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 Методы решения задачи коммивояжёра

Студент Пискунов П.

Группа ИУ7-56Б

Преподаватель Волкова Л.Л., Строганов Д.В.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача коммивояжера — задача комбинаторной оптимизации, цель которой — нахождение самого выгодного маршрута, проходящего через указанные точки по одному разу с возвращением в исходную точку [1].

Задача рассматривается как в симметричном, так и в ассиметричном варианте. В асимметричном варианте задача представляется в виде ориентированного графа, где веса рёбер между одними и теми же вершинами зависят от направления движения [1].

Целью данной лабораторной работы является параметризация метода решения задачи коммивояжера на основе муравьиного метода.

Для достижения цели, требуется выполнить следующие задачи:

1) описать задачу коммивояжера;
2) описать методы решения задачи коммивояжера: метод полного перебора и метод на основе муравьиного алгоритма;
3) реализовать данные алгоритмы;
4) провести сравнительный анализ по времени реализованного алгоритма;
5) подготовить отчет о лабораторной работе.

1 Аналитическая часть

1.1 Методы решения задачи коммивояжера

Полный перебор. Этот метод заключается в переборе всех возможных маршрутов в графе и выборе кратчайшего из них. Сложность такого алгоритма — $O(n!)$ [2], где n — количество городов.

Муравьиный алгоритм. Данный метод основан на принципах поведения колонии муравьев [2].

Муравьи, двигаясь из муравейника в поисках пищи, откладывают феромоны на своем пути. При этом, чем больше плотность феромона, тем короче путь, и, соответственно, чем длиннее путь, тем быстрее феромон испарится, и его плотность будет меньше [3].

Со временем муравьи оставят наибольшее количество феромонов на самом коротком участке пути, что приведет к тому, что большинство муравьев выберет этот самый короткий путь, и, следовательно, они оставят еще больше феромонов на нем, что уменьшит вероятность выбора других маршрутов [3].

При сведении алгоритма к математическим формулам, сначала определяется целевая функция:

$$n = \frac{1}{D}, \tag{1.1}$$

где D — расстояние до заданного пункта [3].

Далее вычисляются вероятности перехода в заданную точку:

$$P = \frac{t^e \cdot n^b}{\sum_{i=1}^{m} t_i^a \cdot n_i^b}, \tag{1.2}$$

где a,b — настраиваемые параметры,
t — концентрация феромона [3].

При $a = 0$ выбирается ближайший город и алгоритм становится «жадным» (выбирает только оптимальные или самые короткие расстояния), при $b = 0$ будут учитываться только след феромона, что может привести к сужению пространства поиска оптимального решения [3].

Последним производится обновление феромона:

$$t_{i+1} = (1 - p) \cdot t_i + \frac{Q}{L_0}, \tag{1.3}$$

где p настраивает скорость испарения феромона,
Q настраивает концентрацию нанесения феромона,
L_0 — длина пути на определенном участке [3].

Последовательность вышеизложенных действий повторяется, пока не будет найден оптимальный маршрут.

Одной из модификаций муравьиного алгоритма является элитарная муравьиная система. При таком подходе искусственно вводятся «элитные» муравьи, усиливающие уровень феромонов, оптимального на данный момент маршрута [4].

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также способы её решения: полным перебором и муравьиным алгоритмом.

2 Конструкторская часть

В этом разделе будут представлены требования к программному обеспечению (ПО) и схема алгоритмов.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

1. наличие интерфейса для выбора действий;
2. возможность выбора алгоритма решения задачи коммивояжера.

2.2 Разработка алгоритма полного перебора

На рисунке 2.1 представлен алгоритм полного перебора.

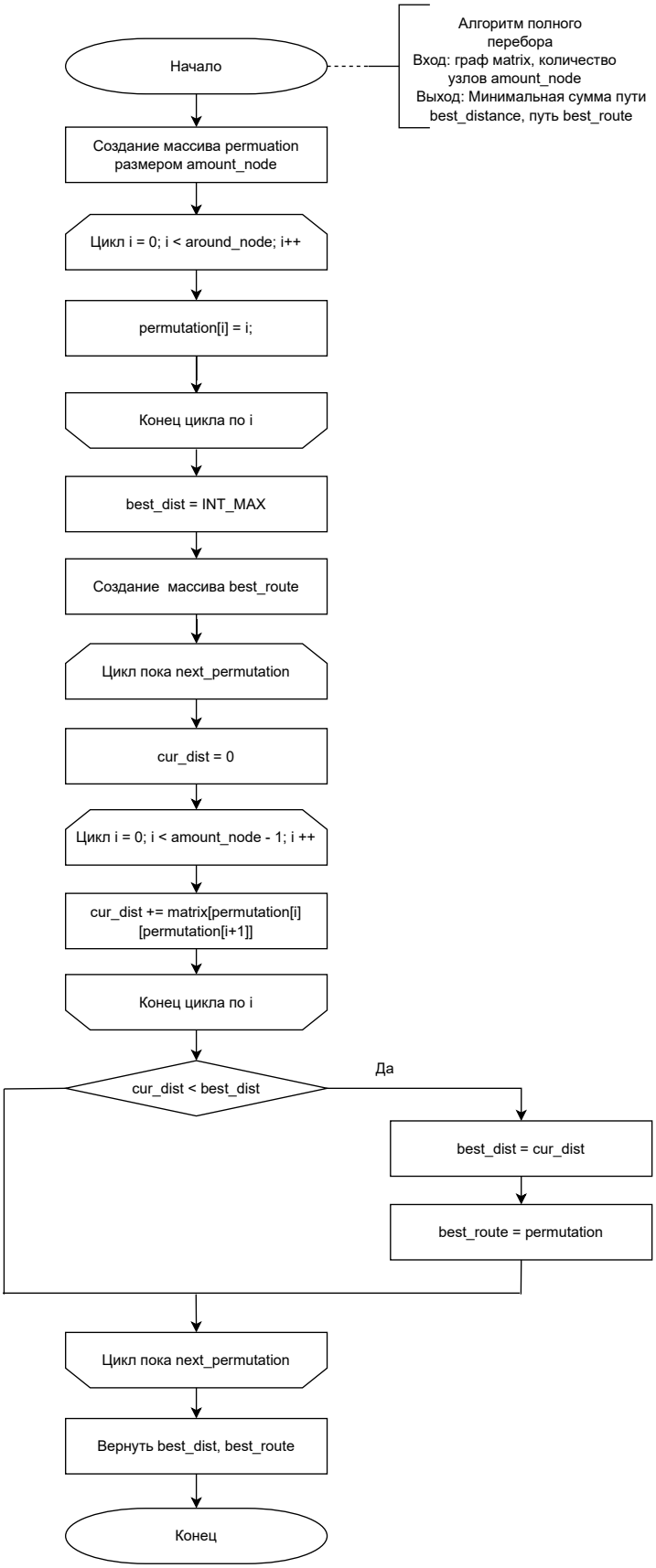

Рисунок 2.1 – Алгоритм полного перебора

2.3 Разработка муравьиного алгоритма

На рисунке 2.2 представлен алгоритм вычисления среднего значения расстояний между всеми парами.

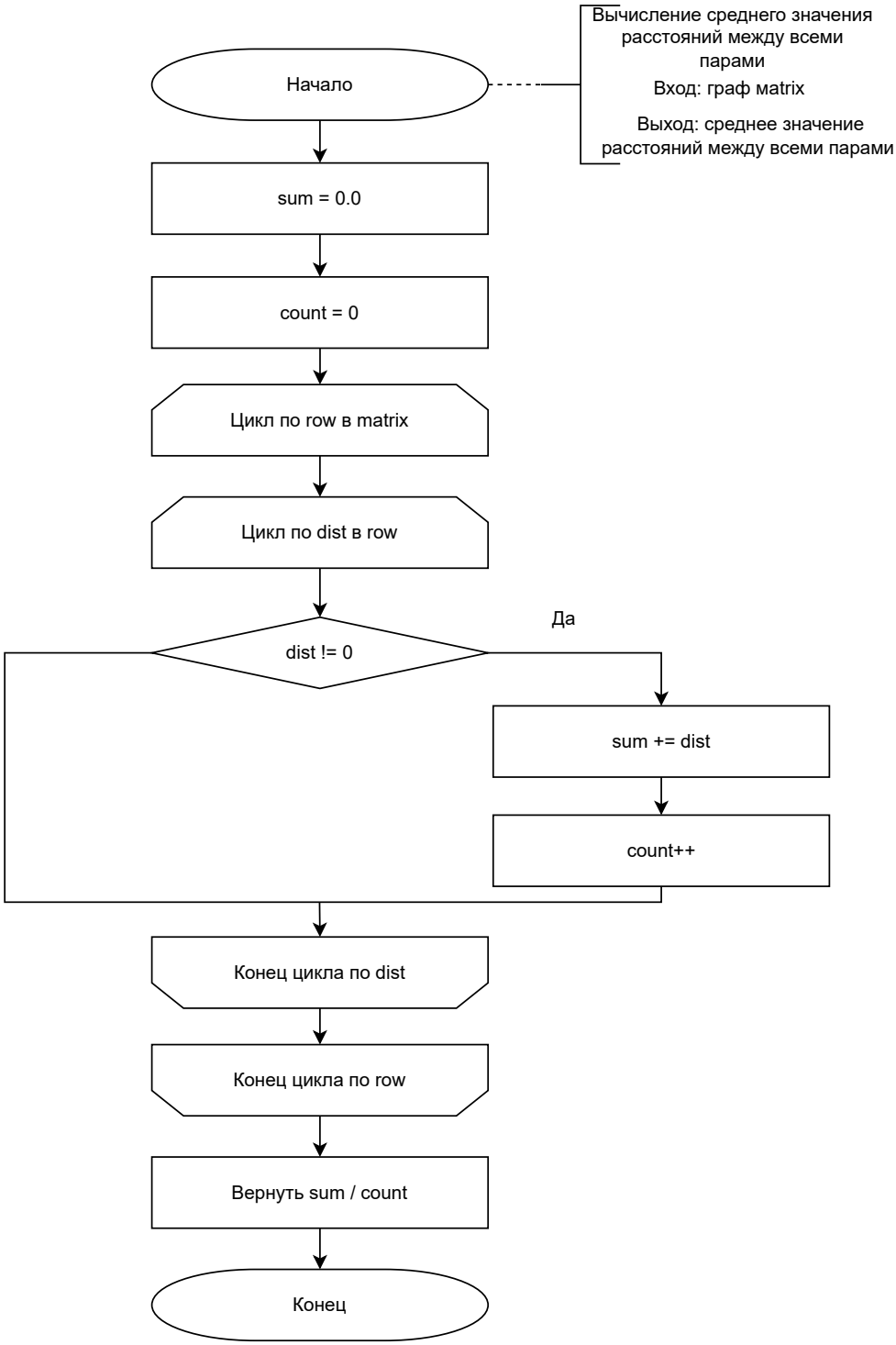

Рисунок 2.2 – Алгоритм вычисления среднего значения расстояний между всеми парами

На рисунке 2.3 представлен алгоритм выбора следующего города, куда муравей должен переместиться.

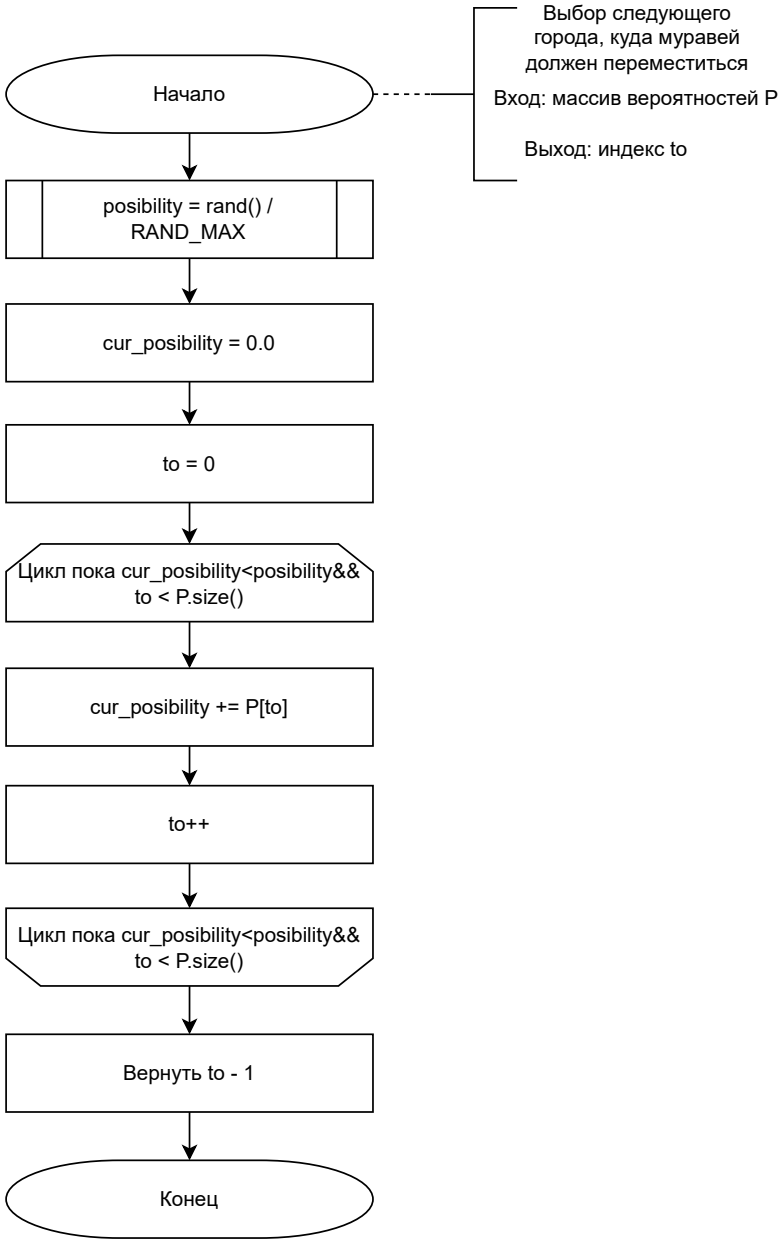

Рисунок 2.3 – Алгоритм выбора следующего города, куда муравей должен переместиться

На рисунке 2.4 представлен алгоритм вычисления пути, пройденного муравьем.

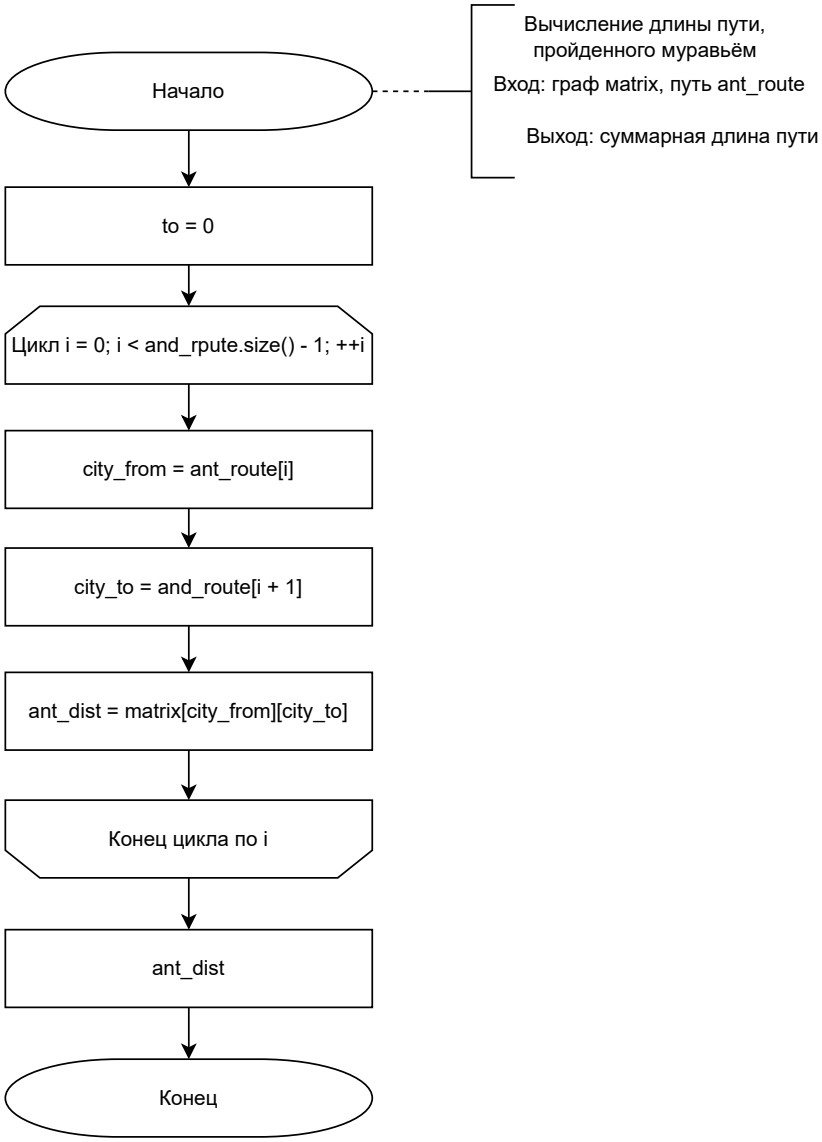

Рисунок 2.4 – Алгоритм вычисления пути, пройденного муравьем

На рисунке 2.5 представлен муравьиный алгоритм.

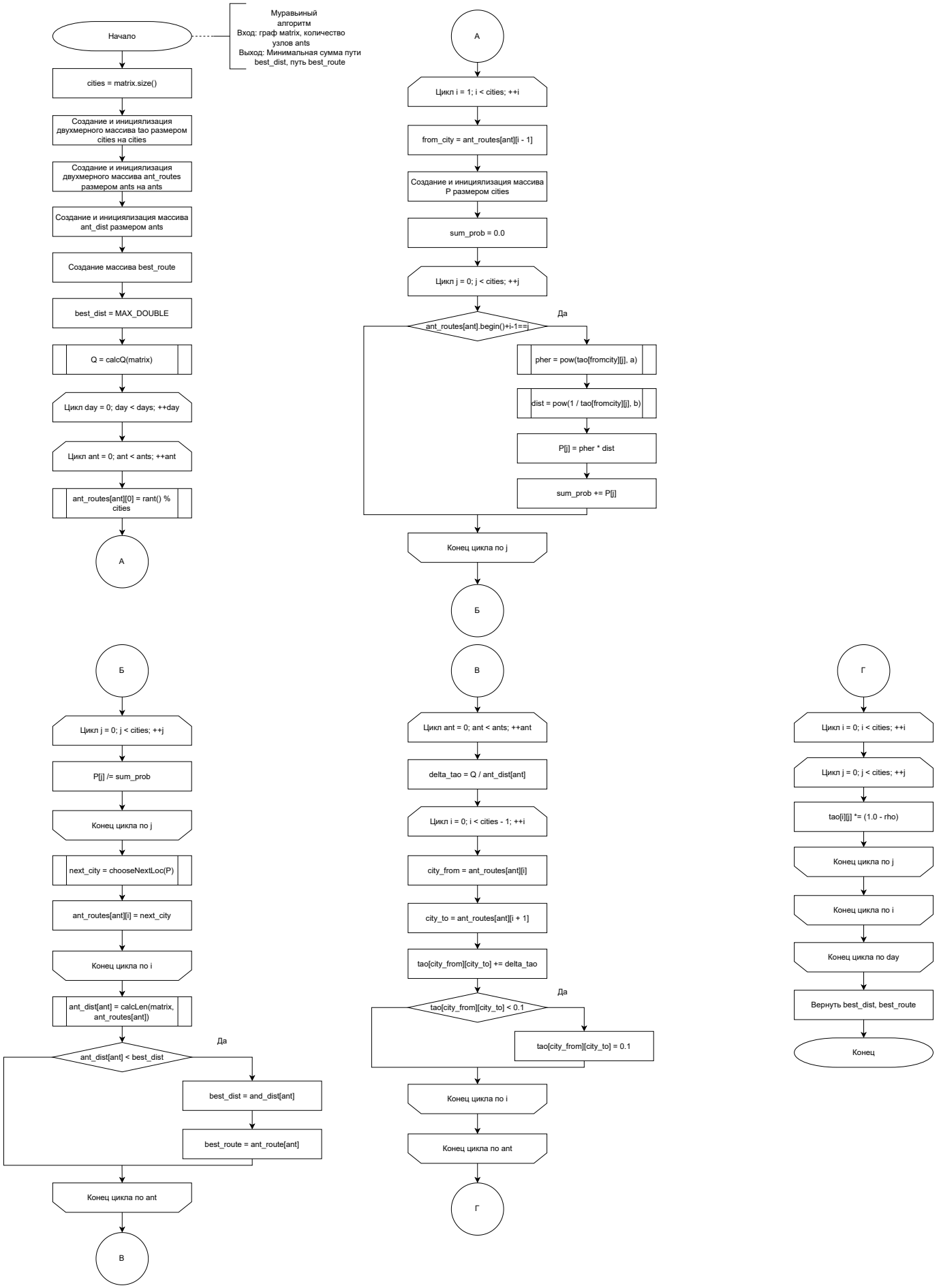

Рисунок 2.5 – Муравьиный алгоритм

Вывод

В данном разделе были перечислены требования к программному обеспечению и построены схемы алгоритмов решения задачи коммивояжера.

3 Технологическая часть

В этом разделе предоставляются листинги реализованных алгоритмов и осуществляется выбор средств реализации.

3.1 Выбор средств реализации

Для выполнения данной лабораторной работы был выбран язык программирования C++. Время измерялось с помощью функции clock() из библиотеки time.h [5].

3.2 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора.

Листинг 3.1 – Алгоритм полного перебора

```
1    pair<int, vector<int>> brute_force_alg(const
    vector<vector<int>> &matrix, const unsigned int &amount_node)
2    {
3        vector<int> permutation(amount_node);
4        for (unsigned int i = 0; i < amount_node; i++)
5            permutation[i] = i;
6
7        int best_dist = INT_MAX;
8        vector<int> best_route;
9
10       do
11       {
12           int cur_dist = 0;
13
14           for (unsigned int i = 0; i < amount_node - 1; ++i)
15               cur_dist += matrix[permutation[i]][permutation[i
    + 1]];
16
17           if (cur_dist < best_dist)
18           {
19               best_dist = cur_dist;
```

```
20                  best_route = permutation;
21              }
22          } while (next_permutation(permutation.begin(),
    permutation.end()));
23
24          return make_pair(best_dist, best_route);
25      }
```

В листинге 3.2 представлена реализация алгоритма вычисления среднего значения расстояний между всеми парами.

Листинг 3.2 – Алгоритм вычисления среднего значения расстояний между всеми парами

```
1     double calcQ(const vector<vector<int>>& matrix)
2     {
3         double sum = 0.0;
4         int count = 0;
5
6         for (const auto& row : matrix)
7             for (int dist : row)
8                 if (dist != 0)
9                     {
10                        sum += dist;
11                        count++;
12                    }
13        return sum / count;
14    }
```

В листинге 3.3 представлена реализация алгоритма выбора следующего города, куда муравей должен переместиться.

Листинг 3.3 – Алгоритм выбора следующего города

```
1   int chooseNextLoc(const vector<double> &P)
2   {
3       double posibility = static_cast<double>(rand()) /
    RAND_MAX;
4       double cur_posibility = 0.0;
5       int to = 0;
6
7       while (cur_posibility < posibility && to < P.size())
8       {
9           cur_posibility += P[to];
10          to++;
11      }
12
13      return to - 1;
14  }
```

В листинге 3.4 представлена реализация алгоритма вычисления пути, пройденного муравьем.

Листинг 3.4 – Алгоритм вычисления пути

```
1   double calcLen(const vector<vector<int>> &matrix, const
    vector<int> &ant_route)
2   {
3       double ant_dist = 0.0;
4
5       for (size_t i = 0; i < ant_route.size() - 1; ++i)
6       {
7           int city_from = ant_route[i];
8           int city_to = ant_route[i + 1];
9           ant_dist += matrix[city_from][city_to];
10      }
11      return ant_dist;
12  }
```

В листинге 3.5 представлена реализация муравьиного алгоритма.

Листинг 3.5 – Муравьиный алгоритм

```
 pair<double, vector<int>> fit(const vector<vector<int>>
&matrix, const int &ants)
 {
     srand(static_cast<unsigned int>(time(nullptr)));
     int cities = matrix.size();
     vector<vector<double>> tao(cities, vector<double>(cities,
1.0));

     vector<vector<int>> ant_routes(ants, vector<int>(cities,
0));
     vector<double> ant_dist(ants, 0.0);

     vector<int> best_route;
     double best_dist = numeric_limits<double>::infinity();

     Q = calcQ(matrix);

     for (int day = 0; day < days; ++day)
     {
         for (int ant = 0; ant < ants; ++ant)
         {
             ant_routes[ant][0] = rand() % cities;

             for (int i = 1; i < cities; ++i)
             {
                 int from_city = ant_routes[ant][i - 1];

                 vector<double> P(cities, 0.0);
                 double sum_prob = 0.0;

                 for (int j = 0; j < cities; ++j)
                 {
                     if (find(ant_routes[ant].begin(),
ant_routes[ant].begin() + i, j) == ant_routes[ant].begin() + i)
                     {
                         double pher = pow(tao[from_city][j],
a);
                         double dist = pow(1.0 /
matrix[from_city][j], b);
```

```
                P[j] = pher * dist;
                sum_prob += P[j];
            }
        }

        for (int j = 0; j < cities; ++j)
        P[j] /= sum_prob;

        int next_city = chooseNextLoc(P);
        ant_routes[ant][i] = next_city;
    }

    ant_dist[ant] = calcLen(matrix, ant_routes[ant]);

    if (ant_dist[ant] < best_dist)
    {
        best_dist = ant_dist[ant];
        best_route = ant_routes[ant];
    }
}
for (int ant = 0; ant < ants; ++ant)
{
    double delta_tao = Q / ant_dist[ant];
    for (int i = 0; i < cities - 1; ++i)
    {
        int city_from = ant_routes[ant][i];
        int city_to = ant_routes[ant][i + 1];
        tao[city_from][city_to] += delta_tao;

        if (tao[city_from][city_to] < MIN_PHER) {
            tao[city_from][city_to] = MIN_PHER;
        }
    }
}
for (int i = 0; i < cities; ++i)
    for (int j = 0; j < cities; ++j)
        tao[i][j] *= (1.0 - rho);
}
return make_pair(best_dist, best_route);
}
```

Вывод

Были представлены листинги всех реализаций алгоритмов — полного перебора и муравьиного. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

4.1 Интерфейс приложения

На рисунке 4.1 представлен интерфейс приложения.

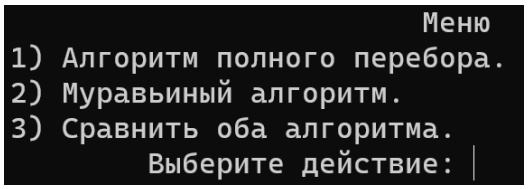

Рисунок 4.1 – Интерфейс приложения

4.2 Технические характеристики

Технические характеристики устройства:

- операционная система — Windows 11 Pro 64 – разрядная система [6];
- оперативная память — 16 Гбайт;
- процессор — 11th Gen Intel(R) Core(TM) i7-1165G7 с тактовой частотой 2.8 ГГц.

4.3 Время выполнения реализаций алгоритмов

На рисунке 4.2 представлено сравнение времени выполнения муравьиного алгоритма и алгоритма полного перебора. Время выполнения измерялось как среднее значение из десяти измерений.

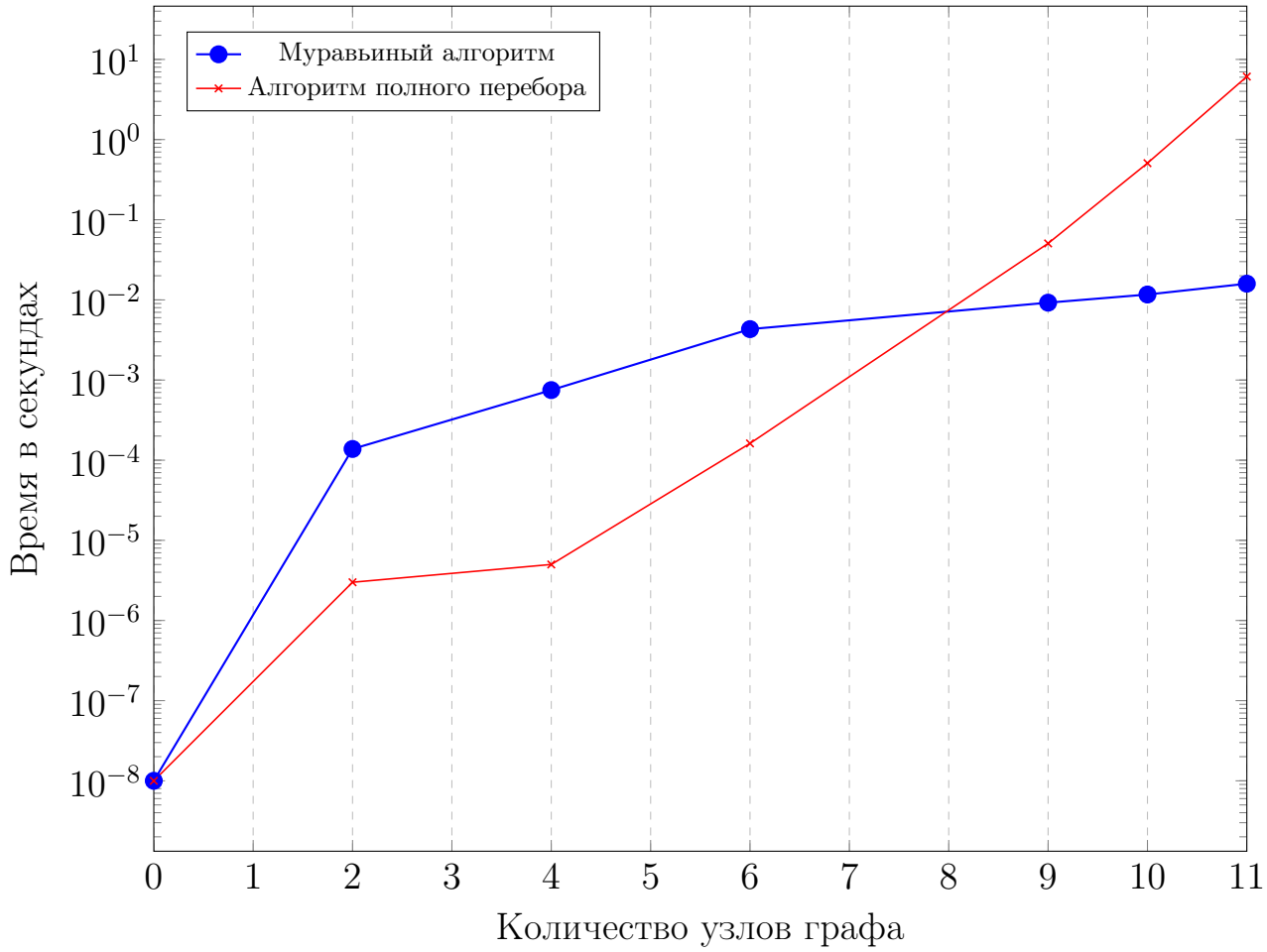

Рисунок 4.2 – Сравнение алгоритмов

Вывод

В результате исследования было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, при размере матрицы, меньшим 9, муравьиный алгоритм медленнее алгоритма полного перебора, а при размере матрицы, равном 9 и выше, муравьиный алгоритм быстрее алгоритма полного перебора. Следовательно, при размерах матриц больше 9 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует получения глобального оптимума при решении задачи.

ЗАКЛЮЧЕНИЕ

В результате исследования было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, при размере матрицы, меньшим 9, муравьиный алгоритм медленнее алгоритма полного перебора, а при размере матрицы, равном 9 и выше, муравьиный алгоритм быстрее алгоритма полного перебора. Следовательно, при размерах матриц больше 9 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует получения глобального оптимума при решении задачи.

В результате выполнения лабораторной работы были выполнены следующие задачи:

1) описана задача коммивояжера;
2) описаны методы решения задачи коммивояжера: метод полного перебора и метод на основе муравьиного алгоритма;
3) реализованы данные алгоритмы;
4) проведен сравнительный анализ по времени реализованного алгоритма;
5) подготовлен отчет о лабораторной работе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 А. Киселева А. Обзор методов решения задачи коммивояжера. Издательство «Научно-исследовательские публикации» (ООО «Вэлборн»), 2019.

2 О. Борознов В. Исследование решения задачи коммивояжера. — АГТУ, Вестник Астраханского государственного технического университета. Режим доступа: `https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera/viewer` (дата обращения: 08.02.2024).

3 С Дробот К., Н. Едемская Е. Решение задачи коммивояжера с помощью муравьиного алгоритма. 2020. С. — 344–348.

4 А. Коцюбинская С. Задача глобальной оптимизации. Муравьиный алгоритм. 2020. С. 537–540.

5 C library function clock() [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/c/chrono/clock_t` (дата обращения: 16.01.2024).

6 Windows 11 Pro [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/en-us/windows/business/windows-11-pro#windows11security` (дата обращения: 16.01.2024).