##

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 дисциплине «Анализ Алгоритмов»

Тема Задача коммивояжера

Студент Смирнов И.В.

Группа ИУ7-52Б

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

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача коммивояжера является важной и вместе с тем трудноразрешимой [1]. Данная задача возникает в обширном классе таких приложений, как, например, распознавание траекторий, образов, построение оптимальных схем движения и т. д.

Задача коммивояжера представляет собой задачу отыскания кратчайшего гамильтонова пути в полном конечном графе с N вершинами. Все известные методы нахождения точного решения включают в себя поиск пространства решений, которое увеличивается экспоненциально в зависимости от N. В данной лабораторной работе формулировка задачи ставится следующим образом: «Найти кратчайший незамкнутый маршрут в ориентированном графе, представляющий карту городов России XVI века. При этом раз в 60 суток наступает смена летнего сезона на зимний или наоборот. Зимой можно ходить по рекам в обе стороны за равную цену, летом по течению в 2 раза быстрее, против — в 4 раза медленнее. Необходимо реализовать метод полного перебора и метод на основе муравьиного алгоритма с элитными муравьями.»

Цель лабораторной работы — выполнить сравнительный анализ метода полного перебора с методом на базе муравьиного алгоритма. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать схемой алгоритма и реализовать метод полного перебора для решения задачи коммивояжера;

— описать схемой алгоритма и реализовать метод на основе муравьиного алгоритма для решения задачи коммивояжера;

— выполнить оценку трудоемкости по разработанным схемам алгоритмов;

— указать преимущества и недостатки реализованных методов;

— выполнить сравнительный анализ двух методов решения задачи коммивояжера;

— выполнить параметризацию метода на основе муравьиного алгоритма по трем его параметрам и дать рекомендации о комбинациях значений параметров для работы алгоритма.

1 Аналитическая часть

В данном разделе будут рассмотрены два метода решения задачи коммивояжера, основанные на полном переборе и муравьином алгоритме.

1.1 Алгоритм полного перебора

Алгоритм полного перебора (АПП) осуществляет поиск в пространстве $N!$ решений посредством перебора всех вариантов маршрута и выбирает наикратчайший из них. Результатом работы алгоритма является точное решение. Недостатком АПП является его временная сложность — пространство поиска растёт факториально.

1.2 Муравьиный алгоритм

В основе муравьиного алгоритма лежит идея моделирования поведения колонии муравьев. Каждый муравей определяет свой маршрут на основе оставленных другими муравьями феромонов, а также сам оставляет оставляет феромоны, чтобы последующие муравьи ориентировались по ним. В результате при прохождении каждым муравьем своего маршрута наибольшее число феромонов остается на самом оптимальном пути. Временная сложность алгоритма была оценена как $683 - (42,467N) + (1,0696N^2)$ [2]. Однако главный недостаток алгоритма заключается в том, что, по сравнению с алгоритмом полного перебора, он даёт приближенное решение задачи, а не точное.

ВЫВОД

В данном разделе были рассмотрены два основных алгоритма, используемые в методах решения задачи коммивояжера.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов, а также оценены трудоемкости каждого из алгоритмов.

2.1 Представление алгоритмов

На вход алгоритмов подается матрица стоимостей G размера $N \times N$; на выходе - минимальная стоимость маршрута $minCost$, а также сам маршрут, представленный последовательностью посещенных городов $bestRoute$.

Для муравьиного алгоритма на вход также требуются следующие параметры: коэффициент влияния феромона $alpha$, коэффициент испарения феромона rho, максимальное количество итераций $tmax$, количество обычных муравьев $numA$, количество элитных муравьев $numEA$.

На рисунках 2.1 — 2.6 приведены схемы двух алгоритмов, используемых в методах решения задачи коммивояжера: полного перебора, муравьиного.

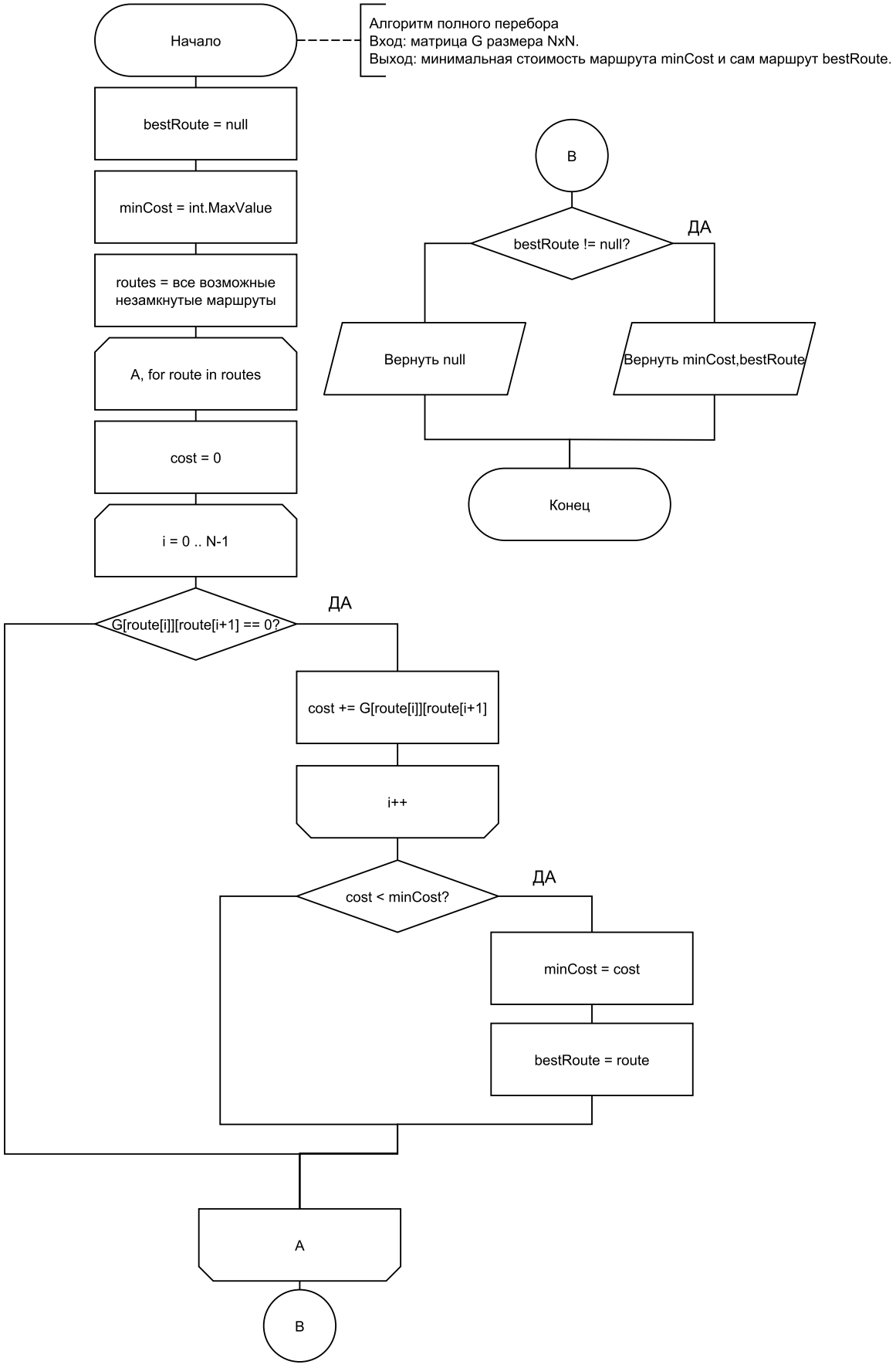

Рисунок 2.1 – Схема алгоритма полного перебора

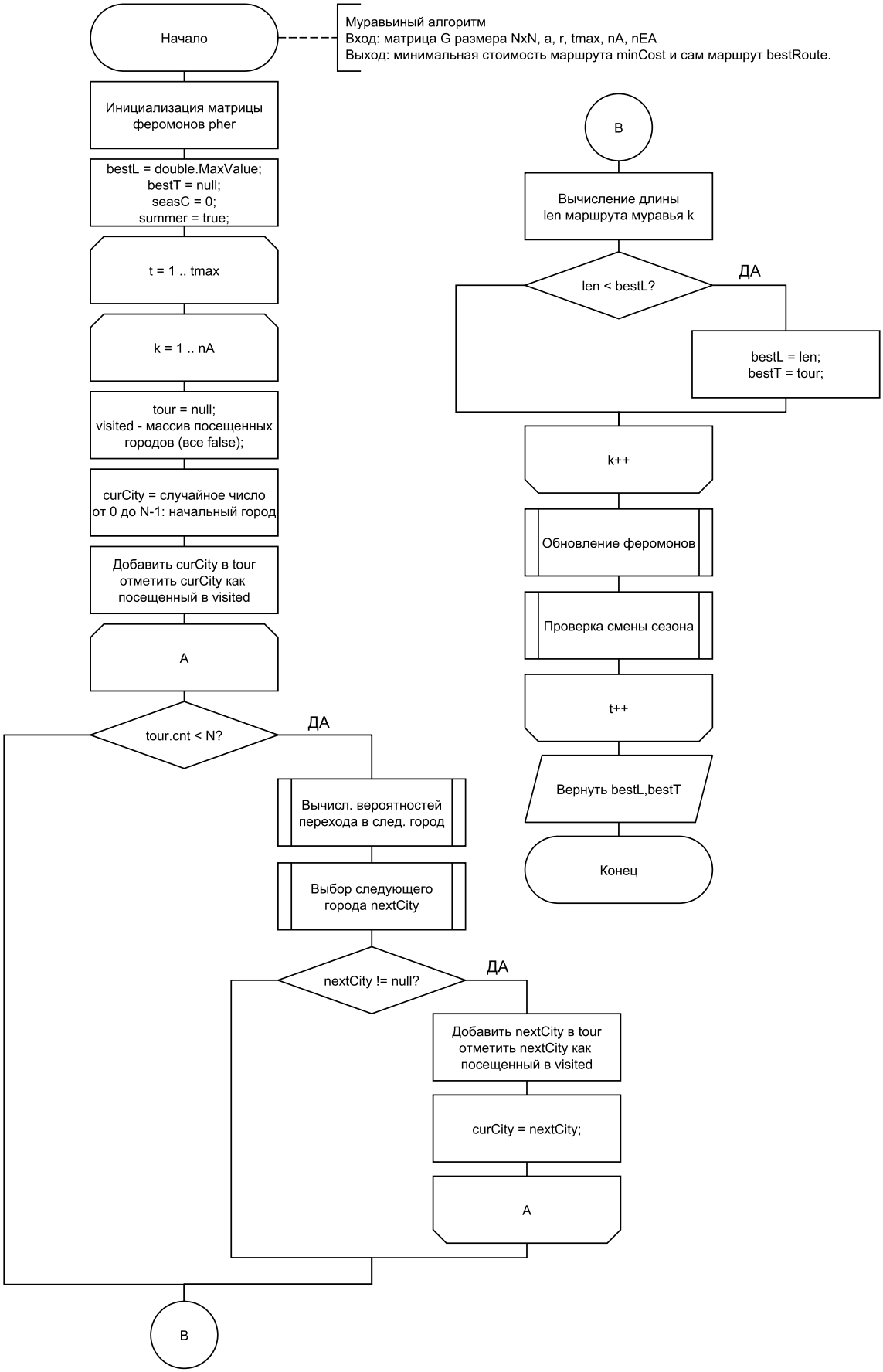

Рисунок 2.2 – Схема муравьиного алгоритма

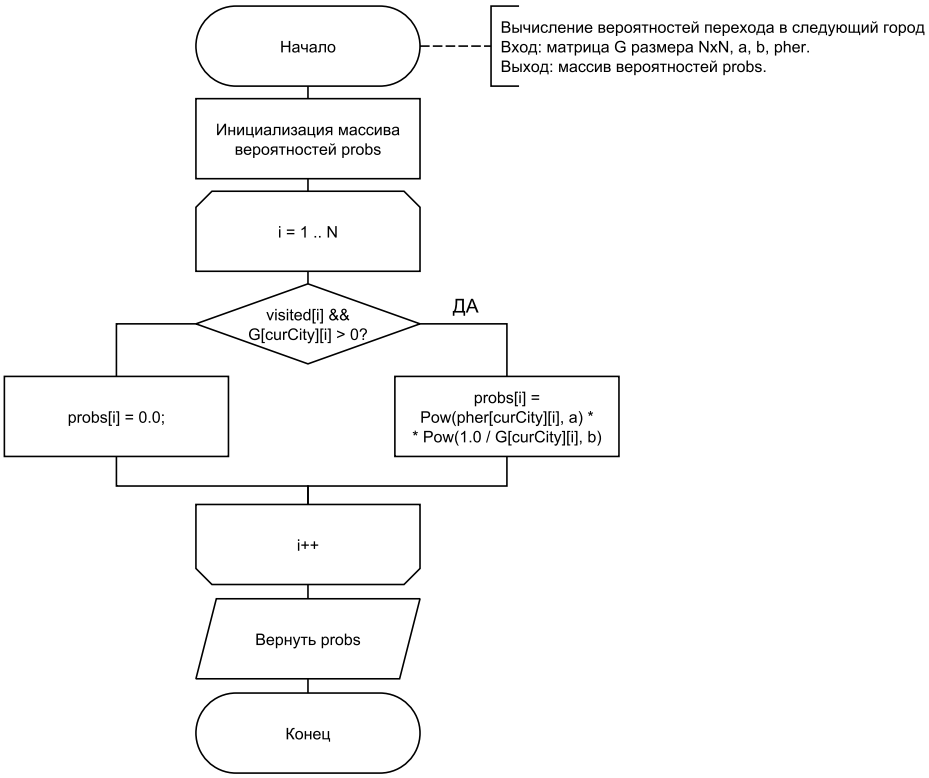

Рисунок 2.3 – Схема вычисления вероятностей

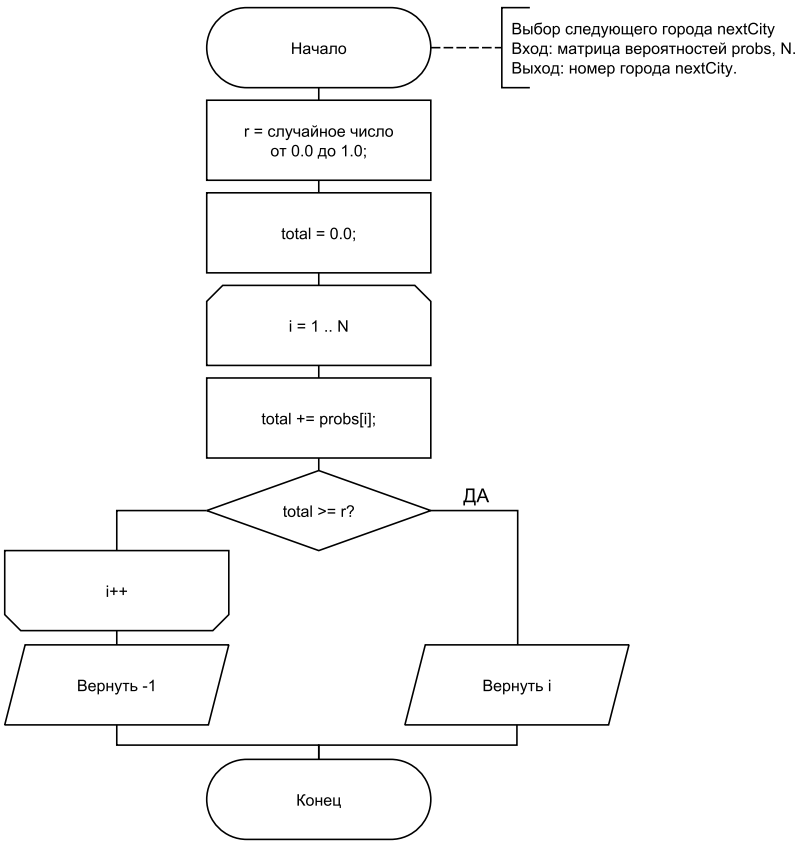

Рисунок 2.4 – Схема вычисления нового города

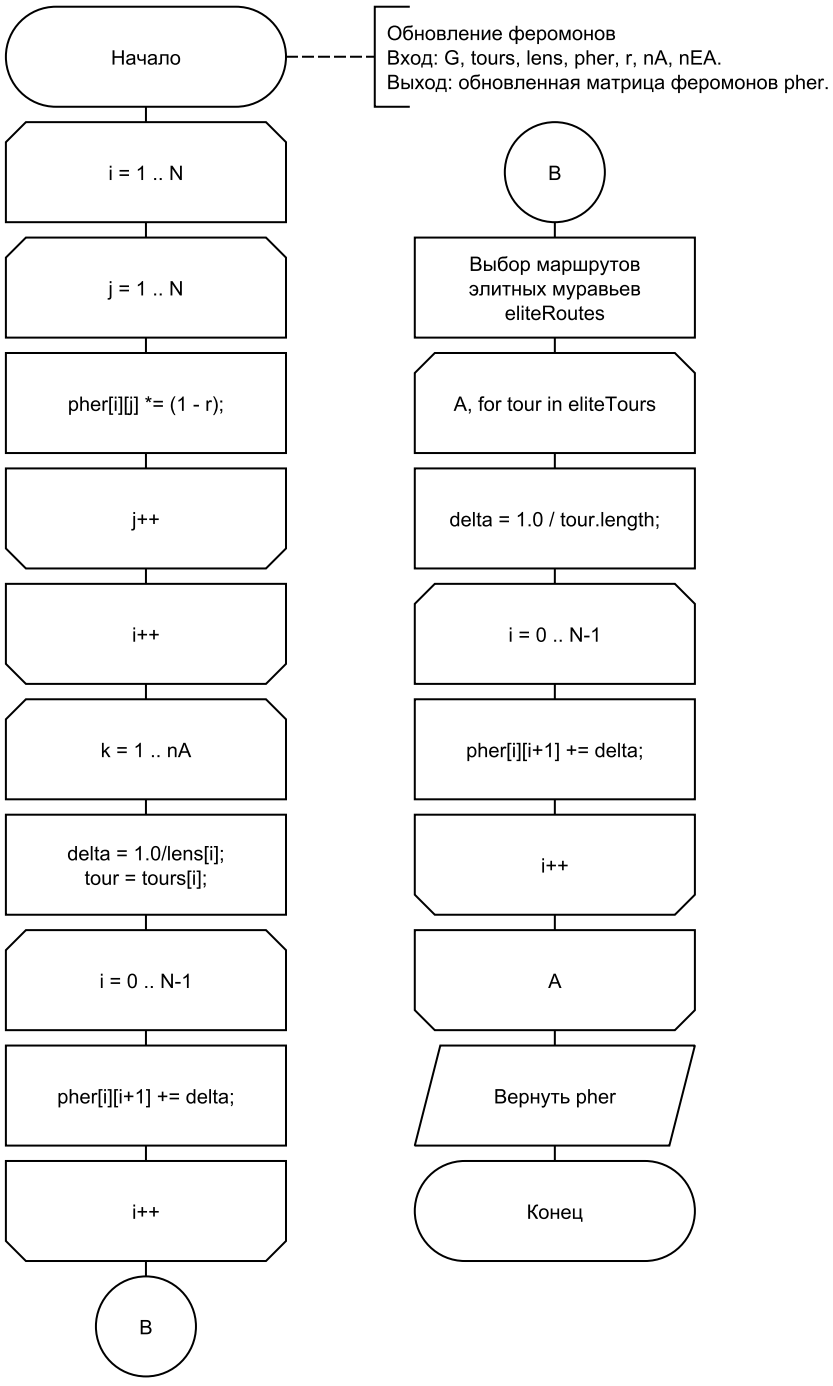

Рисунок 2.5 – Схема обновления феромонов

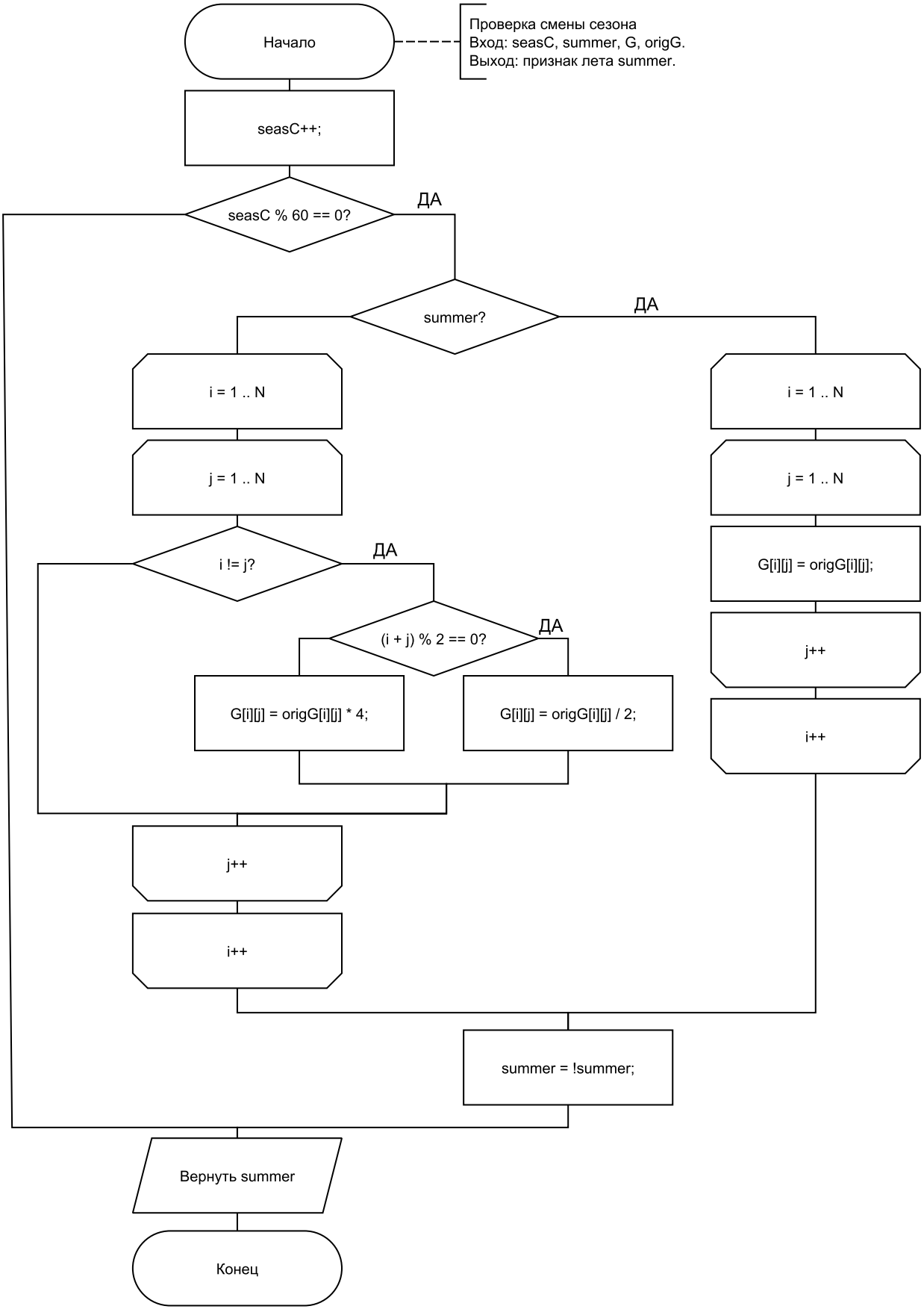

Рисунок 2.6 – Схема смены сезона

2.2 Модель вычислений

Для последующего вычисления трудоемкости была введена модель вычислений:

1. операции из списка (2.2) имеют трудоемкость 1;

$$+, -, ==, !=, <, >, <=, >=, [], ++, --, \&\&, ||; \tag{2.1}$$

2. операции из списка (2.2) имеют трудоемкость 2;

$$*, /, \%, pow, \tag{2.2}$$

где pow - операция возведения в степень.

3. трудоемкость условного оператора `if условие then A else B` рассчитывается как (2.3);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4. трудоемкость цикла рассчитывается как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

5. трудоемкость вызова функции/возврата результата равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости представленных ранее алгоритма полного перебора и муравьиного алгоритма. Трудоемкость начальной инициализации переменных $minCost$ и $bestRoute$, поскольку данное действие есть во всех алгоритмах и не является самым трудоемким.

Введены обозначения:

— N — кол-во строк и столбцов матрицы стоимостей;

— t_{max} — кол-во итераций;

— K — кол-во муравьев;

— E — кол-во элитных муравьев.

2.3.1 Алгоритм полного перебора

Трудоемкость алгоритма полного перебора для решения задачи коммивояжера состоит из:

— генерации всех перестановок N городов, трудоемкость которой:

$$f = N! \cdot N; \tag{2.5}$$

— обработки каждой перестановки, трудоемкость которой:

$$f = f_{\text{ПВМ}} + f_{\text{ВСМ}} + f_{\text{обновления}}; \tag{2.6}$$

— проверка валидности маршрута (ПВМ) требует $N - 1$ сравнения, а значит трудоемкость этого этапа равняется:

$$f_{\text{ПВМ}} = N - 1; \tag{2.7}$$

— вычисление стоимости маршрута (ВСМ) подразумевает суммирование стоимостей переходов между городами, трудоемкость считается как:

$$f_{\text{ВСМ}} = 3 \cdot (N - 1); \tag{2.8}$$

— при обновлении значений маршрута и длины проводятся 2 сравнения и 2 присваивания, трудоемкость которых равна:

$$f_{\text{обновления}} = 4; \tag{2.9}$$

В итоге трудоемкость алгоритма полного перебора равна:

$$f_{bruteforce} = N! \cdot N + N - 1 + 3 \cdot (N - 1) + 4 = N! \cdot N + 4N \tag{2.10}$$

2.3.2 Муравьиный алгоритм

Трудоемкость муравьиного алгоритма состоит из:

— инициализации параметров, трудоемкость которой пренебрежимо мала по сравнению с другими этапами:

$$f = 1; \tag{2.11}$$

— инициализация матрицы феромонов pher:

$$f_{init} = N^2; \tag{2.12}$$

— основного цикла по итерациям:

$$f_{tmax} = 2 + t_{max} \cdot (f_t); \tag{2.13}$$

— цикла по муравьям:

$$f_t = 2 + K \cdot (f_k); \tag{2.14}$$

— начальной инициализации параметров муравья (tour, visited, curCity):

$$f_{initK} = 2 + N + 3 = N + 5; \tag{2.15}$$

— построения маршрута, количество шагов которого равно N, а для каждого непосещенного города i вычисляется вероятность перехода, значит общая трудоемкость данного этапа равна:

$$f_{buildR} = 3+N+2+N\cdot(2+N+2+N\cdot(2+7+14+2)+4) = 5+9N+26N^2; \tag{2.16}$$

— выбора нового города:

$$f_{chooseR} = 2 + N \cdot (N + N) = 2 + 2N^2; \tag{2.17}$$

— вычисления длины маршрута:

$$f_{len} = 2 + (N - 1) = N + 1; \tag{2.18}$$

Значит трудоемкость цикла по муравьям равна:

$$f_t = 2+K\cdot(N+5+5+9N+26N^2+2+2N^2+N+1) = 2+K(13+11N+28N^2); \tag{2.19}$$

— обновления феромонов:

$$f_{pher} \approx N^2; \tag{2.20}$$

— нанесение феромонов муравьями и элитными муравьями:

$$f_{addpher} \approx K \cdot N + E \cdot N; \tag{2.21}$$

— смены сезона:

$$f_{season} = 5 + 2 + N \cdot (2 + N \cdot (5 + 2 + 5)) = 7 + 2N + 12N^2; \tag{2.22}$$

Итого, результирующая трудоемкость муравьиного алгоритма равна (2.23)

$$f_{final} = 1 + f_{init} + 2 + t_{max} \cdot (f_t + f_{pher} + f_{addpher} + f_{season}); \tag{2.23}$$

$$f_{final} \approx 28 t_{max} \cdot N^2 K \approx 28 N^4. \tag{2.24}$$

ВЫВОД

В данном разделе были представлены схемы алгоритмов для решения задачи коммивояжера, а также приведены трудоемкости каждого из алгоритмов.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода.

3.1 Требования к программному обеспечению

Входные данные: квадратная матрица стоимостей и дополнительные параметры к муравьиному алгоритму;

Выходные данные: минимальная стоимость маршрута и сам маршрут (массив посещенных городов).

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования $C\#$ [3]. Выбор обсуловлен наличием $LINQ$—выражений [4], упрощающих работу с коллекциями и данными, а именно для нахождения всех возможных замкнутых маршрутов и для выбора элитных муравьев из всех муравьев. Время было замерено с помощью методов класса *Stopwatch* [5].

3.3 Реализация алгоритмов

В листингах 3.1 — 3.2 представлены реализации алгоритмов полного перебора и муравьиного.

Листинг 3.1 – Алгоритм полного перебора

```
static void RunBruteForce()
{
    Console.WriteLine("\nBruteForce algo...");
    List<int> bestRoute = null;
    int minCost = int.MaxValue;
    foreach (var permutation in GetPermutations(Enumerable.Range(0, n),
       n))
    {
        int cost = 0;
        bool validRoute = true;

        for (int i = 0; i < permutation.Count() - 1; i++)
        {
            int from = permutation.ElementAt(i);
            int to = permutation.ElementAt(i + 1);

            if (graph[from, to] == 0)
            {
                validRoute = false;
                break;
            }
            cost += graph[from, to];
        }
        if (validRoute && cost < minCost)
        {
            minCost = cost;
            bestRoute = permutation.ToList();
        }
    }
    if (bestRoute != null)
    {
        Console.WriteLine("Min cost: " + minCost);
        Console.WriteLine("Best route: " +
            string.Join(" -> ", bestRoute.Select(i => cities[i])));
    }
    else
        Console.WriteLine("No route found.");
}
```

Листинг 3.2 – Муравьиный алгоритм

```
public List<int> Run()
{
    for (int t = 0; t < tMax; t++)
    {
        List<List<int>> tours = new List<List<int>>();
        List<double> lengths = new List<double>();

        for (int k = 0; k < numAnts; k++)
        {
            List<int> tour = ConstructSolution();
            double length = CalculateTourLength(tour);
            tours.Add(tour);
            lengths.Add(length);

            if (length < BestLength)
            {
                BestLength = length;
                BestTour = new List<int>(tour);
            }
        }

        UpdatePheromones(tours, lengths);

        seasonCounter++;
        if (seasonCounter % 60 == 0)
        {
            UpdateSeason();
            isSummer = !isSummer;
        }
    }
    return BestTour;
}

List<int> ConstructSolution()
{
    List<int> tour = new List<int>();
    bool[] visited = new bool[n];
    int currentCity = rand.Next(n);
    tour.Add(currentCity);
    visited[currentCity] = true;
```

```
    while (tour.Count < n)
    {
        int nextCity = SelectNextCity(currentCity, visited);
        if (nextCity == -1)
            break;
        tour.Add(nextCity);
        visited[nextCity] = true;
        currentCity = nextCity;
    }
    return tour;
}

int SelectNextCity(int currentCity, bool[] visited)
{
    double[] probabilities = new double[n];
    double sum = 0.0;
    for (int i = 0; i < n; i++)
    {
        if (!visited[i] && graph[currentCity, i] > 0)
        {
            probabilities[i] = Math.Pow(pheromone[currentCity, i],
               alpha) *
                               Math.Pow(1.0 / graph[currentCity, i],
                                  beta);
            sum += probabilities[i];
        }
        else
            probabilities[i] = 0.0;
    }
    if (sum == 0.0)
        return -1;
    double r = rand.NextDouble() * sum;
    double total = 0.0;
    for (int i = 0; i < n; i++)
    {
        total += probabilities[i];
        if (total >= r)
            return i;
    }

    return -1;
```

```
}

void UpdatePheromones(List<List<int>> tours, List<double> lengths)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            pheromone[i, j] *= (1.0 - rho);

    for (int k = 0; k < tours.Count; k++)
    {
        double delta = 1.0 / lengths[k];
        List<int> tour = tours[k];

        for (int i = 0; i < tour.Count - 1; i++)
        {
            int from = tour[i];
            int to = tour[i + 1];
            pheromone[from, to] += delta;
        }
    }

    var eliteAnts = lengths.Select((length, index) => new { length,
       index })
                           .OrderBy(l => l.length)
                           .Take(numEliteAnts)
                           .Select(l => tours[l.index]);
    foreach (var tour in eliteAnts)
    {
        double delta = 1.0 / CalculateTourLength(tour);
        for (int i = 0; i < tour.Count - 1; i++)
        {
            int from = tour[i];
            int to = tour[i + 1];
            pheromone[from, to] += delta;
        }
    }
}

double CalculateTourLength(List<int> tour)
{
    double length = 0.0;
```

```
    for (int i = 0; i < tour.Count - 1; i++)
        length += graph[tour[i], tour[i + 1]];
    return length;
}

void UpdateSeason()
{
    if (isSummer)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                graph[i, j] = originalGraph[i, j];
    }
    else
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i != j)
                    if ((i + j) % 2 == 0)
                        graph[i, j] = originalGraph[i, j] / 2;
                    else
                        graph[i, j] = originalGraph[i, j] * 4;
    }
}
```

ВЫВОД

В данном разделе были рассмотрены требования к программному обеспечению, используемые средства реализации, а также приведены листинги кода двух алгоритмов решения задачи коммивояжера.

4 Исследовательская часть

В данном разделе будут выполнены сравнительный анализ двух методов решения задачи коммивояжера и параметризация метода на основе муравьиного алгоритма по трем его параметрам.

4.1 Технические характеристики

Характеристики используемого оборудования:

— операционная система — Windows 11 Home [6];

— память — 16 Гб;

— процессор — Intel(R) Core(TM) i5-10300H CPU @ 2.50ГГц [7].

4.2 Время выполнения алгоритмов

Результаты замеров времени работы двух алгоритмов решения задачи коммивояжера приведены в таблице 4.1. Время работы алгоритмов замерялось на указанном ранее оборудовании. Замеры времени проводились на матрицах одинаковой длины от 1 до 9 и усреднялись для каждого набора одинаковых серий замеров. Каждое значение получено путем взятия среднего из 40 измерений. Зависимости времени выполнения задачи от размера матрицы стоимостей для двух алгоритмов представлены на рисунке 4.1.

Таблица 4.1 – Время работы алгоритмов (в мс)

Размер матрицы	Полный перебор	Муравьиный
1	0,115	0,863
2	0,159	0,943
3	0,081	0,948
4	0,071	1,472
5	0,952	2,460
6	8,413	3,565
7	85,019	4,527
8	862,511	5,422
9	11282,25	6,619

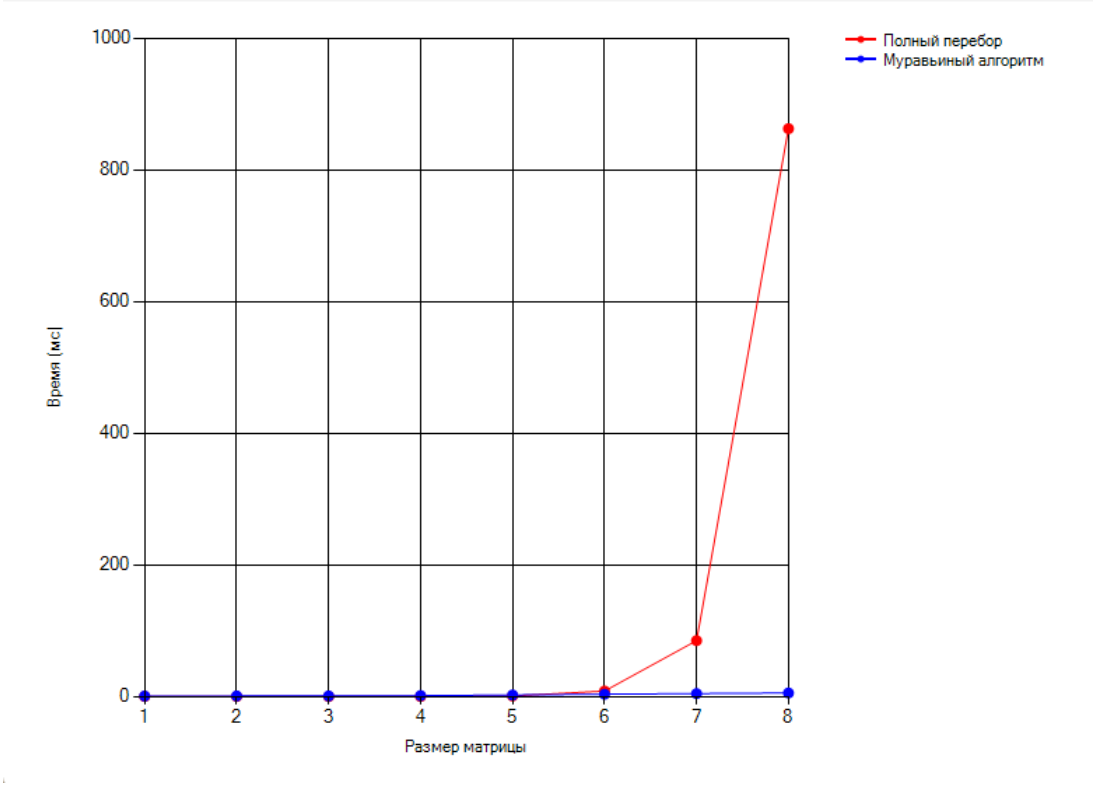

Рисунок 4.1 – Сравнение алгоритмов по времени

4.3 Параметризация

Целью проведения параметризации является определение таких комбинаций параметров, при которых муравьиный алгоритм дает наилучшие результаты.

В результате параметризации будет получена таблица со следующими столбцами:

— a — коэффициент влияния феромона;

— r — коэффициент испарения феромона;

— t — количество дней/итераций;

— mx, md, av — максимальное, медианное и среднее арифметическое значение отклонения стоимости полученного марщрута от эталонного для каждого из трех графов.

4.4 Класс данных

Класс данных представляет собой набор из трех ориентированных графов, представленных в виде матриц стоимостей для 9 вершин:

$$\mathbf{G_1} = \begin{pmatrix} 0 & 10 & 15 & 20 & 25 & 30 & 35 & 40 & 45 \\ 12 & 0 & 37 & 27 & 19 & 26 & 33 & 18 & 17 \\ 14 & 35 & 0 & 32 & 22 & 23 & 28 & 24 & 26 \\ 16 & 29 & 30 & 0 & 33 & 17 & 27 & 28 & 31 \\ 18 & 21 & 24 & 35 & 0 & 37 & 32 & 20 & 23 \\ 20 & 31 & 29 & 19 & 31 & 0 & 22 & 26 & 29 \\ 22 & 33 & 34 & 25 & 28 & 24 & 0 & 22 & 25 \\ 24 & 23 & 26 & 27 & 22 & 29 & 26 & 0 & 13 \\ 26 & 25 & 28 & 29 & 25 & 31 & 29 & 15 & 0 \end{pmatrix} \tag{4.1}$$

$$\mathbf{G_2} = \begin{pmatrix} 0 & 12 & 18 & 24 & 30 & 36 & 42 & 48 & 54 \\ 14 & 0 & 23 & 29 & 35 & 41 & 47 & 53 & 59 \\ 16 & 25 & 0 & 38 & 44 & 50 & 56 & 62 & 68 \\ 18 & 27 & 40 & 0 & 53 & 59 & 65 & 71 & 77 \\ 20 & 29 & 42 & 55 & 0 & 68 & 74 & 80 & 86 \\ 22 & 31 & 44 & 57 & 70 & 0 & 83 & 89 & 95 \\ 24 & 33 & 46 & 59 & 72 & 85 & 0 & 98 & 104 \\ 26 & 35 & 48 & 61 & 74 & 87 & 96 & 0 & 110 \\ 28 & 37 & 50 & 63 & 76 & 89 & 102 & 115 & 0 \end{pmatrix} \tag{4.2}$$

$$\mathbf{G_3} = \begin{pmatrix} 0 & 14 & 20 & 18 & 24 & 32 & 40 & 26 & 22 \\ 16 & 0 & 24 & 30 & 36 & 40 & 44 & 28 & 20 \\ 18 & 26 & 0 & 28 & 32 & 36 & 40 & 30 & 22 \\ 20 & 28 & 30 & 0 & 38 & 42 & 46 & 32 & 24 \\ 22 & 30 & 34 & 40 & 0 & 48 & 52 & 34 & 26 \\ 24 & 32 & 36 & 42 & 48 & 0 & 56 & 36 & 28 \\ 26 & 34 & 38 & 44 & 50 & 56 & 0 & 38 & 30 \\ 28 & 36 & 40 & 46 & 52 & 58 & 64 & 0 & 18 \\ 30 & 38 & 42 & 48 & 54 & 60 & 66 & 18 & 0 \end{pmatrix} \tag{4.3}$$

Вырезка наилучших релультатов представлена в таблице 4.2.

Таблица 4.2 – Результаты параметризации (вырезка)

a	r	t	mxG1	avG1	mdG1	mxG2	avG2	mdG2	mxG3	avG3	mdG3
0,5	0,3	50	3,00	0,90	0,00	6,00	5,10	6,00	4,00	2,80	2,00
1	0,3	50	0,00	0,00	0,00	9,00	6,90	6,00	2,00	1,60	2,00
1,5	0,3	50	0,00	0,00	0,00	9,00	7,50	7,50	4,00	2,60	2,00
2	0,1	100	1,00	0,10	0,00	6,00	4,20	4,50	2,00	1,00	1,00
2,5	0,1	100	0,00	0,00	0,00	9,00	6,60	6,00	2,00	1,80	2,00
1	0,3	100	0,00	0,00	0,00	9,00	7,50	9,00	2,00	1,20	2,00
1,5	0,3	100	0,00	0,00	0,00	12,00	6,90	9,00	6,00	3,20	2,00
2	0,3	100	3,00	0,30	0,00	15,00	8,70	9,00	6,00	3,00	2,00
2,5	0,3	250	3,00	-0,90	0,00	15,00	-21,00	6,00	6,00	-1,60	2,00

4.5 Вывод

В результате исследования было получено, что при $N < 7$ муравьиный алгоритм и алгоритм полного перебора выполянют задачу за примерно одинаковое время, а при $N \geq 7$ муравьиный алгоритм работает около в раз быстрее, чем алгоритм полного перебора.

Также было выявлено, что при $a = 2; r = 0,1; t = 100;$ муравьиный алгоритм дает наилучшие результаты.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: проведен сравнительный анализ двух методов решения задачи коммивояжера на основе алгоритма полного перебора и муравьиного алгоритма.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

— описана схема алгоритма и реализован метод полного перебора для решения задачи коммивояжера;

— описана схема алгоритма и реализован метод на основе муравьиного алгоритма для решения задачи коммивояжера;

— выполнена оценка трудоемкости по разработанным схемам алгоритмов;

— указаны преимущества и недостатки реализованных методов;

— выполнен сравнительный анализ двух методов решения задачи коммивояжера;

— выполнена параметризация метода на основе муравьиного алгоритма по трем его параметрам и дать рекомендации о комбинациях значений параметров для работы алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Борознов В. О. Исследование решения задачи коммивояжера //Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. — 2009г.

2. Штобва С. Д. Муравьиные алгоритмы [Электронный ресурс]. URL: `https://masters.donntu.ru/2014/fknt/zuy/library/article5.pdf` (дата обращения: 01.12.2024).

3. Albahari B. A comparative overview of C [Электронный ресурс]. URL: `http://genamics.com/developer/csharp_comparative` (дата обращения: 02.12.2024).

4. Rattz J. Pro LINQ: Language Integrated Query in C# 2008. — Apress, 2008г.

5. C# Stopwatch (How It Works For Developers) [Электронный ресурс]. URL: `https://ironpdf.com/blog/net-help/csharp-stopwatch-guide/` (дата обращения: 02.12.2024).

6. Windows 11 Home [Электронный ресурс]. URL: `https://www.officepakke.dk/products/windows-11-home` (дата обращения: 02.12.2024).

7. Intel® Core™ i5-10300H Processor [Электронный ресурс]. URL: `https://ark.intel.com/content/www/us/en/ark/products/201839/intel-core-i5-10300h-processor-8m-cache-up-to-4-50-ghz.html` (дата обращения: 02.12.2024).

ПРИЛОЖЕНИЕ А

Результаты параметризации