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

Группа ИУ7-51Б

Тема работы **Задача коммивояжёра**

Студент Баранов Николай Алексеевич

Преподаватель Волкова Лилия Леонидовна

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы — подобрать оптимальные параметры для метода решения задачи коммивояжёра, основанном на муравьином алгоритме.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать постановку задачи коммивояжёра, метод решения полным перебором и метод на основе муравьиного алгоритма;
2) написать программу, реализующую оба алгоритма;
3) провести параметризацию муравьиного алгоритма по 3 параметрам.

1 Аналитическая часть

1.1 Модель вычислений

Модель вычислений описывает правила для задания оценки ресурсной эффективности алгоритмов.

Трудоёмкость следующих операций принимается за 1: =, +, −, + =, − =, ==, ! =, <, <=, >, >=, [], <<, >>, <<=, >>=, &&, ||, &, |, ˆ, & =, | =.

Трудоёмкость следующих операций принимается за 2: ∗, /, %, ∗ =, / =, % =.

Трудоёмкость функции возведения в степень $pow(a, b)$ принимается за 3.

Трудоёмкость цикла с известным количеством повторений рассчитывается следующим образом: пусть имеется цикл for (init; cond; inc) { body }, где init — блок инициализации переменных, cond — блок проверки условия, inc — блок изменения переменных, body — тело цикла. Пусть цикл выполнился N раз. Тогда трудоёмкость цикла f_{for} рассчитывается по формуле (1.1):

$$f_{for} = f_{init} + f_{cond} + N \cdot (f_{cond} + f_{inc} + f_{body}) \tag{1.1}$$

где f_{init} — трудоёмкость блока инициализации, f_{cond} — трудоёмкость блока проверки условия, f_{inc} — трудоёмкость блока изменения переменных, f_{body} — трудоёмкость тела цикла.

Трудоёмкость цикла с предусловием рассчитывается следующим образом: пусть имеется цикл while (cond) { body }, где cond — блок проверки условия, body — тело цикла. Пусть цикл выполнился N раз. Тогда трудоёмкость цикла f_{while} рассчитывается по формуле (1.2).

$$f_{while} = f_{cond} + N \cdot (f_{cond} + f_{body}) \tag{1.2}$$

где f_{cond} — трудоёмкость блока проверки условия, f_{body} — трудоёмкость тела цикла.

Трудоёмкость условного оператора рассчитывается следующим образом: пусть имеется условный оператор if (cond) then { body1 } else { body2 }, где cond — блок проверки условия, body1 — тело при выполнении условия, body2

— тело при невыполнении условия. Тогда трудоёмкость условного оператора f_{if} рассчитывается по формуле (1.3):

$$f_{if} = f_{cond} + \begin{cases} f_{body1} & \text{; условие cond выполнилось} \\ f_{body2} & \text{; иначе} \end{cases}, \tag{1.3}$$

где f_{cond} — трудоёмкость блока проверки условия, f_{body1} — трудоёмкость тела при выполнении условия, f_{body2} — трудоёмкость тела при невыполнении условия.

Операцией выделения памяти пренебрегаем.

1.2 Постановка задачи

Задача коммивояжёра состоит в поиске кратчайшего пути через все города на карте, в который каждый город входит ровно 1 раз [1]. Согласно варианту, необходимо искать гамильтонов цикл, поэтому путь должен быть замкнутым [2]. Также на рёбрах расположены затраты по времени перехода между вершинами в днях (измерение дней в вершинах не совпадает с измерением дней в муравьином алгоритме), при этом предел по времени — 80 дней.

1.3 Метод решения полным перебором

В данном методе рассматриваются все возможные варианты пути [1]. Преимуществом данного метода является гарантированное нахождение оптимального решения (если решение существует). Недостатком является временная сложность $O(n!)$.

1.4 Метод решения на основе муравьиного алгоритма

Муравьиные алгоритмы основаны на имитации природных механизмов самоорганизации муравьёв [3]. Метод заключается в поиске маршрута колонией муравьёв в течении t дней. Каждый день делится на 4 фазы — утро, день, вечер и ночь.

Утром муравьи покидают колонию и размещаются по 1 в каждом городе. В фазу «день» каждый из муравьёв пытается решить задачу. У каждого

муравья есть 3 чувства, исходя из которых он выбирает маршрут:

— память — муравей не заходит в один и тот же город дважды;

— зрение — находясь в городе, муравей может видеть все другие города, в которые можно перейти из данного города;

— обоняние — муравей чувствует следы феромона на рёбрах, соединяющих города.

Вероятность перехода P муравья k из города i в город j в день t рассчитывается по формуле (1.4):

$$P_{ij,k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_{ik}} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} & ; j \in J_{ik} \\ 0 & ; \text{иначе} \end{cases} \tag{1.4}$$

где $\tau_{ij}(t)$ — концентрация феромона на пути из города i в город j в день t, η_{ij} — видимость города j из города i, J_{ik} — множество городов, в которые можно попасть из города i и которые не были посещены муравьём k, $\alpha \in [0, 1]$ — коэффициент стадности, $\beta \in [0, 1]$ — коэффициент жадности, причём $\alpha + \beta = 1$. Видимость η_{ij} рассчитывается по формуле 1.5:

$$\eta_{ij} = \frac{1}{D_{ij}} \tag{1.5}$$

где D_{ij} — расстояние из города i в город j.

Вечером муравьи возвращаются в муравейник. Ночью для каждого пути из города i в город j обновляется концентрация феромона τ_{ij} по формуле (1.6):

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1 - \rho) + \triangle\tau_{ij}(t) \tag{1.6}$$

где $\triangle\tau_{ij}(t)$ — изменение концентрации феромона на пути из города i в город j в день t. $\triangle\tau_{ij}(t)$ рассчитывается по формуле (1.7):

$$\triangle\tau_{ij}(t) = \sum_{k=1}^{m} \triangle\tau_{ij,k}(t) \tag{1.7}$$

где $\delta\tau_{ij,k}(t)$ — концентрация феромона, которую оставил муравей k на пути из города i в город j в день t, m — число муравьёв. $\triangle\tau_{ij,k}(t)$ рассчитывается по формуле (1.8):

$$\triangle\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)} & ;(i,j) \in T_k(t) \\ 0 & ; \text{иначе} \end{cases} \tag{1.8}$$

где Q — количество феромона у муравья, значение которого выбирается одного порядка со значением оптимального маршрута, $L_k(t)$ — длина маршрута, пройденного муравьём k в день t, $T_k(t)$ — маршрут, пройденный муравьём k в день t. Кличество феромона Q можно рассчитать как отношение суммы всех дуг и удвоенного числа городов. Необходимо гарантировать необнуление феромона на ребре.

Преимуществом данного метода является временная сложность. Недостатком является отсутствие гарантии нахождения оптимального решения.

Выводы

В данном разделе была описана постановка задачи коммивояжёра и методы её решения: метод полного перебора и метод, основанный на муравьином алгоритме.

2 Конструкторская часть

2.1 Алгоритм, использующий полный перебор

На рисунке 2.1 представлена схема алгоритма, использующего полный перебор. На вход он получает количество городов n и матрицу смежности matrix размером n на n. В случае отсутствия ребра между городами в матрицу смежности в соответствующую ячейку вместо значения времени перехода заносится -1.

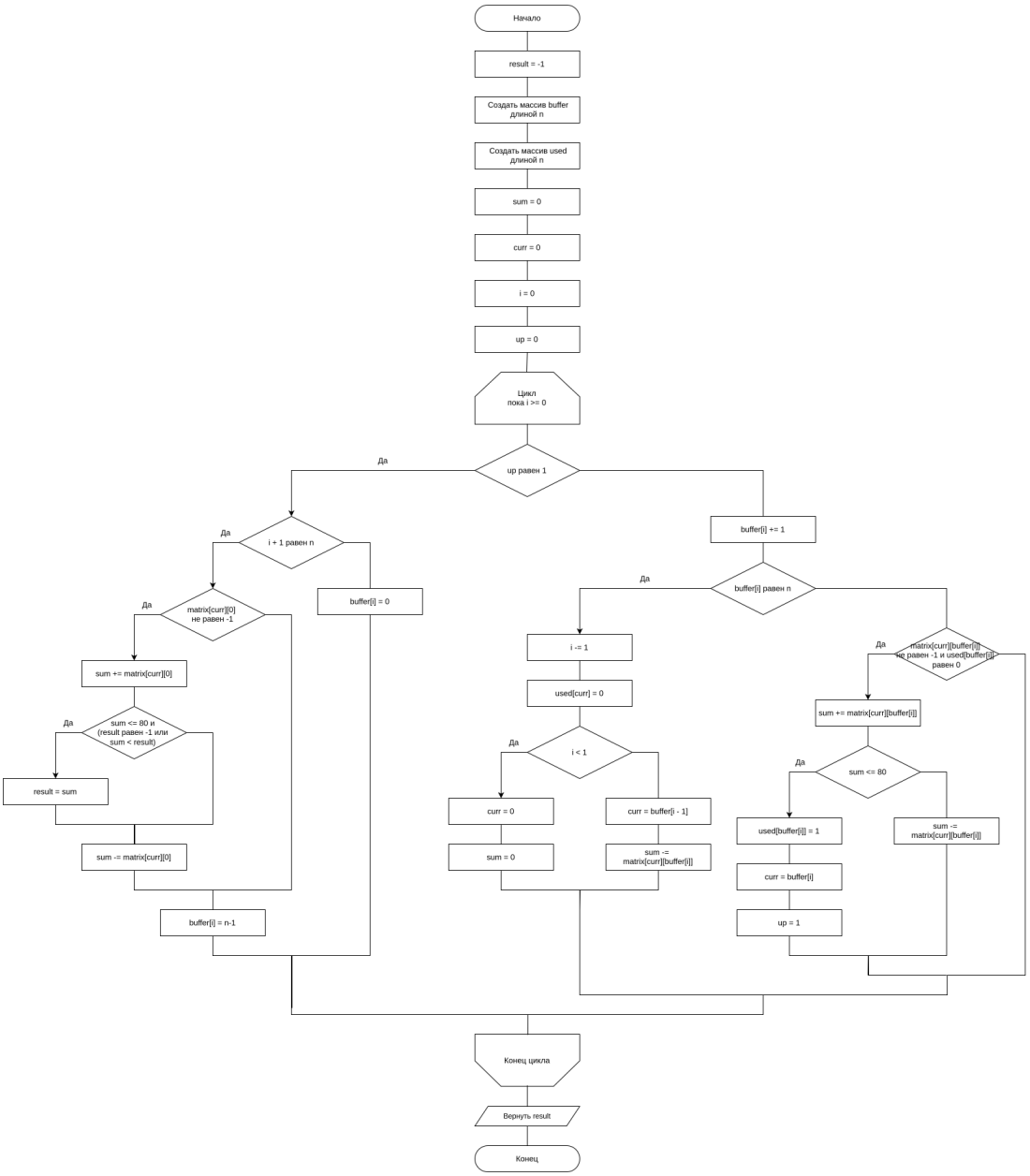

Рисунок 2.1 – Схема алгоритма, использующего полный перебор

В худшем случае дороги есть между всеми городами. Трудоёмкость блока начальной инициализации данных равна 5. Трудоёмкость блока про-

верки нового цикла в случае обновления составляет 21, в случае сохранения старой длины — 20, при этом количество возможных рассматриваемых циклов равно $\frac{(n-1)!}{2}$, при этом каждый цикл проверяется 2 раза в зависимости от направления. Трудоёмкость блока выставления курсора равна 6, при этом курсор выставляется $(n-1)!$ раз. Трудоёмкость блока возврата равна 12 при возврате на нулевой индекс и 16 в противном случае, при этом возврат на нулевой индекс происходит n раз, на остальные — $(n-2)!$ раз. Трудоёмкость блока входа в новый город составляет 25, вход в новый город происходит $(n-1)!$ раз. Трудоёмкость игнорирования города составляет 14, игнорирование города происходит $(n-2)\cdot(n-1)!$ раз. Тогда трудоёмкость поиска полным перебором в худшем случае $f_{worst\_complete}$ равна $5+20.5\cdot(n-1)!+6\cdot(n-1)!+12\cdot n+16\cdot(n-2)!+25\cdot(n-1)!+14\cdot(n-2)\cdot(n-1)!+1 = 14\cdot n! - 1.5\cdot(n-1)! + 16\cdot(n-2)! + 12\cdot n + 1$.

2.2 Муравьиный алгоритм

На рисунках 2.2-2.4 представлена схема муравьиного алгоритма. На вход он получает количество городов N, матрицу смежности matrix размером N на N, коэффициент стадности alpha, коэффициент испарения rho и количество дней days.

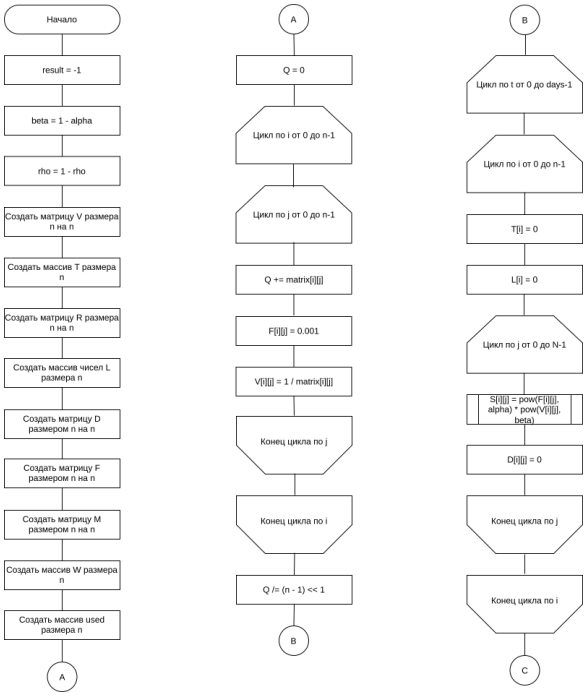

Рисунок 2.2 – Схема муравьиного алгоритма (начало)

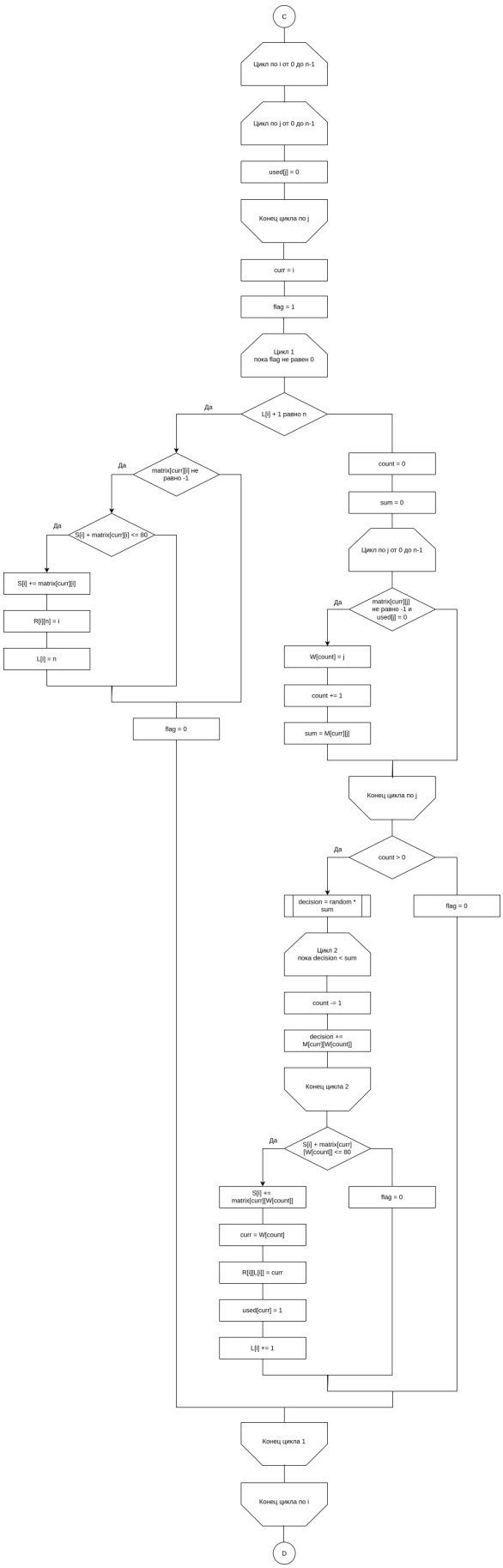

Рисунок 2.3 – Схема муравьиного алгоритма (продолжение)

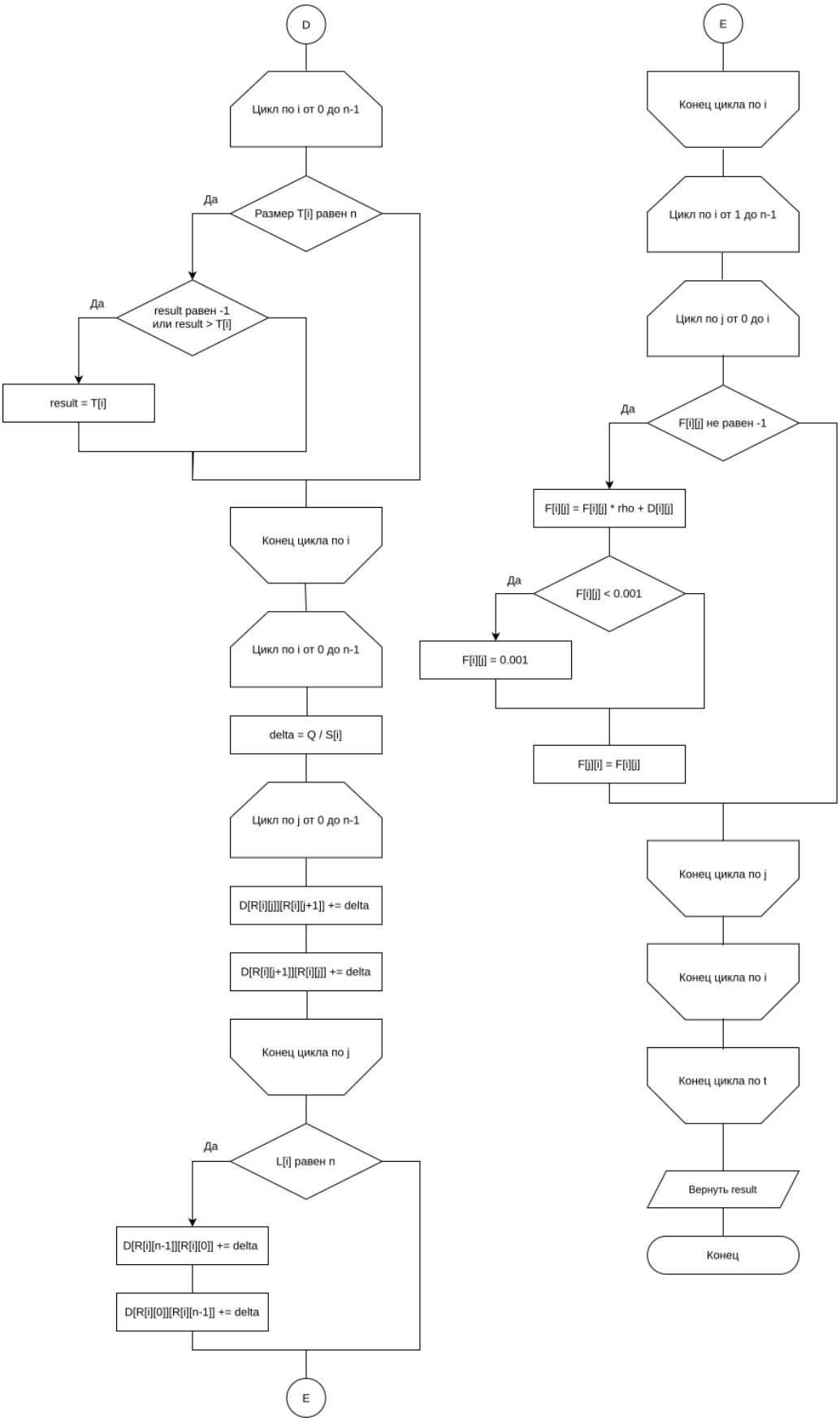

Рисунок 2.4 – Схема муравьиного алгоритма (окончание)

В худшем случае дороги есть между всеми городами. Трудоёмкость муравьиного алгоритма f_{worst_ants} рассчитывается по формуле (2.1):

$$f_{worst\_ants} = f_{prep} + 2 + days \cdot (2 + f_{morning} + f_{day} + f_{evening} + f_{night}) + 1 \quad (2.1)$$

где f_{prep} — трудоёмкость этапа подготовки данных, $f_{morning}$ — трудоёмкость фазы утра, f_{day} — трудоёмкость фазы дня, $f_{evening}$ — трудоёмкость фазы вечера, f_{night} — трудоёмкость фазы ночи.

Трудоёмкость этапа подготовки данных f_{prep} равна $4 + 2 + n \cdot (2 + 2 + n \cdot (2 + 11)) + 4 = 13 \cdot n^2 + 4 \cdot n + 10$. Трудоёмкость фазы утра $f_{morning}$ равна $2 + n \cdot (2 + 4 + 2 + n \cdot (2 + 15)) = 17 \cdot n^2 + 8 \cdot n + 2$. Трудоёмкость фазы дня f_{day} в худшем случае равна $2 + n \cdot (2 + 2 + n \cdot 4 + 2 + (n - 1) \cdot 31 + \frac{n(n-1)}{2} \cdot 11 + 20) = 5.5 \cdot n^3 + 29.5 \cdot n^2 - 7 \cdot n + 2$. Трудоёмкость фазы вечера $f_{evening}$ в худшем случае равна $9 \cdot n + 2$. Трудоёмкость фазы ночи f_{night} в худшем случае равна $2 + n \cdot (2 + 4 + 2 + n \cdot (2 + 16) + 16) + 2 + 4 \cdot (n - 1) + \frac{n(n-1)}{2} \cdot 26 - n \cdot 3 = 18 \cdot n^2 + 24 \cdot n + 2 + 13 \cdot n^2 - 7 \cdot n - 2 = 31 \cdot n^2 + 17 \cdot n$.

Трудоёмкость муравьиного алгоритма f_{worst_ants} в худшем случае равна $13 \cdot n^2 + 4 \cdot n + 10 + 2 + days \cdot (2 + 17 \cdot n^2 + 8 \cdot n + 2 + 5.5 \cdot n^3 + 29.5 \cdot n^2 - 7 \cdot n + 2 + 9 \cdot n + 2 + 31 \cdot n^2 + 17 \cdot n) = 13 \cdot n^2 + 4 \cdot n + 12 + days \cdot (5.5 \cdot n^3 + 77.5 \cdot n^2 + 2 \cdot n + 8)$.

Выводы

В данном разделе были построены схемы алгоритмов, а также была выполнена оценка трудоёмкости алгоритмов.

3 Технологическая часть

3.1 Средства реализации

Для реализации программы был выбран язык программирования Java, так как он содержит все необходимые средства для реализации программы.

3.2 Реализации алгоритмов

В листингах 3.1–3.2 представлена реализация алгоритма решения задачи коммивояжёра полным перебором.

Листинг 3.1 – Реализация алгоритма решения задачи коммивояжёра полным перебором (начало)

```
@Override
public double solve(double @NotNull [] @NotNull [] matrix) {
    var result = super.solve(matrix);
    var buffer = new int[matrix.length];
    var used = new boolean[matrix.length];
    var sum = .0;
    var current = 0;
    var index = 0;
    var up = false;
    while (index >= 0) {
        if (up) {
            if (index + 1 == matrix.length) {
                if (matrix[current][0] != -1) {
                    sum += matrix[current][0];
                    if (sum < DAYS + EPSILON && (result == -1 ||
                       sum < result)) {
                        result = sum;
                    }
                    sum -= matrix[current][0];
                }
                buffer[index] = matrix.length - 1;
            } else {
                buffer[index] = 0;
            }
            up = false;
        } else {
            ++buffer[index];
            if (buffer[index] == matrix.length) {
```

Листинг 3.2 – Реализация алгоритма решения задачи коммивояжёра полным перебором (окончание)

```
                --index;
                used[current] = false;
                if (index < 1) {
                    current = 0;
                    sum = 0;
                } else {
                    current = buffer[index - 1];
                    sum -= matrix[current][buffer[index]];
                }
            } else if (matrix[current][buffer[index]] != -1 && !
               used[buffer[index]]) {
                sum += matrix[current][buffer[index]];
                if (sum < DAYS + EPSILON) {
                    used[buffer[index]] = true;
                    current = buffer[index];
                    up = true;
                    ++index;
                } else {
                    sum -= matrix[current][buffer[index]];
                }
            }
        }
    }
    return result;
}
```

В листингах 3.3–3.6 представлена реализация алгоритма решения задачи коммивояжёра полным перебором.

Листинг 3.3 – Реализация муравьиного алгоритма (начало)

```
@Override
public double solve(double @NotNull [] @NotNull [] matrix) {
    var result = super.solve(matrix);
    var visibility = new double[matrix.length][matrix.length];
    var results = new double[matrix.length];
    var routes = new int[matrix.length][matrix.length];
    var lengths = new int[matrix.length];
    var delta = new double[matrix.length][matrix.length];
    var pheromones = new double[matrix.length][matrix.length];
    var multi = new double[matrix.length][matrix.length];
```

Листинг 3.4 – Реализация муравьиного алгоритма (продолжение)

```
    var ways = new int[matrix.length];
    var used = new boolean[matrix.length];
    var quote = .0;
    for (var i = 0; i < matrix.length; ++i) {
        for (var j = 0; j < matrix[i].length; ++j) {
            pheromones[i][j] = EPSILON;
            visibility[i][j] = 1 / matrix[i][j];
            quote += matrix[i][j];
        }
    }
    quote /= (matrix.length - 1) << 1;
    for (var day = 0; day < days; ++day) {
        // Morning
        for (var i = 0; i < matrix.length; ++i) {
            results[i] = 0;
            lengths[i] = 0;
            for (var j = 0; j < matrix.length; ++j) {
                delta[i][j] = 0;
                multi[i][j] = Math.pow(pheromones[i][j], alpha) *
                      Math.pow(visibility[i][j], beta);
            }
        }
        // Day
        for (var ant = 0; ant < matrix.length; ++ant) {
            Arrays.fill(used, false);
            var flag = true;
            var curr = ant;
            while (flag) {
                if (lengths[ant] + 1 == matrix.length) {
                    if (matrix[curr][ant] != -1 && results[ant] +
                        matrix[curr][ant] <   DAYS + EPSILON) {
                        results[ant] += matrix[curr][ant];
                        routes[ant][lengths[ant]++] = ant;
                    }
                    flag = false;
                } else {
                    var count = 0;
                    var sum = .0;
                    for (var i = 0; i < matrix.length; ++i) {
```

Листинг 3.5 – Реализация муравьиного алгоритма (продолжение)

```
                    if (matrix[curr][i] != -1 && !used[i] &&
                       i != ant) {
                        ways[count++] = i;
                        sum += multi[curr][i];
                    }
                }
                if (count > 0) {
                    var decision = Math.random() * sum;
                    while (decision < sum) {
                        decision += multi[curr][ways[--count
                           ]];
                    }
                    if (results[ant] + matrix[curr][ways[
                       count]] < DAYS + EPSILON) {
                        results[ant] += matrix[curr][ways[
                           count]];
                        curr = ways[count];
                        routes[ant][lengths[ant]++] = curr;
                        used[curr] = true;
                    } else {
                        flag = false;
                    }
                } else {
                    flag = false;
                }
            }
        }
    }
    // Evening
    for (var ant = 0; ant < matrix.length; ++ant) {
        if (lengths[ant] == matrix.length && (result == -1 ||
            results[ant] < result)) {
            result = results[ant];
        }
    }
    // Night
    for (var ant = 0; ant < matrix.length; ++ant) {
        if (lengths[ant] > 0) {
            var antDelta = quote / results[ant];
            for (var i = 0; i < lengths[ant] - 1; ++i) {
```

Листинг 3.6 – Реализация муравьиного алгоритма (окончание)

```
                    delta[routes[ant][i]][routes[ant][i + 1]] +=
                       antDelta;
                    delta[routes[ant][i + 1]][routes[ant][i]] +=
                       antDelta;
                }
                if (lengths[ant] == matrix.length) {
                    delta[routes[ant][matrix.length - 1]][routes[
                       ant][0]] += antDelta;
                    delta[routes[ant][0]][routes[ant][matrix.
                       length - 1]] += antDelta;
                }
            }
        }
        for (var i = 1; i < matrix.length; ++i) {
            for (var j = 0; j < i; ++j) {
                pheromones[i][j] = delta[i][j] + rho * pheromones
                   [i][j];
                if (pheromones[i][j] < EPSILON) {
                    pheromones[i][j] = EPSILON;
                }
                pheromones[j][i] = pheromones[i][j];
            }
        }
    }
    return result;
}
```

3.3 Функциональное тестирование

В таблице 3.1 представлены результаты выполнения функциональных тестов для алгоритма полного перебора. На вход в первой строке программа получает количество маршрутов между городами N — натуральное число. В последующих $3 \cdot N$ строках программа получает N описаний маршрутов по 3 строки каждое — в первой строке идёт название первого города, во второй — второго, в третьей — время пути между ними. Названия городов не должны совпадать или быть пустыми, а длина пути не должна быть меньше 0.001. Программа возвращает длину кратчайшего гамильтонова цикла, либо -1, если цикл не найден или все существующие циклы занимают больше 80 дней.

Таблица 3.1 – Результаты выполнения функциональных тестов

Входные данные	Ожидаемый вывод	Полученный вывод
	Ошибка. Неверный формат ввода.	Ошибка. Неверный формат ввода.
-1	Ошибка. Неверный формат ввода.	Ошибка. Неверный формат ввода.
1 Омск Омск	Ошибка. Неверный формат ввода.	Ошибка. Неверный формат ввода.
1 Воронеж Москва -3	Ошибка. Неверный формат ввода.	Ошибка. Неверный формат ввода.
1 Воронеж Москва 0.25	0.500	0.500
2 Воронеж Москва 0.25 Москва Казань 0.5	-1.000	-1.000

В таблице 3.2–3.3 представлены результаты выполнения функциональных тестов для муравьиного алгоритма. На вход в первой строке программа получает количество маршрутов между городами N — натуральное число. В последующих $3 \cdot N$ строках программа получает N описаний маршрутов по 3 строки каждое — в первой строке идёт название первого города, во второй — второго, в третьей — время пути между ними. Названия городов не должны совпадать или быть пустыми, а длина пути не должна быть меньше 0.001. Вместо результата была проведена проверка полученного маршрута. Маршруты были получены из матрицы *routes* при помощи отладчика. Корректным

считается маршрут, в котором посещены все вершины и в котором нет переходов по несуществующим рёбрам. Все полученные маршруты оказались корректными.

Таблица 3.2 – Результаты выполнения функциональных тестов (начало)

Входные данные	Маршрут
7 0 1 10 1 2 10 2 3 15 3 4 35 0 4 10 4 1 20 3 0 20	[4, 1, 2, 3, 0]

Таблица 3.3 – Результаты выполнения функциональных тестов (окончание)

Входные данные	Маршрут
5 0 1 1 1 2 2 2 3 3 3 4 4 4 0 5	[1, 2, 3, 4, 0]

Выводы

В данном разделе были выбраны средства реализации, разработана спроектированная программа, представлен графический интерфейс и проведено тестирование.

4 Исследовательская часть

4.1 Параметризация муравьиного алгоритма

Параметризация проводилась по параметрам *alpha* — коэффициенту стадности, *rho* — коэффициенту испарения, T — количеству дней. *alpha* варьировался от 0.1 до 0.9 с шагом 0.2, *rho* аналогично, T принимал значения 5, 10, 50, 100, 500. Для параметризации были подготовлены 3 класса данных.

В класс данных G_1 входят следующие города: Москва, Стамбул, Вашингтон, Лондон, Пекин, Бразилиа, Камберра, Нью-Дели, Тегеран и Буэнос-Айрес. Связи между городами представлены в таблице 4.1.

Таблица 4.1 – Связи в классе данных G_1

Город 1	Город 2	Время
Москва	Стамбул	2.7
Москва	Пекин	6.6
Стамбул	Вашингтон	12.5
Стамбул	Лондон	10.5
Вашингтон	Лондон	3
Вашингтон	Камберра	17.6
Лондон	Камберра	13.2
Лондон	Буэнос-Айрес	13.8
Пекин	Бразилиа	11.9
Пекин	Тегеран	6.6
Бразилиа	Тегеран	9.9
Бразилиа	Буэнос-Айрес	1.5
Камберра	Нью-Дели	4
Камберра	Бразилиа	13.6
Нью-Дели	Пекин	4
Нью-Дели	Москва	8.9
Тегеран	Москва	5.4
Тегеран	Нью-Дели	6
Буэнос-Айрес	Камберра	15
Буэнос-Айрес	Нью-Дели	12.2

В класс данных G_2 входят следующие города: Санкт-Петербург, Шанхай, Нью-Йорк, Сидней, Рио-де-Жанейро, Венеция, Киото, Анталья, Хургада и Ливерпуль. Связи между городами представлены в таблице 4.2.

Таблица 4.2 – Связи в классе данных G_2

Город 1	Город 2	Время
Санкт-Петербург	Шанхай	7.8
Санкт-Петербург	Анталья	2.4
Шанхай	Киото	8
Шанхай	Анталья	3
Нью-Йорк	Сидней	11.9
Нью-Йорк	Ливерпуль	6.4
Сидней	Рио-де-Жанейро	6.6
Сидней	Хургада	8.4
Рио-де-Жанейро	Санкт-Петербург	10
Рио-де-Жанейро	Венеция	8
Венеция	Киото	7.3
Венеция	Шанхай	8.1
Киото	Нью-Йорк	4.2
Киото	Сидней	2.3
Анталья	Хургада	1.4
Анталья	Венеция	3.2
Хургада	Санкт-Петербург	3.3
Хургада	Ливерпуль	4.7
Ливерпуль	Киото	8.4
Ливерпуль	Рио-де-Жанейро	10.3

В класс данных G_3 входят следующие города: Воронеж, Воркута, Омск, Владивосток, Челябинск, Нижний Тагил, Тобольск, Рязань, Екатеринбург и Волгоград. Связи между городами представлены в таблице 4.3.

Таблица 4.3 – Связи в классе данных G_3

Город 1	Город 2	Время
Воронеж	Рязань	1.5
Воронеж	Волгоград	2.5
Воркута	Нижний Тагил	6
Воркута	Челябинск	6.2
Омск	Тобольск	7.8
Омск	Екатеринбург	8.1
Владивосток	Омск	9
Владивосток	Воронеж	11.9
Челябинск	Владивосток	7.6
Челябинск	Тобольск	6.6
Нижний Тагил	Владивосток	8.6
Тобольск	Екатеринбург	0.5
Тобольск	Рязань	6.9
Рязань	Нижний Тагил	7.6
Рязань	Волгоград	4.5
Екатеринбург	Воркута	5.4
Екатеринбург	Воронеж	6.3
Волгоград	Челябинск	5.8
Волгоград	Омск	8.8

В ходе параметризации для каждого набора параметров и каждого класса данных было произведено 10 запусков, в результате которых для класса данных G_i были вычислены максимальное отклонение max_i, медиана med_i и среднее арифметическое отклонений avg_i. Результаты параметризации представлены в Приложении А.

Наименьшие отклонения были при минимальных значениях коэффициента стадности $alpha$ и коэффициента испарения rho. При значениях времени поиска t 100 и 500 во всех случаях было найдено оптимальное решение.

4.2 Замеры времени работы

Замеры проводились на ноутбуке Acer Swift SF314-510G, процессор 1th Gen Intel® Core™ i7-1165G7. Для замера процессорного времени использовался метод *getCurrentThreadCpuTime* класса *ThreadMXBean* из пакета *java.lang.management* [4]. Сравнения проводились на графах с 2, 4, 6, 8 и 10 вершинами. Для муравьиного алгоритма количество дней t было принято за 50. Для каждого размера графа было произведено 100 запусков, после чего было взято среднее время работы. Результаты замеров представлены на рисунке 4.1.

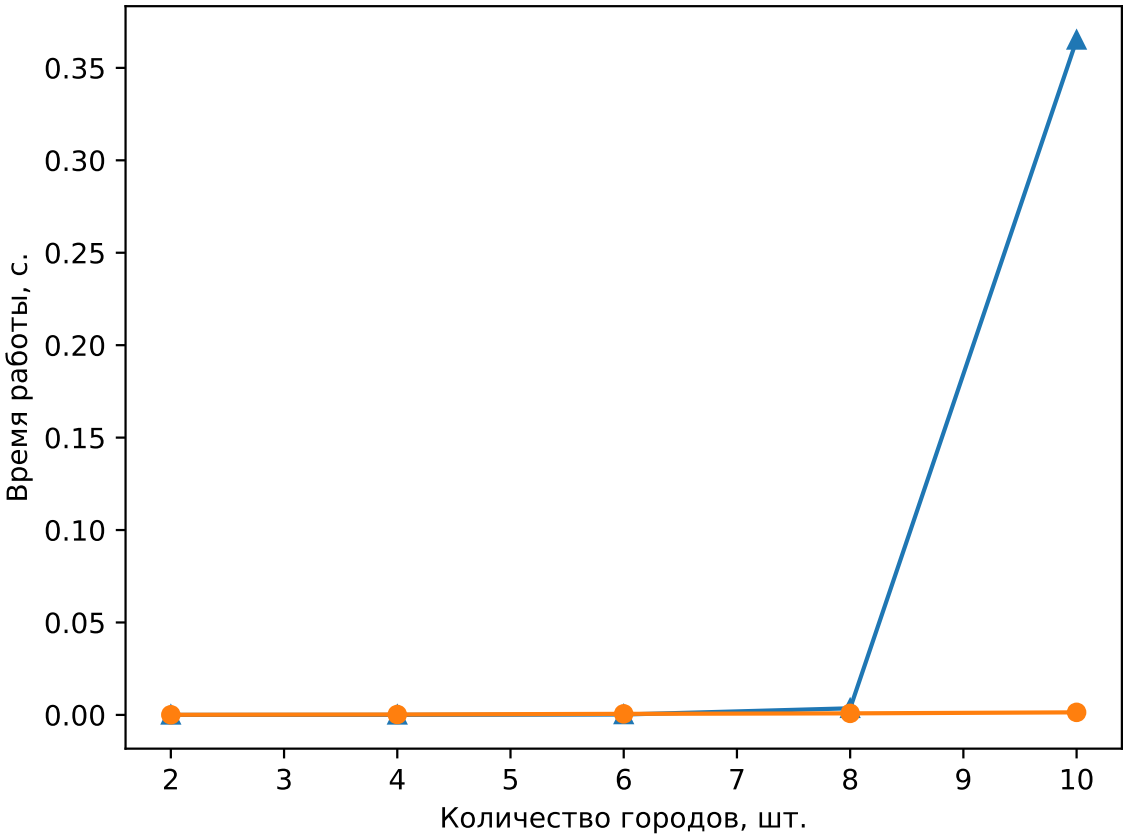

Рисунок 4.1 – Результаты замеров времени для алгоритмов решения задачи коммивояжёра

В ходе замеров было выявлено, что муравьиный алгоритм работает быстрее алгоритма полного перебора.

4.3 Выводы

В данном разделе была проведена параметризация муравьиного алгоритма, а также были проведены замеры времени работы для алгоритма полного перебора и муравьиного алгоритма.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы цель была достигнута. Были выполнены все поставленные задачи:

1) были описаны постановка задачи коммивояжёра, метод решения полным перебором и метод на основе муравьиного алгоритма;
2) была написана программа, реализующая оба алгоритма;
3) была проведена параметризация муравьиного алгоритма по 3 параметрам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Бороздов В. О.* Исследование решения задачи коммивояжёра. — Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика, 2009.
2. *Шведов. А. В.*, *Гадасин. Д. В.*, *Вакурин. И. В.* Разгрузка очереди сети при помощи Гамильтонова цикла. — REDS: Телекоммуникационные устройства и системы, №3, 2021. — С. 44—52.
3. *Штовба С. Д.* Муравьиные алгоритмы. — Мастерская решений, 2003. — С. 70—75.
4. Документация языка программирования *Java* – класс $ThreadMXBean$ [Электронный ресурс]. — Режим доступа: `https://docs.oracle.com/javase/8/docs/api/java/lang/management/ThreadMXBean.html#getCurrentThreadCpuTime--` (дата обращения: 20.12.2024).

ПРИЛОЖЕНИЕ А