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»

Отчёт

по лабораторной работе № 6

Название: Муравьиный алгоритм. Решение задачи коммивояжера.

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ Д.О. Склифасовский

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение **3**

1 Аналитический раздел **4**

1.1 Общие сведения 4

1.2 Алгоритм полного перебора 4

1.3 Муравьиный алгоритм 5

1.4 Вывод 6

2 Конструкторский раздел **7**

2.1 Алгоритм полного перебора 7

2.2 Муравьиный алгоритм 10

2.3 Вывод 10

3 Технологический раздел **11**

3.1 Общие требования 11

3.2 Сведения о модулях программы 11

3.3 Листинг кода программы 12

3.4 Алгоритм полного перебора 12

3.5 Муравьиный алгоритм 13

3.6 Вывод 15

4 Экспериментальный раздел **16**

4.1 Замер времени 16

4.2 Параметризация 17

4.3 Вывод 17

Заключение **18**

Литература **19**

Введение

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

Целью данной лабораторной работы является создание приложения для наглядного представления работы муравьиного алгоритма.

Задачи данной лабораторной работы:

1) изучить существующие методы решения задачи;
2) реализовать алгоритм полного перебора и муравьиный алгоритм;
3) сравнить время работы этих алгоритмов.

1 Аналитический раздел

Задача коммивояжера занимает особое место в комбинаторной оптимизации и исследовании операций. Исторически она была одной из тех задач, которые послужили толчком для развития этих направлений. С точки зрения приложений, она не представляет интерес. Куда важнее её обобщения для транспорта и логистики, когда несколько транспортных средств ограниченной грузоподъемности должны обслуживать клиентов, посещая их в заданные временные окна.

1.1 Общие сведения

В задаче коммивояжера рассматривается 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

Под полным перебором понимается методика разрешения задач математики путем рассмотрения всех возможных вариантов. Уровень сложности при полном переборе напрямую связан с количеством допустимых решений задачи. В случае, когда область решений огромна, время полного перебора может исчисляться десятками и даже сотнями лет, и при этом итоговый результат возможно ещё не будет найден. Все задачи класса NP могут быть решены при помощи полного перебора.

1.3 Муравьиный алгоритм

Идея муравьиного алгоритма - моделирование поведения муравьев, связанное с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя кратчайший путь.

С учетом особенности задачи коммивояжера, можно описать правила поведения муравьев при выборе пути:

1) муравьи имеют собственную "память" - у муравья уже есть список посещенных городов;

2) муравьи обладают "зрением" - видимость есть эвристическое желание посетить город j, если муравей находится в городе i;

$$\eta_{ij} = \frac{1}{D_{ij}}$$

3) муравьи обладают "обонянием" - они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьев.

На этом основании можно сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода k-ого муравья из города i в город j:

$$\begin{cases} P_{ij,k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in J_{ik}} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}, & j \in J_{i,k}; \\ P_{ij,k}(t) = 0, & j \notin J_{i,k} \end{cases}$$

Где α, β - параметры, задающие веса следа феромона.

Пройдя ребро (i, j), муравей откладывает на нем некоторое количество феромона. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t, а $L_k(t)$ - длина этого маршрута. Пусть Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество

феромона может быть задано в виде:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{cases}$$

Правила внешней среды в первую очередь определяют испарение феромона. Пусть $p \in [0,1]$ есть коэффициент испарения, тогда правило испарения имеет вид:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t),$$

где m - количество муравьев в колонии.

1.4 Вывод

В данном разделе представлены общие сведения задачи коммивояжера, описание алгоритмов полного перебора и муравьиный.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов полного перебора и маравьиный.

2.1 Алгоритм полного перебора

Функция GetRoute

На рисунке 1 показана основная функция GetRoute.

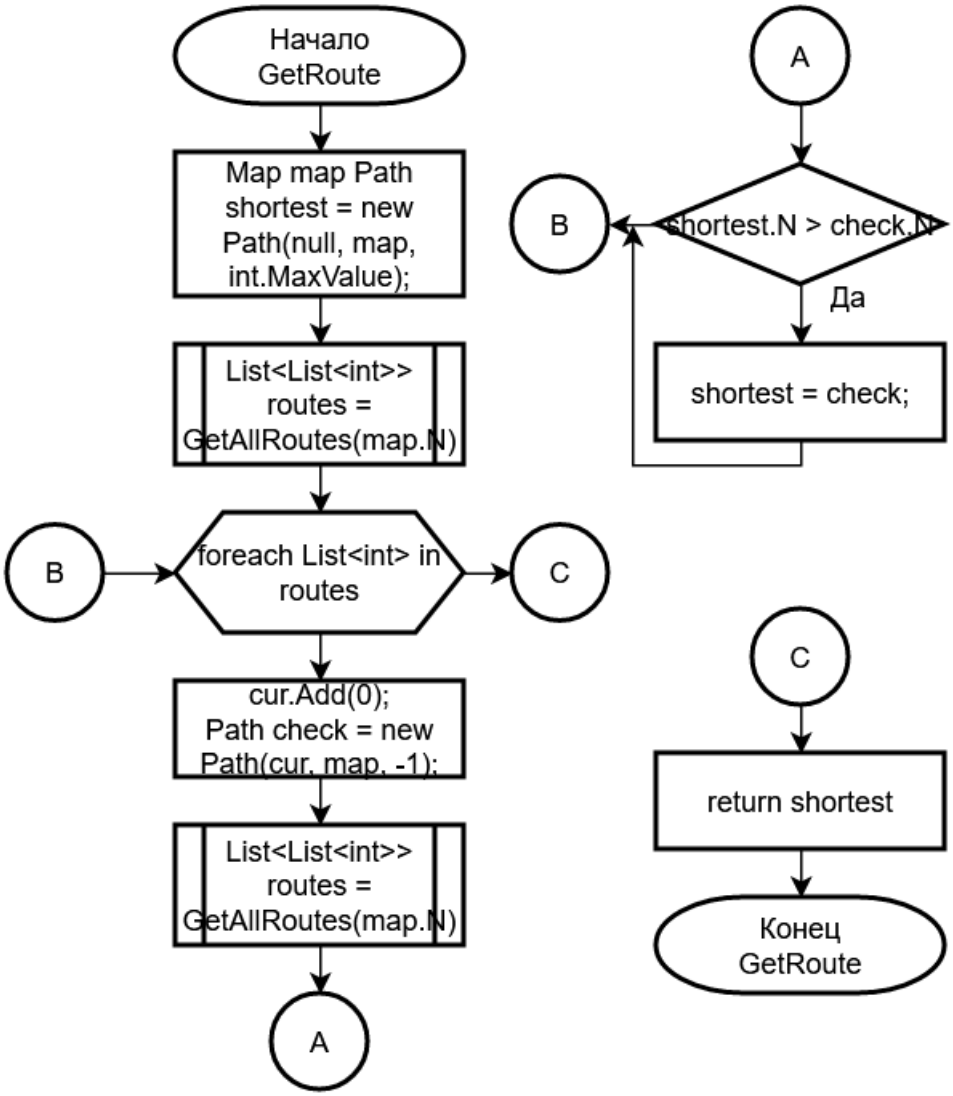

Рисунок 1 – Функция GetRoute

Функция GetAllRoutes

На рисунке 2 показана функция GetAllRoutes.

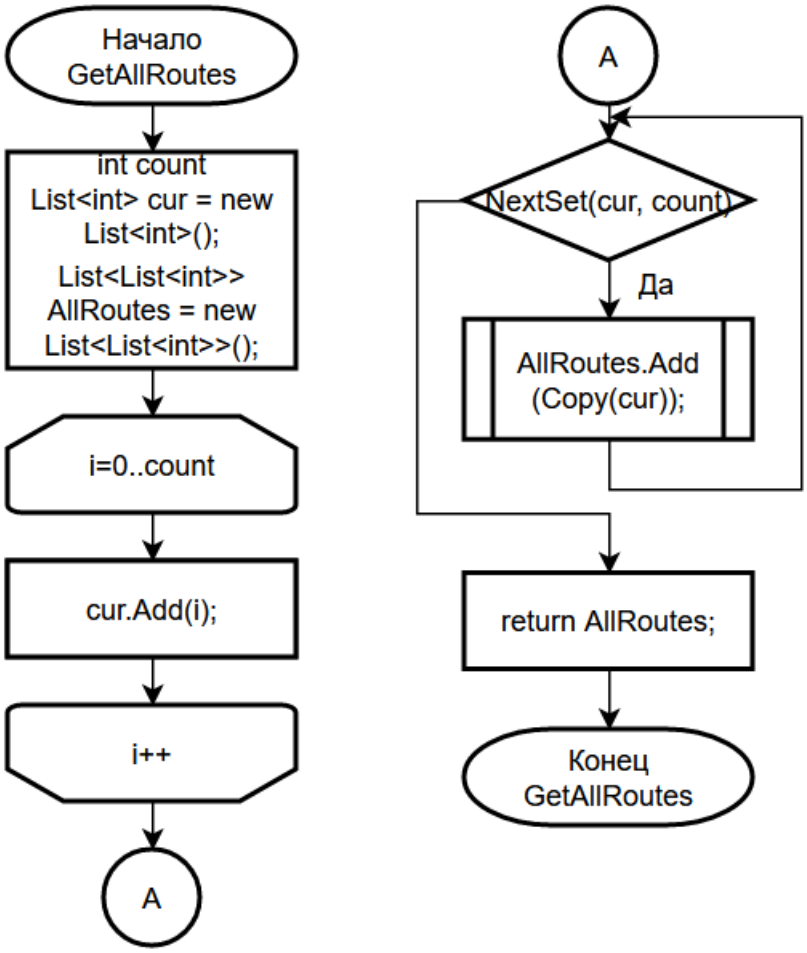

Рисунок 2 – Функция GetAllRoutes

Функция NextSet

На рисунке 3 показана функция NextSet.

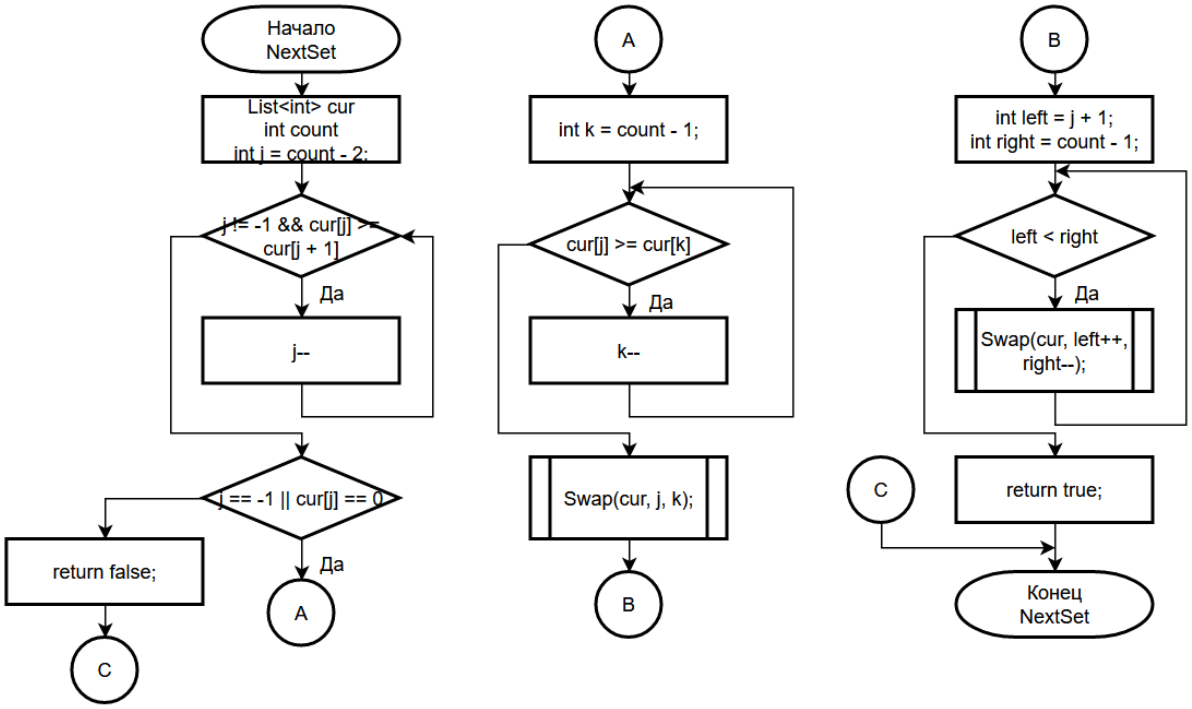

Рисунок 3 – Функция NextSet

2.2 Муравьиный алгоритм

На рисунке 4 показан муравьиный алгоритм.

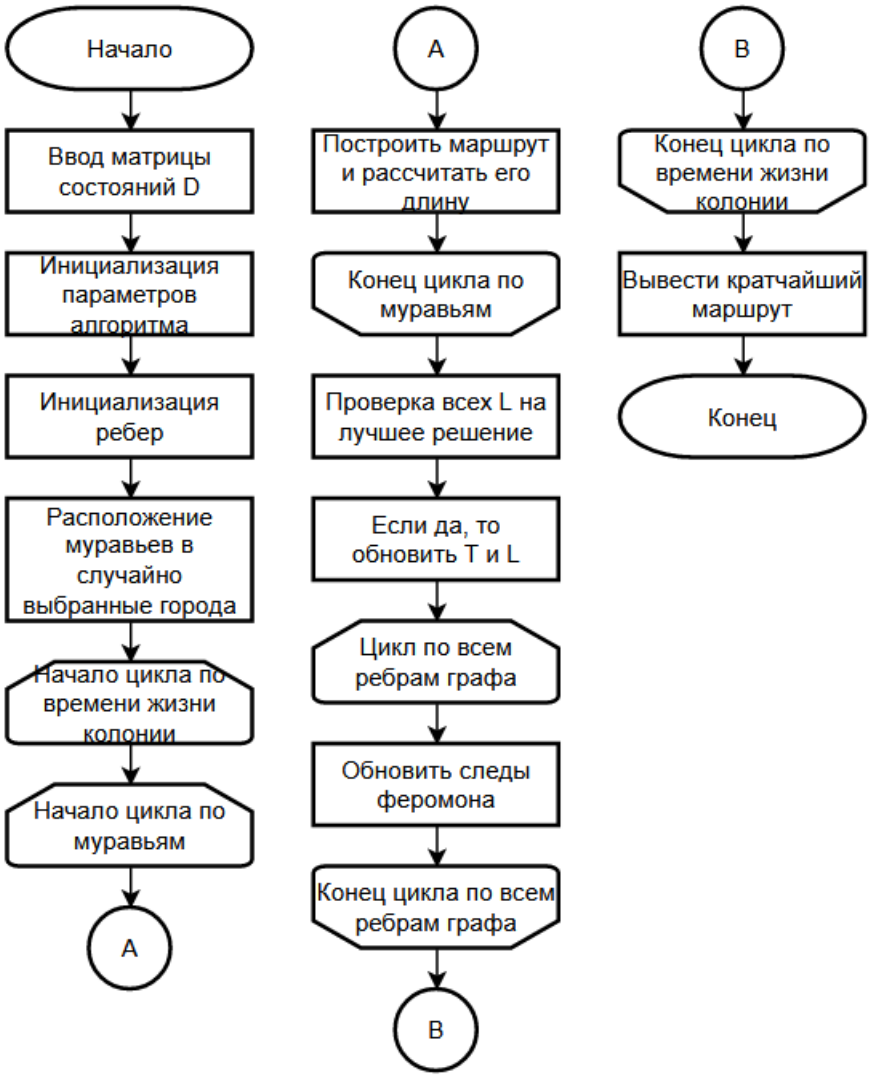

Рисунок 4 – Муравьиный алгоритм

2.3 Вывод

В данном разделе были разработаны алгоритмы полного перебора и муравьиный.

3 Технологический раздел

В данном разделе даны общие требования к программе, средства реализации и сама реализация алгоритмов.

3.1 Общие требования

Требования к программе

1) алгоритм полного перебора должен возвращать кратчайший путь;

2) работа программы должна быть корректна.

В качестве языка программирования был выбран C#[1], так как я знаком с данным языком программирования, имею представление о способах тестирования программы.

Средой разработки Visual Studio.[2]

Для замеров процессорного времени используется функция $Stopwatch$.[3][4]

3.2 Сведения о модулях программы

Программа состоит из:

1) Program.cs - главный файл программы, в котором располагается точка входа в программу;

2) BruteForce.cs - файл класса алгоритма полного перебора;

3) AntAlgorithm.cs - файл класса муравьиного алгоритма;

4) Map.cs - файл класса, содержащего матрицу смежности;

5) Path.cs - файл класса, содержащий путь и методы для работы с ним.

3.3 Листинг кода программы

3.4 Алгоритм полного перебора

В листинге 1 представлена основная функция GetRoute.

Листинг 1 – GetRoute

```
public static Path GetRoute(Map map)
{
    Path shortest = new Path(null, map, int.MaxValue);
    foreach (List<int> cur in GetAllRoutes(map.N))
    {
        cur.Add(0);
        Path check = new Path(cur, map, -1);
        check.GetDistance();
        if (shortest.N > check.N)
            shortest = check;
    }
    return shortest;
}
```

В листинге 2 представлена функция GetAllRoutes.

Листинг 2 – GetAllRoutes

```
private static List<List<int>> GetAllRoutes(int count)
{
    List<List<int>> AllRoutes = new List<List<int>>();
    List<int> cur = new List<int>();
    for (int i = 0; i < count; i++)
        cur.Add(i);
    while (NextSet(cur, count))
        AllRoutes.Add(Copy(cur));
    return AllRoutes;
}
```

В листинге 3 представлена функция NextSet.

Листинг 3 – NextSet

```
private static bool NextSet(List<int> cur, int count)
{
    int j = count - 2;
    while (j != -1 && cur[j] >= cur[j + 1]) j--;
    if (j == -1 || cur[j] == 0)
        return false;
    int k = count - 1;
    while (cur[j] >= cur[k])
        k--;
    Swap(cur, j, k);
    int left = j + 1;
    int right = count - 1;
    while (left < right)
        Swap(cur, left++, right--);
    return true;
}
```

3.5 Муравьиный алгоритм

В листинге 4 представлен муравьиный алгоритм.

Листинг 4 – NextSet

```
public static Path GetRoute(Map map, int maxTime, double alpha,
   double beta, double Q, double pho)
{
    Random r = new Random();

    Path shortest = new Path(null, map, int.MaxValue);

    int count = map.N;
    double[,] pher = InitPheromone(0.1, count);

    List<Ant> ants = null;

```

```
12      for (int time = 0; time < maxTime; time++)
13      {
14          ants = InitAnts(map);
15          for (int i = 0; i < count - 1; i++)
16          {
17              double[,] deltaPher = InitPheromone(0, count);
18              foreach (Ant ant in ants)
19              {
20                  int curTown = ant.LastVisited();
21
22                  double sum = 0;
23                  for (int town = 0; town < count; town++)
24                  {
25                      if (!ant.IsVisited(town))
26                      {
27                          double tau = pher[curTown, town];
28                          double eta = 1 / map[curTown, town];
29                          sum += Math.Pow(tau, alpha) * Math.Pow(eta
                                , beta);
30                      }
31                  }
32
33                  double check = r.NextDouble();
34                  int newTown = 0;
35                  for (; check > 0; newTown++)
36                  {
37                      if (!ant.IsVisited(newTown))
38                      {
39                          double tau = pher[curTown, newTown];
40                          double eta = 1 / map[curTown, newTown];
41                          double chance = Math.Pow(tau, alpha) *
                                Math.Pow(eta, beta) / sum;
42                          check -= chance;
43                      }
44                  }
```

```
45                  newTown--;
46                  ant.VisitTown(newTown);
47                  deltaPher[curTown, newTown] += Q / map[curTown,
                        newTown];
48              }
49              for (int k = 0; k < count; k++)
50                  for (int t = 0; t < count; t++)
51                      pher[k, t] = (1 - pho) * pher[k, t] +
                            deltaPher[k, t];
52          }
53          foreach (Ant ant in ants)
54          {
55              ant.VisitTown(ant.Start);
56
57              if (ant.GetDistance() < shortest.N)
58                  shortest = ant.GetPath();
59          }
60      }
61      return shortest;
62  }
```

3.6 Вывод

В данном разделе были представлены общие требования к программе, сведения о модулях и листинги алгоритмов полного перебора и муравьиный.

4 Экспериментальный раздел

В данном разделе приведен анализ времени работы алгоритмов.

4.1 Замер времени

Проводится замер времени при разных размерах матрица смежности (разном количестве вершин графа). На рисунке 5 показан результат замеров.

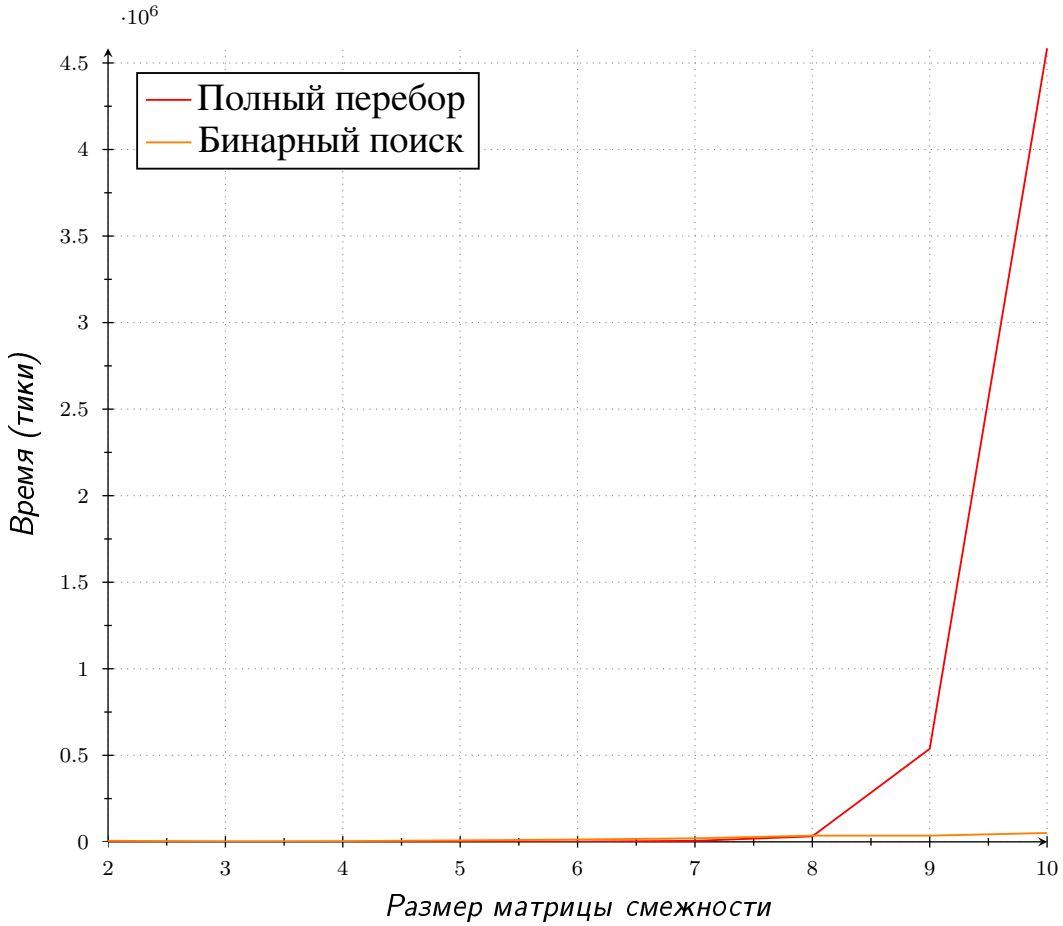

Рисунок 5 – Результаты замеров процессорного времени.

Можно увидеть, что муравьиный алгоритм намного быстрее, чем алгоритм полного перебора, при большом размере графа. До 7 вершин алгоритм полного перебора работает практически одинаково, а далее начинает резко увеличиваться время работы.

4.2 Параметризация

Проводится выборка лучших сочетаний значений α, β, ρ. На рисунке 6 можно увидеть результат тестирования.

0,1	0,0	0,0	0
0,1	0,0	0,3	0
0,1	0,0	0,6	0
0,1	0,0	0,9	0
0,2	0,0	0,0	0
0,2	0,0	0,9	0
0,3	0,0	0,3	0
0,3	0,0	0,6	5
0,3	0,0	0,9	5
0,4	0,0	0,0	0
0,4	0,0	0,3	0
0,4	0,0	0,6	0
0,5	0,0	0,0	0
0,5	0,0	0,3	0
0,5	0,0	0,6	0
0,5	0,0	0,9	0
0,6	0,0	0,0	0
0,6	0,0	0,3	0
0,6	0,0	0,6	0
0,7	0,0	0,0	0
0,7	0,0	0,3	5
0,7	0,0	0,6	5
0,7	0,0	0,9	0
0,8	0,0	0,0	0
0,8	0,0	0,3	0
0,8	0,0	0,6	0
0,9	0,0	0,0	0
0,9	0,0	0,3	5
0,9	0,0	0,6	0
0,9	0,0	0,9	5

Рисунок 6 – Результат тестирования

4.3 Вывод

По результатам тестирования можно сделать вывод, что на небольших размерах графа алгоритм полного перебора работает быстрее, но на больших размерах он во много раз медленнее муравьиного алгоритма.

Наиболее правильные результаты получаются при $\alpha = 0.5..0.9$, $\beta = 0$.

Заключение

В ходе выполнения лабораторной работы было создано приложения для наглядного представления работы муравьиного алгоритма. Были выполнены следующие задачи:

1) изучены существующие методы решения задачи;
2) реализованы алгоритмы полного перебора и муравьиный алгоритм;
3) были проведены эксперименты.

При размерах графа больше 7-8 выгоднее использовать муравьиный алгоритм, так как он в несколько раз быстрее алгоритма полного перебора. Но не всегда муравьиный алгоритм дает минимальный результат.

Литература

1. Документация по C#. -URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 24.10.2020). -Текст: электронный.

2. Документация по семейству продуктов Visual Studio. -URL: https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019 (дата обращения: 01.10.2020). -Текст: электронный.

3. Stopwatch Класс. -URL: https://goo.su/2e99 (дата обращения: 24.10.2020). -Текст: электронный.

4. Под капотом у Stopwatch. -URL: https://habr.com/ru/post/226279/ (дата обращения: 24.10.2020). Текст: электронный.