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 № 7
по дисциплине «Анализ алгоритмов»

Тема Решение задачи коммивояжера

Студент Калашников С.Д.

Группа ИУ7-53Б

Преподаватель Волкова Л.Л., Строганов Ю.В.

Москва, 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является изучение задачи коммивояжера и реализация алгоритмов полного перебора и муравьиного алгоритма для ее решения.

Для достижения поставленной цели требуется решить ряд задач:

1) исследовать задачу коммивояжера;

2) реализовать алгоритм полного перебора и муравьиный алгоритм для решения задачи коммивояжера;

3) реализовать данные алгоритмы;

4) провести параметризацию муравьиного алгоритма на нескольких классах данных;

5) провести замеры времени для реализованных алгоритмов;

6) выполнить анализ полученных результатов;

7) по итогам работы составить отчет.

1 Аналитическая часть

1.1 Задача коммивояжера

В задаче коммивояжера рассматривается n городов и матрица попарно различных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз. Иногда условием задачи коммивояжера является возврат в тот город, с которого начинался маршрут.

1.2 Алгоритм полного перебора

Для решения задачи коммивояжера алгоритм полного перебора предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Смысл перебора состоит в том, что перебираются все варианты объезда городов и выбирается оптимальный, что гарантирует точное решение задачи. Однако, при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (сложность алгоритма равна $n!$).

1.3 Муравьиный алгоритм

Муравьиный алгоритм — метод решения задач коммивояжера на основании моделирования поведения колонии муравьев.

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.

Для каждого муравья переход из города i в город j зависит от трех составляющих: памяти муравья, видимости и виртуального следа феромона.

— Память муравья — это список посещенных муравьем городов, заходить в которые еще раз нельзя. Используя этот список, муравей гарантированно не попадет в один и тот же город дважды. Данный список возрастает при совершении маршрута и обнуляется в начале каждой итерации алгоритма.

— Видимость — величина, обратная расстоянию, рассчитывающаяся по формуле 1.1.

$$\eta_{ij} = 1/D_{ij}, \tag{1.1}$$

где D_{ij} — расстояние между городами i и j. Видимость — это локальная статическая информация, выражающая эвристическое желание посетить город j из города i, то есть чем ближе город, тем больше желание посетить его.

— Виртуальный след феромона на ребре (i, j) представляет подтвержденное муравьиным опытом желание посетить город j из города i. След феромона является глобальной и динамичной информацией — она изменяется после каждой итерации алгоритма, отражая приобретенный муравьями опыт.

Формула вычисления вероятности перехода в заданную точку (1.2).

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q=1}^{m} \tau_{iq}^a \eta_{iq}^b}, \text{вершина не была посещена ранее муравьем k,} \\ 0, \text{иначе,} \end{cases} \tag{1.2}$$

где a — параметр влияния длины пути, b — параметр влияния феромона, τ_{ij} — расстояния от города i до j, η_{ij} — количество феромонов на ребре (i, j).

Правило обновления феромона после движения всех муравьев:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}. \tag{1.3}$$

При этом

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^k, \tag{1.4}$$

где

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, \text{ребро посещено k-ым муравьем,} \\ 0, \text{иначе.} \end{cases} \tag{1.5}$$

Существует несколько оптимизаций данного алгоритма, одна из которых введение так называемых элитных муравьев. Элитный муравей усиливает ребра наилучшего маршрута, найденного с начала работы алгоритма. Количество феромона, откладываемого на ребрах наилучшего текущего маршрута T^{+}, принимается равным Q/L^{+}, где L^{+} — длина маршрута T^{+}. Этот феромон побуждает муравьев к исследованию решений, содержащих несколько ребер наилучшего на данный момент маршрута T^{+}. Если в муравейнике есть e элитных муравьев, то ребра маршрута T^{+} будут получать общее усиление:

$$\Delta\tau_e = e * Q/L^{+}. \tag{1.6}$$

Вывод

В этом разделе была изучена задача коммивояжера и используемые для ее решения алгоритмы: полный перебор и муравьиный алгоритм с оптимизацией в виде элитных муравьев.

2 Конструкторская часть

2.1 Описание алгоритмов

На рисунках 2.1 и 2.2 приведены схемы алгоритмов полного перебора и муравьиного алгоритма для решения задачи коммивояжера соответственно.

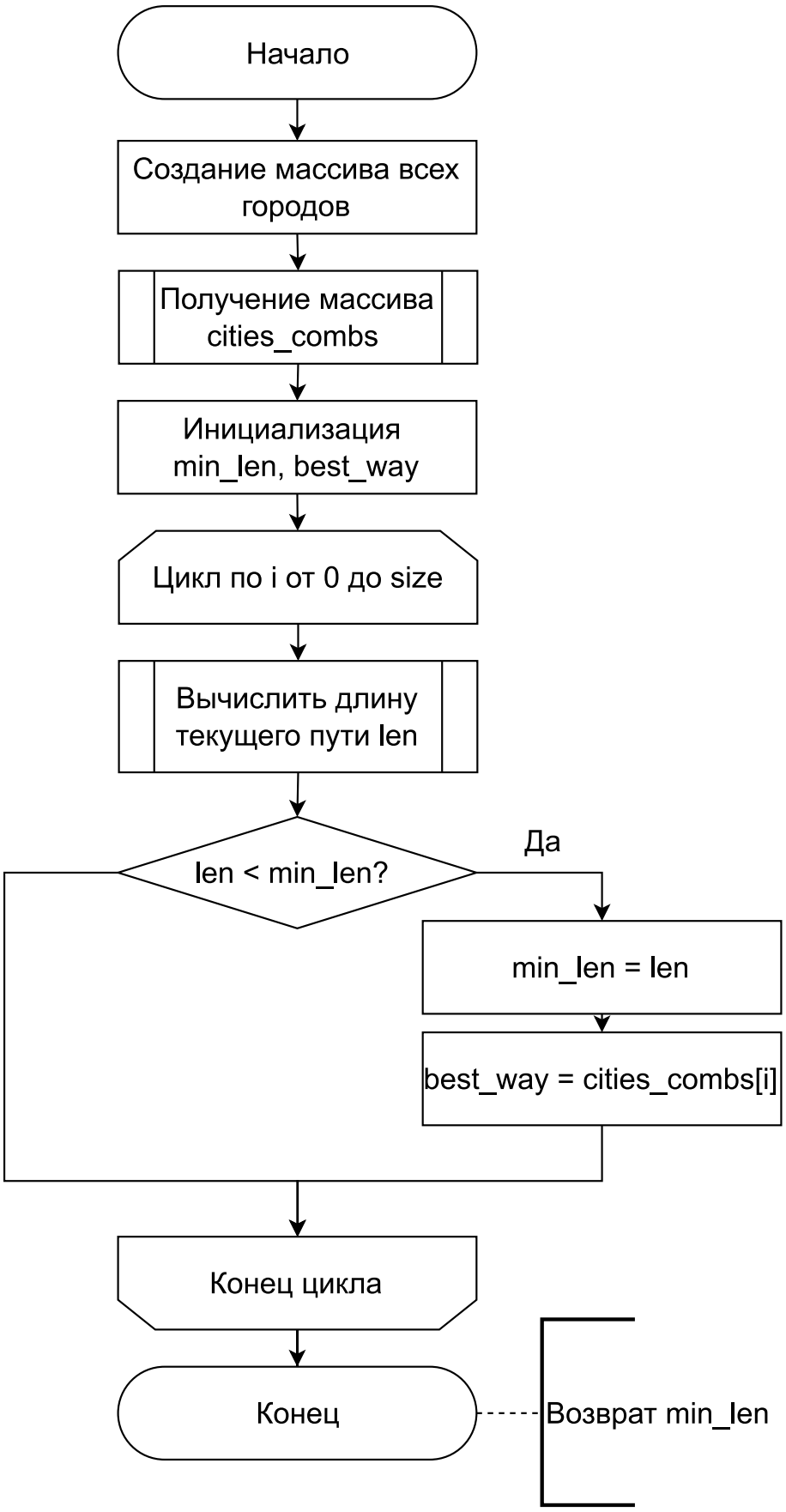

Рис. 2.1 – Схема алгоритма полного перебора

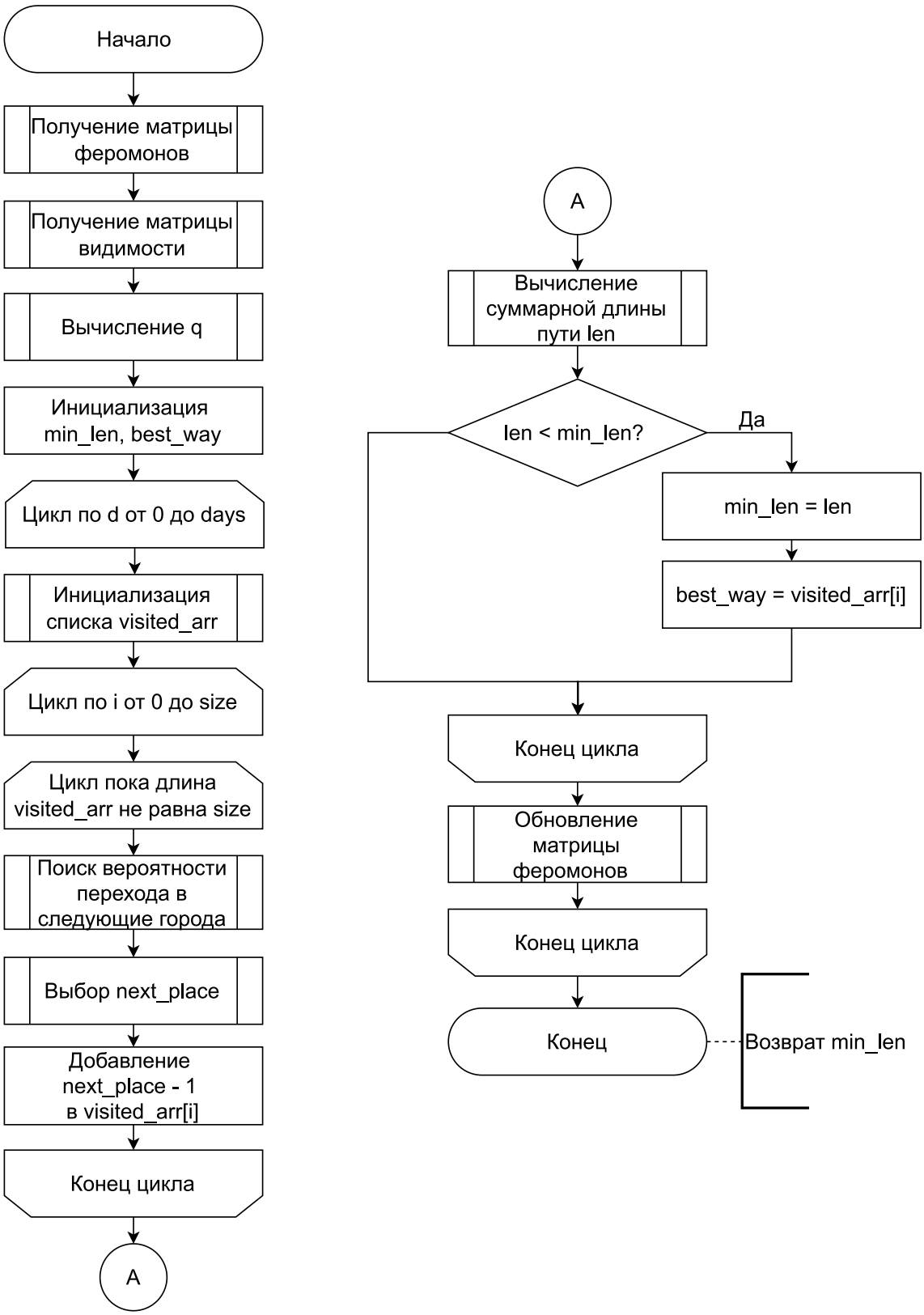

Рис. 2.2 – Схема муравьиного алгоритма

Вывод

В данном разделе были приведены схемы алгоритмов.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритмов.

3.1 Требования к программному обеспечению

Программа должна предоставлять пользователю две функции: поиск гамильтонова цикла с помощью алгоритма полного перебора, поиск гамильтонова цикла с помощью муравьиного алгоритма.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования $c\#$. В текущей лабораторной работе требуется замерить время работы реализаций алгоритмов. Для этого используется класс $Stopwatch$ из библиотеки $System.Diagnostics$.

3.3 Сведения о модулях программы

Программа состоит из следующих классов:

— $AntAlgorithm$ — класс, реализующий муравьиный алгоритм;

— $Program$ — класс, реализующий основную программу;

— $BruteForce$ — класс, реализующий алгоритм полного перебора;

3.4 Реализация алгоритмов

В листингах 3.1–3.2 представлены реализации алгоритмов.

Листинг 3.1 — Реализация алгоритма полного перебора

```
class BruteForce
{
  public static Path GetShortestPath(Map m)
  {
    int[] routeIndexes = new int[m.n - 1];
    for (int i = 0; i < m.n - 1; i++)
    {
      routeIndexes[i] = i + 1;
    }
    var allRoutes = GetAllRoutes(routeIndexes);
    Path shortestPath = new Path(int.MaxValue);
    foreach (List<int> path in allRoutes)
    {
      path.Insert(0, 0);
      path.Add(0);
      Path cur = new Path(m, path.ToArray());
      if (cur.distance < shortestPath.distance)
      shortestPath = cur;
    }

    return shortestPath;
  }

  public static List<List<T>> GetAllRoutes<T>(IList<T> arr, List<
    List<T>> res = null, List<T> current = null)
```

```
25    {
26      if (res == null)
27      res = new List<List<T>>();
28      if (arr.Count == 0)
29      {
30        res.Add(current);
31        return res;
32      }
33      for (int i = 0; i < arr.Count; i++)
34      {
35        List<T> lst = new List<T>(arr);
36        lst.RemoveAt(i);
37        List<T> next;
38        if (current == null)
39        next = new List<T>();
40        else
41        next = new List<T>(current);
42        next.Add(arr[i]);
43        GetAllRoutes(lst, res, next);
44      }
45      return res;
46    }
47  }
```

Листинг 3.2 — Реализация муравьиного алгоритма

```
1       public static Path GetShortestPath(Map map, int nIter,
            double alpha, double beta, double Q, double ro)
2   {
3     Path minPath = new Path(int.MaxValue);
4     int n = map.n;
5     double[][] pheromones = Program.InitMatr(n, 0.1);
6     List<Ant> ants = new List<Ant>();
7     for (int t = 0; t < nIter; t++)
8     {
9       ants = InitAnts(map, n);
10      for (int i = 0; i < n - 1; i++)
11      {
12        double[][] pheromonesIter = Program.InitMatr(n, (double)0);
13        for (int j = 0; j < ants.Count(); j++)
```

```
14        {
15          double sumChance = 0, chance = 0;
16          bool flag = false;
17          Ant curAnt = ants[j];
18          int curCity = curAnt.path.route.Last();
19          for (int sityId = 0; sityId < n; sityId++)
20          {
21            if (curAnt.visited[sityId] == false)
22            {
23              sumChance += Math.Pow(pheromones[curCity][sityId],
                    alpha) * Math.Pow(1 / (map.distance[curCity,sityId
                    ]), beta);
24              flag = true;
25            }
26          }
27          double x = r.NextDouble();
28          int k = 0;
29          for (; x > 0; k++)
30          {
31            if (curAnt.visited[k] == false)
32            {
33              chance = Math.Pow(pheromones[curCity][k], alpha) *
                    Math.Pow(1 / (map.distance[curCity,k]), beta);
34              chance /= sumChance;
35              x -= chance;
36            }
37          }
38          k--;
39          ants[j].VisitedTown(k);
40          pheromonesIter[curCity][k] += Q / (map.distance[curCity,k
                ]);
41        }
42        for (int ii = 0; ii < n; ii++)
43        for (int j = 0; j < n; j++)
44        pheromones[ii][j] = (1 - ro) * pheromones[ii][j] +
              pheromonesIter[ii][j];
45      }
46
47      foreach (Ant a in ants)
48      {
```

```
49          a.VisitedTown(a.iStartTown);
50          if (a.GetDistance() < minPath.distance)
51          {
52              minPath = a.path;
53          }
54      }
55    }
56    return minPath;
57  }
58  public static List<Ant> InitAnts(Map m, int n)
59  {
60    List<Ant> ants = new List<Ant>();
61    for (int i = 0; i < n; i++)
62    {
63      ants.Add(new Ant(m, 0));
64    }
65    return ants;
66  }
```

Вывод

В данном разделе были рассмотрены листинги используемых алгоритмов.

4 Исследовательская часть

В данном разделе будет проведен сравнительный анализ времени работы реализаций алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее:

— операционная система Windows 11 Pro Версия 22H2 (22621.674) [1];

— память 16 ГБ;

— процессор 11th Gen Intel(R) Core(TM) i5-11400 2.59 ГГц [2], 6 физических и 12 логических ядер.

При тестировании компьютер был включен в сеть электропитания. Во время замеров процессорного времени устройство было нагружено только встроенными приложениями окружения, а также системой тестирования.

4.2 Время выполнения реализаций алгоритмов

На рис. 4.1 и рис. 4.2 показаны результаты замеров времени.

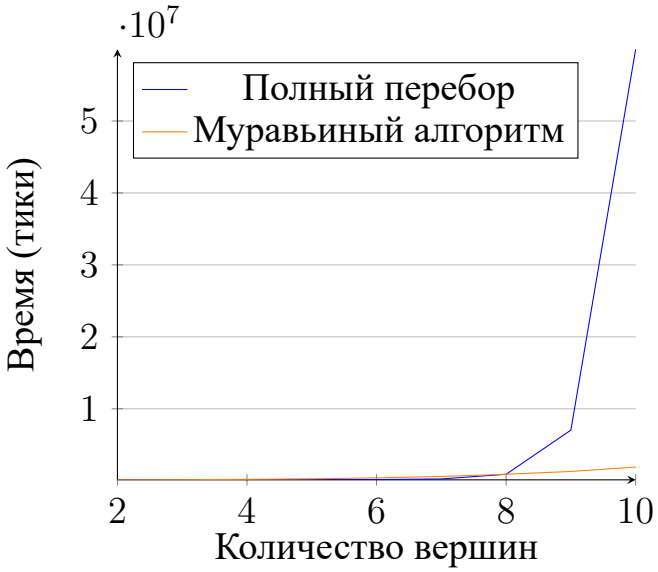

Рис. 4.1 – Сравнение времени работы алгоритмов при увеличении размера графа

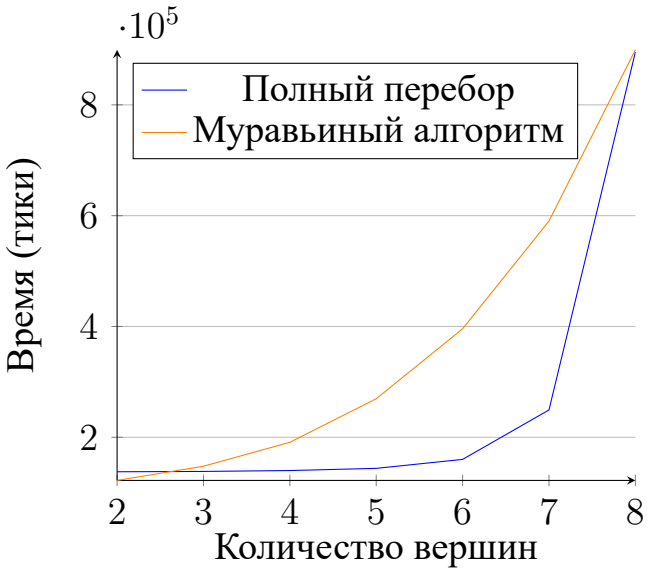

Рис. 4.2 – Сравнение времени работы алгоритмов на малых размерах графа

4.3 Параметризация муравьиного алгоритма

Для различных значений параметров α, β, ρ и t_{max} для каждой из нескольких матриц смежности с помощью муравьиного алгоритма и перебора была найдена некоторая длина маршрута. Далее выбраны наилучшие сочетания параметров муравьиного алгоритма на этих данных.

Параметр α менялся от 0 до 10, параметр ρ менялся от 0.1 до 0.9, параметр t_{max} менялся от 5 до 100.

Итого были выявлены оптимальные сочетания параметров (представлены в таблице 4.1):

Таблица 4.1 — Результаты решения задачи параметризации

α	β	ρ	t_{max}
4	6	0.6	20
6	4	0.3	40
1	9	0.7	50
6	4	0.9	70
3	7	0.6	80
6	4	0.25	90

4.4 Вывод

По полученным результатам иследования можно сделать вывод, что на больших размерностях (размер графа больше 9) полный перебор крайне медленен относительно муравьиного алгоритма.

ЗАКЛЮЧЕНИЕ

Цель, которая была поставлена в начале лабораторной работы, была достигнута: изучена задача коммивояжера и реализован алгоритм полного перебора и муравьиный алгоритм для ее решения.

В ходе выполнения лабораторной работы были решены все задачи:

1) исследована задача коммивояжера;

2) описаны алгоритм полного перебора и муравьиный алгоритм для решения задачи коммивояжера;

3) реализованы данные алгоритмы;

4) проведена параметризация муравьиного алгоритма на нескольких классах данных;

5) проведены замеры времени для реализованных алгоритмов;

6) выполнен анализ полученных результатов;

7) по итогам работы составлен отчет.

Временной анализ показал, что неэффективно использовать полный перебор на графе размера больше 10.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Windows 11, version 22H2 [Эл. ресурс]*. Режим доступа: https://clck.ru/32NCXx (дата обращения: 14.10.2022).
2. *Процессор Intel® Core™ i7 [Эл. ресурс]*. Режим доступа: https://clck.ru/yeQa8 (дата обращения: 14.10.2022).