayList<ArrayList<Integer>> matrix) {
24          dists = new ArrayList<>();
25          ArrayList<Integer> temp = new ArrayList<>();
26          for (int i = 0; i < matrix.size(); i++) {
27              temp.add(i);
28          }
29          generatePermutations(temp, 0);
30
31          int minCost = Integer.MAX_VALUE;
32          int d;
33          for (int i = 0; i < dists.size(); i++) {
34              d = matrix.get(dists.get(i).get(0)).get(dists.get(i).get(dists.get(0).size()
                    - 1));
35              for (int j = 1; j < dists.get(0).size(); j++) {
36                  d += matrix.get(dists.get(i).get(j - 1)).get(dists.get(i).get(j));
37              }
38
39              if (d < minCost) {
40                  minCost = d;
41              }
42          }
43      }
```

Листинг 3.2: Реализация муравьиного алгоритма

```
1       public static int l_min = Integer.MAX_VALUE;
2       public static int LMIN = Integer.MAX_VALUE;
3       public static ArrayList<Integer> routeMin = new ArrayList<>();
4
5       public static ArrayList<Integer> newArrayWithoutLast(ArrayList<Integer> to, int
            last) {
6           ArrayList<Integer> cur = new ArrayList<>();
7           for (int i = 0; i < to.size(); i++) {
8               if (to.get(i) != last) {
9                   cur.add(to.get(i));
10              }
11          }
12
13          return cur;
14      }
15
16
17      public static int getRoute(ArrayList<Integer> all, int start, ArrayList<Integer>
            route, Integer len, ArrayList<ArrayList<Integer>> matrix,
            ArrayList<ArrayList<Double>> tao, ArrayList<ArrayList<Double>> attraction, int
            alpha, int beta) {
18          route.clear();
19          route.add(start);
20
```

```
21          ArrayList<Integer> to = newArrayWithoutLast(all, start);
22          int n_1 = tao.size() - 2;
23          int from;
24          double coin, sum;
25          boolean flag;
26
27          for (int i = 0; i < n_1; i++) {
28              sum = 0;
29              flag = true;
30              from = route.get(i);
31              ArrayList<Double> p = getProbability(from, to, tao, attraction, alpha, beta);
32              coin = Double.valueOf((new Random().nextInt() % 10000)) / 10000;
33              for (int j = 0; j < p.size() && flag; j++) {
34                  sum += p.get(j);
35                  if (coin < sum) {
36                      route.add(to.get(j));
37                      len += matrix.get(from).get(to.get(j));
38                      to = newArrayWithoutLast(to, to.get(j));
39                      flag = false;
40                  }
41              }
42          }
43
44          len += matrix.get(route.get(route.size() - 1)).get(to.get(0));
45          route.add(to.get(0));
46          len += matrix.get(route.get(route.size() - 1)).get(route.get(0));
47          route.add(route.get(0));
48
49          return len;
50      }
51
52
53      public static ArrayList<Double> getProbability(int from, ArrayList<Integer> to,
            ArrayList<ArrayList<Double>> tao, ArrayList<ArrayList<Double>> attraction, int
            alpha, int beta) {
54          double znam = 0.0, chisl = 0.0;
55          int n = to.size();
56          ArrayList<Double> result = new ArrayList<>(n);
57          for (int i =0; i < n; i++) {
58              result.add(0.0);
59          }
60
61          for (int i = 0; i < n; i++) {
62              znam += Math.pow(tao.get(from).get(to.get(i)), alpha) *
                    Math.pow(attraction.get(from).get(to.get(i)), beta);
63          }
64          for (int j = 0; j < n; j++) {
65              chisl = Math.pow(tao.get(from).get(to.get(j)), alpha) *
```

```
                Math.pow(attraction.get(from).get(to.get(j)), beta);
            result.set(j, chisl / znam);
        }

        return result;
    }

    public static boolean inRoute(int a, int b, ArrayList<Integer> route) {
        boolean res = false;
        int m = route.size() - 1;
        for (int i = 0; i < m; i++) {
            if (a == route.get(i) && b == route.get(i + 1)) {
                res = true;
            }
        }

        return res;
    }


    public static void ant(int size, ArrayList<ArrayList<Integer>> matrix, int alpha,
        int beta, double q, int timeMax, FileWriter fileWriter) throws IOException {
        l_min = Integer.MAX_VALUE;
        routeMin.clear();

        double taoMin, taoStart, Q;
        ArrayList<Integer> all = new ArrayList<>(size);
        Q = 350.0;
        taoMin = 0.001;
        taoStart = 0.5;

        ArrayList<ArrayList<Integer>> routes = new ArrayList<>(size);
        ArrayList<Integer> lens = new ArrayList<>(size);

        ArrayList<ArrayList<Double>> attraction = new ArrayList<>(size);
        ArrayList<ArrayList<Double>> tao = new ArrayList<>(size);

        for (int i = 0; i < size; i++) {
            attraction.add(new ArrayList<>(size));
            tao.add(new ArrayList<>(size));
            routes.add(new ArrayList<>(size));
            lens.add(0);
            all.add(0);

            for (int j = 0; j < size;j ++) {
                attraction.get(i).add(0.0);
                tao.get(i).add(0.0);
                routes.get(i).add(0);
```

```
112             }
113         }
114
115         for (int i = 0; i < size; i++) {
116             lens.set(i, 0);
117             all.set(i, i);
118
119             for (int j = 0; j < size; j++) {
120                 if (i != j) {
121                     attraction.get(i).set(j, 1.0 / matrix.get(i).get(j));
122                     tao.get(i).set(j, taoStart);
123                 }
124             }
125         }
126
127
128         for (int time = 0; time < timeMax; time++) {
129             for (int k = 0; k < size; k++) {
130                 int len = getRoute(all, k, routes.get(k), lens.get(k), matrix, tao,
                        attraction, alpha, beta);
131                 lens.set(k, len);
132                 if (lens.get(k) < l_min) {
133                     l_min = lens.get(k);
134                     routeMin = routes.get(k);
135                 }
136             }
137             for (int i = 0; i < size; i++) {
138                 for (int j = 0; j < size; j++) {
139                     double sum = 0.0;
140                     for (int m = 0; m < size; m++) {
141                         if (inRoute(i, j, routes.get(m))) {
142                             sum += Q / lens.get(m);
143                         }
144                     }
145
146                     tao.get(i).set(j, tao.get(i).get(j) * (1 - q) + sum);
147                     if (tao.get(i).get(j) < taoMin) {
148                         tao.get(i).set(j, taoMin);
149                     }
150                 }
151             }
152         }
153
154
155         if (l_min < LMIN) {
156             LMIN = l_min;
157         }
158
```

```
159         fileWriter.write(timeMax + "␣" + alpha + "␣" + beta + "␣" + q + "␣" + l_min + "␣
                ");
160
161         for (int i = 0; i < routeMin.size(); i++) {
162             fileWriter.write(routeMin.get(i) + "␣");
163         }
164         fileWriter.write("\n");
165     }
```

3.3 Тестирование функций

В таблице 3.1 приведены тесты для функций, реализующих алгоритм кодирования строки. Тесты пройдены успешно.

Количество городов	Матрица расстояний	Ожидаемый результат
4	$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{pmatrix}$	80
4	$\begin{pmatrix} 0 & 8 & 9 & 7 \\ 8 & 0 & 4 & 4 \\ 9 & 4 & 0 & 2 \\ 7 & 4 & 2 & 0 \end{pmatrix}$	21

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

В данном разделе будет произведено сравнение алгоритма полного перебора для решения задачи коммивояжёра и муравьиного алгоритма, а также будет приведена демонстрация работы программы.

4.1 Технические характеристики

- Операционная система: Windows 10. [4]
- Память: 16 GiB.
- Процессор: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz. [5]

4.2 Временные характеристики

Для сравнения возьмем 8 матриц расстояний размерностью: [3, 4, 5, ... 10]. Возспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма N раз (N >= 10), после чего поделим на N. Тем самым получим достаточно точные характеристики времени. Результат сравнения алгоритма полного перебора и муравьиного алгоритма представлен на рис 4.1.

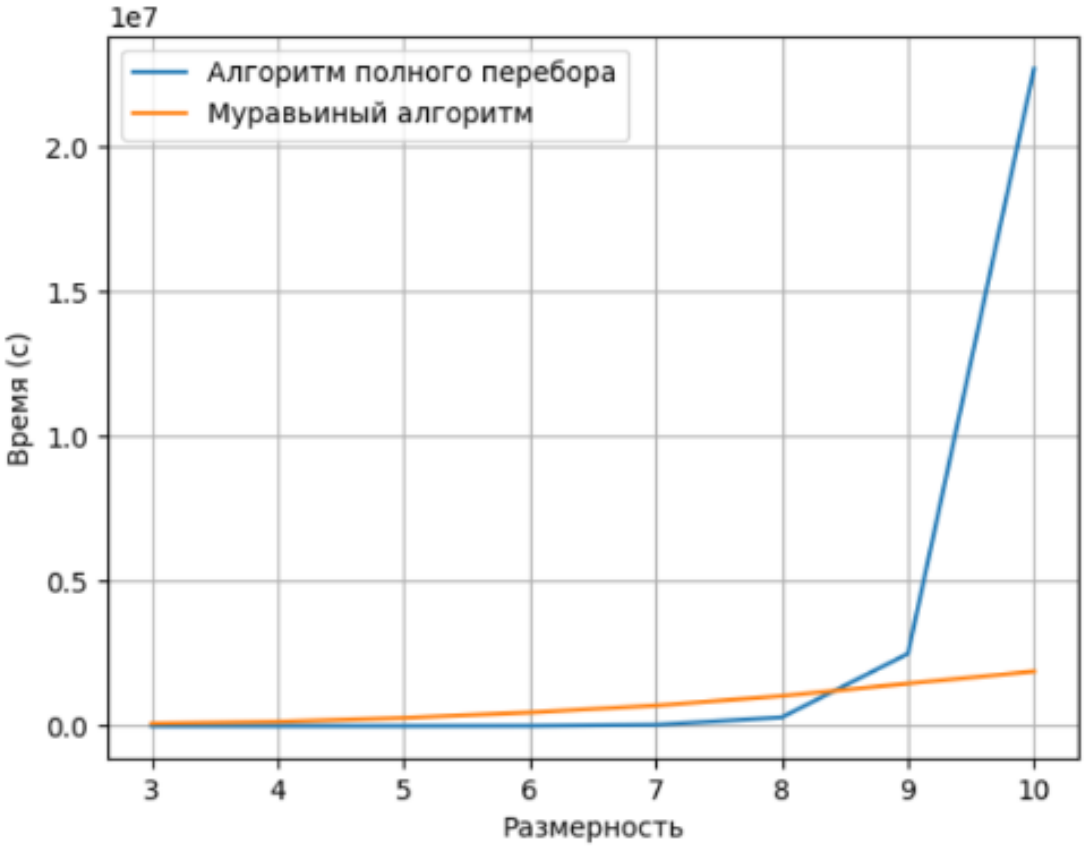

Рис. 4.1: Результат сравнения

4.3 Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Рассмотрим матрицу смежностей размерностью 10×10.

Таблица 4.1: Матрица смежностей

0	0	1	2	3	4	5	6	7	8	9
0	0	1790	200	1900	63	1659	1820	1395	2382	649
1	1790	0	1573	2435	1515	714	892	2193	1590	1003
2	200	1573	0	833	392	2404	962	902	141	1123
3	1900	2435	833	0	2283	1652	2362	2262	1512	2166
4	63	1515	392	2283	0	1322	290	1305	2100	969
5	1659	714	2404	1652	1322	0	256	78	2236	2041
6	1820	892	962	2362	290	256	0	1180	1547	1279
7	1395	2193	902	2262	1305	78	1180	0	1640	1161
8	2382	1590	141	1512	2100	2236	1547	1640	0	2212
9	649	1003	1123	2166	969	2041	1279	1161	2212	0

α	β	p	Результат	Разница
0	1	0	6986	0
0	1	0.1	6986	0
0	1	0.2	6986	0
0	1	0.3	6986	0
0	1	0.4	6986	0
0	1	0.5	6986	0
0	1	0.6	6986	0
0	1	0.7	6986	0
0	1	0.8	6986	0
0	1	0.9	6992	6
0	1	1	6986	0
0.1	0.9	0	6986	0
0.1	0.9	0.1	6992	6
0.1	0.9	0.2	6986	0
0.1	0.9	0.3	6986	0
0.1	0.9	0.4	6986	0
0.1	0.9	0.5	6986	0
0.1	0.9	0.6	6986	0
0.1	0.9	0.7	6986	0
0.1	0.9	0.8	6986	0
0.1	0.9	0.9	7165	179
0.1	0.9	1	6986	0
0.2	0.8	0	6986	0
0.2	0.8	0.1	6986	0
0.2	0.8	0.2	6986	0
0.2	0.8	0.3	6992	6
0.2	0.8	0.4	6992	6
0.2	0.8	0.5	6992	6
0.2	0.8	0.6	6986	0
0.2	0.8	0.7	6992	6
0.2	0.8	0.8	6986	0
0.2	0.8	0.9	6986	0
0.2	0.8	1	6986	0

Рис. 4.2: Таблица коэффициентов. Часть 1

α	β	p	Результат	Разница
0.3	0.7	0	6986	0
0.3	0.7	0.1	6986	0
0.3	0.7	0.2	7139	153
0.3	0.7	0.3	7139	153
0.3	0.7	0.4	6986	0
0.3	0.7	0.5	6986	0
0.3	0.7	0.6	6986	0
0.3	0.7	0.7	6986	0
0.3	0.7	0.8	6992	6
0.3	0.7	0.9	6992	6
0.3	0.7	1	6986	0
0.4	0.6	0	6986	0
0.4	0.6	0.1	6992	6
0.4	0.6	0.2	6986	0
0.4	0.6	0.3	6986	0
0.4	0.6	0.4	6986	0
0.4	0.6	0.5	6992	6
0.4	0.6	0.6	6992	6
0.4	0.6	0.7	6986	0
0.4	0.6	0.8	7139	153
0.4	0.6	0.9	6986	0
0.4	0.6	1	6992	6
0.5	0.5	0	7139	153
0.5	0.5	0.1	6986	0
0.5	0.5	0.2	6986	0
0.5	0.5	0.3	7139	153
0.5	0.5	0.4	6986	0
0.5	0.5	0.5	6986	0
0.5	0.5	0.6	6986	0
0.5	0.5	0.7	6986	0
0.5	0.5	0.8	6986	0
0.5	0.5	0.9	6986	0
0.5	0.5	1	6986	0

Рис. 4.3: Таблица коэффициентов. Часть 2

α	β	p	Результат	Разница
0.6	0.4	0	7139	153
0.6	0.4	0.1	6992	6
0.6	0.4	0.2	6986	0
0.6	0.4	0.3	6986	0
0.6	0.4	0.4	7139	153
0.6	0.4	0.5	6992	6
0.6	0.4	0.6	6986	0
0.6	0.4	0.7	6986	0
0.6	0.4	0.8	6986	0
0.6	0.4	0.9	6992	6
0.6	0.4	1	6986	0
0.7	0.3	0	6986	0
0.7	0.3	0.1	6986	0
0.7	0.3	0.2	6986	0
0.7	0.3	0.3	7139	153
0.7	0.3	0.4	7165	179
0.7	0.3	0.5	7139	153
0.7	0.3	0.6	6992	6
0.7	0.3	0.7	6992	6
0.7	0.3	0.8	6986	0
0.7	0.3	0.9	6992	6
0.7	0.3	1	6986	0
0.8	0.2	0	7139	153
0.8	0.2	0.1	7562	576
0.8	0.2	0.2	6992	6
0.9	0.1	0.2	6992	6
0.9	0.1	0.3	6986	0
0.9	0.1	0.4	7139	153
0.9	0.1	0.5	7329	343
0.9	0.1	0.6	7217	231
0.9	0.1	0.7	7139	153
0.9	0.1	0.8	7217	231
0.9	0.1	0.9	7376	390
0.9	0.1	1	6986	0

Рис. 4.4: Таблица коэффициентов. Часть 3

α	β	p	Результат	Разница
1	0	0	8531	1545
1	0	0.1	8588	1602
1	0	0.2	6986	0
1	0	0.3	7720	734
1	0	0.4	7554	568
1	0	0.5	6992	6
1	0	0.6	7920	934
1	0	0.7	7217	231
1	0	0.8	7874	888
1	0	0.9	7446	460
1	0	1	8119	1133

Рис. 4.5: Таблица коэффициентов. Часть 4

Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. В результате сравнения алгоритма полного перебора и муравьиного алгоритма по времени было получено, что при относительно небольших размерах матрицы смежности (от 2 до 8) алгоритм полного перебора работает быстрее (при размере 2 в 1000 раз). Однако при размере матрицы равном 9, время работы алгоритма полного перебора становится сопоставимым с временем работы муравьиного алгоритма. Более того при размерах матрицы больших 9, время работы алгоритма полного перебора начинает резко возрастать, и становится более чем в 10 раз медленнее муравьиного алгоритма.

Исходя из проведенных исследований, можно сделать вывод, что муравьиный алгоритм решения задачи коммивояжёра выигрывает у алгоритма полного перебора при размерностях матриц равных 9 и более. В случае, когда количество вершин в графе меньше 9, лучше использовать алгоритм полного перебора.

Заключение

В рамках выполнения данной лабораторной работы были достигнуты следующие цели:

- исследованы подходы к решению задачи коммивояжёра;
- проведено сравнение существующих методов поиска кратчайшего пути;
- определены средства программной реализации;
- приведены схемы рассматриваемых алгоритмов;
- определены требования к ПО;
- приведены сведения о модулях ПО;
- приведены экспериментальные замеры временных характеристик реализованных алгоритмов;
- подготовлен отчет по проделанной работе.

Список литературы

[1] Задача коммивояжёра. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения 09.12.2020).

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

[3] Documentation for Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs.

[4] Клиентская документация по Windows для ИТ-специалистов [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/windows/resources/.

[5] Процессор Intel® Core™ i7-4700HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html.