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 №6
по дисциплине «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков
Студент Звягин Д.О.
Группа ИУ7-53Б
Преподаватель Волкова Л.Л., Строганов Д.В.

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Задача коммивояжера 4

1.2 Метод полного перебора 4

1.3 Метод на основе муравьиного алгоритма 4

1.3.1 Математическая модель метода на основе муравьиного алгоритма 5

2 Конструкторская часть **8**

2.1 Разработка алгоритмов 8

3 Технологическая часть **15**

3.1 Требования к программному обеспечению 15

3.2 Средства реализации 15

3.3 Реализация алгоритмов 16

3.4 Функциональные тесты 22

4 Исследовательская часть **23**

4.1 Технические характеристики 23

4.2 Результаты проводимых исследований 23

4.3 Параметризация 23

4.3.1 Класс данных 24

ЗАКЛЮЧЕНИЕ **26**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **27**

Приложение А **28**

ВВЕДЕНИЕ

В лабораторной работе будут рассмотрены методы решения задачи коммивояжера.

Целью данной лабораторной работы является рассмотрение методов решения задачи коммивояжера, а именно метод полного перебора и муравьиный (роевой) алгоритм.

Для достижения поставленной цели необходимо решить следующие задачи:

1) рассмотреть методы решения задачи коммивояжера;
2) разработать рассмотренные методы;
3) реализовать разработанные алгоритмы;
4) провести анализ временных затрат реализованных алгоритмов;
5) провести параметризацию для муравьиного (роевого) алгоритма.

Индивидуальный вариант для выполнения лабораторной работы:

— неориентированный граф;
— с элитными муравьями;
— карта перемещения по Африке;
— гамильтонов цикл.

1 Аналитическая часть

1.1 Задача коммивояжера

Обычно задачу коммивояжера формулируют следующим образом [1]: Пусть дан список городов и расстояний между каждой парой городов, необходимо найти кратчайший маршрут, в котором будет единожды посещён каждый город и вернётся в исходную точку.

Данная формулировка подразумевает возврат в исходный город, то есть поиск “Гамильтонова” пути. Существуют и формулировки, не требующие точного возврата в исходную точку.

Существует ряд методов для решения этой задачи. В данной работе будут рассмотрены метод полного перебора и метод с использованием “Муравьиного” (роевого) алгоритма.

1.2 Метод полного перебора

При использовании данного метода решение задачи получается путем перебора всех возможных подходящих путей и выбор наименьшего из них, путём поиска наименьшего суммарного расстояния.

Преимущество данного метода заключается в том, полученный результат гарантированно является оптимальным решением.

Главный недостатком этого метода является его алгоритмическая сложность. Полный перебор подразумевает сложность — O($n!$), где n — количество вершин графа городов [2].

1.3 Метод на основе муравьиного алгоритма

Данный метод основан на имитации природных механизмов самоорганизации муравьев [3].

Если наблюдать за муравьями, можно увидеть, что они почти всегда ходят по конкретным путям, и эти пути зачастую являются довольно эффективными.

В природе муравьи используют свои органы чувств для ориентации в пространстве, а также феромон для непрямого обмена информацией друг с другом.

Имитация поведения колонии муравьёв позволяет эффективно решать за-

дачи для поиска путей на графах, так как сам феромон и его свойство — испарение, позволяют не задерживаться в локальных экстремумах для комбинаторных задач или обходить разные пути как, например, в задаче коммивояжера.

1.3 Математическая модель метода на основе муравьиного алгоритма

Пусть муравей имеет следующие характеристики-чувства:

1) зрение — способность определить длину ребра;
2) обоняние — способность чуять феромон;
3) память — способность запомнить пройденный маршрут.

Когда перед муравьём встаёт задача выбора одного ребра графа из нескольких возможных, он ориентируется на свои чувства.

Для нашей задачи, запомнив посещённые вершины графа, муравей не станет выбирать их.

Ориентируясь на длину ребра с помощью зрения, посредством функции (1), определяется начальная привлекательность ребра, в зависимости от его длины (значения метки ребра в матрице смежности):

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

где D_{ij} — расстояние от текущей вершины i до заданной вершины j.

Как видно из вершины, привлекательность ребра обратно пропорциональна его длине [3].

Вероятность перехода муравья в конкретную вершину графа на основе всех чувств выражается формулой (2).

$$p_{k,ij} = \begin{cases} \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum_{q \notin J_k} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, j \notin J_k \\ 0, j \in J_k \end{cases} \tag{2}$$

где α — влияния длины пути, β — влияния феромона, τ_{ij} — количество феромона на ребре ij, η_{ij} — привлекательность ребра ij на основе его длины, J_k — список посещенных за текущий день вершин. При этом коэффициенты α и β связаны отношением $\beta = 1 - \alpha$.

Муравьиный алгоритм работает, симулируя несколько “дней” работы колонии.

Каждый “день” каждый муравей проходит по одному пути.

Ночью, перед наступлением очередного дня (после завершения работы всех муравьев), значение феромона обновляется по формуле (3).

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1-p) + \Delta\tau_{ij}(t). \tag{3}$$

При этом

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{N} \Delta\tau_{ij}^{k}(t), \tag{4}$$

где

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q/L_k, \text{ребро посещено муравьем } k \text{ в текущий день } t, \\ 0, \text{иначе,} \end{cases} \tag{5}$$

где Q — “квота”, общее количество феромона, выделяемое муравьём. Значение Q равно средней длине ребра графа (или средней метке вершин графа). L_k — общая длина пути, пройденного муравьём k в этот день.

Так как при переходах между вершинами используется вероятность (2), то необходимо проверить, чтобы она не стала нулевой, а именно ввести значение τ_{min} — минимально возможное значение феромона, и не допускать уменьшение значения феромона ниже τ_{min}.

В итоге путь муравья выбирается по следующему алгоритму:

1) каждый муравей имеет список запретов — список уже посещенных вершин графа;
2) зрение отвечает за эвристическое желание посетить вершину;
3) обоняние отвечает за ощущение феромона на определенном ребре. При этом количество феромона на ребре в день t обозначается как $\tau_{i,j}(t)$;
4) при прохождение ребра муравей откладывает на нем феромон, который показывает оптимальность выбора данного ребра, его количество вычисляется по формуле (5).

Следует отметить, что обновление значения феромона происходит только в конце очередного для, то есть если в течение одного дня муравей один муравей

прошёл по определённому пути, значение феромона на этом пути не будет обновлено до окончания этого дня.

Если все муравьи начинают свой путь в одной конкретной вершине, то решение задачи может потерять в точности, так как оптимальные пути выхода с этой вершины будут получать всё больше и больше феромона. Согласно постановке задачи, вершина начала пути не определена, то есть может быть выбрана любая из доступных.

Также существуют вариации муравьиного алгоритма, значительно увеличивающие его эффективность.

В этой работе рассматривается вариация с “элитными” муравьями. Элитные муравьи проходят свой путь только по наилучшему маршруту, найденному в алгоритме на данной итерации.

Такое поведение искусственно добавляет дополнительный феромон на рёбра лучшего маршрута, найденного на данный момент.

Такой алгоритм может получить решения, отличающиеся от того, что получит обычный муравьиный алгоритм.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунке 1 представлена схема алгоритма для получения всех перестановок в массиве. На рисунке 2 представлена схема алгоритма полного перебора. На рисунках 3 и 4 представлена схема муравьиного алгоритма. А также на рисунках 5 - 7 представлены схемы вспомогательных частей муравьиного алгоритма.

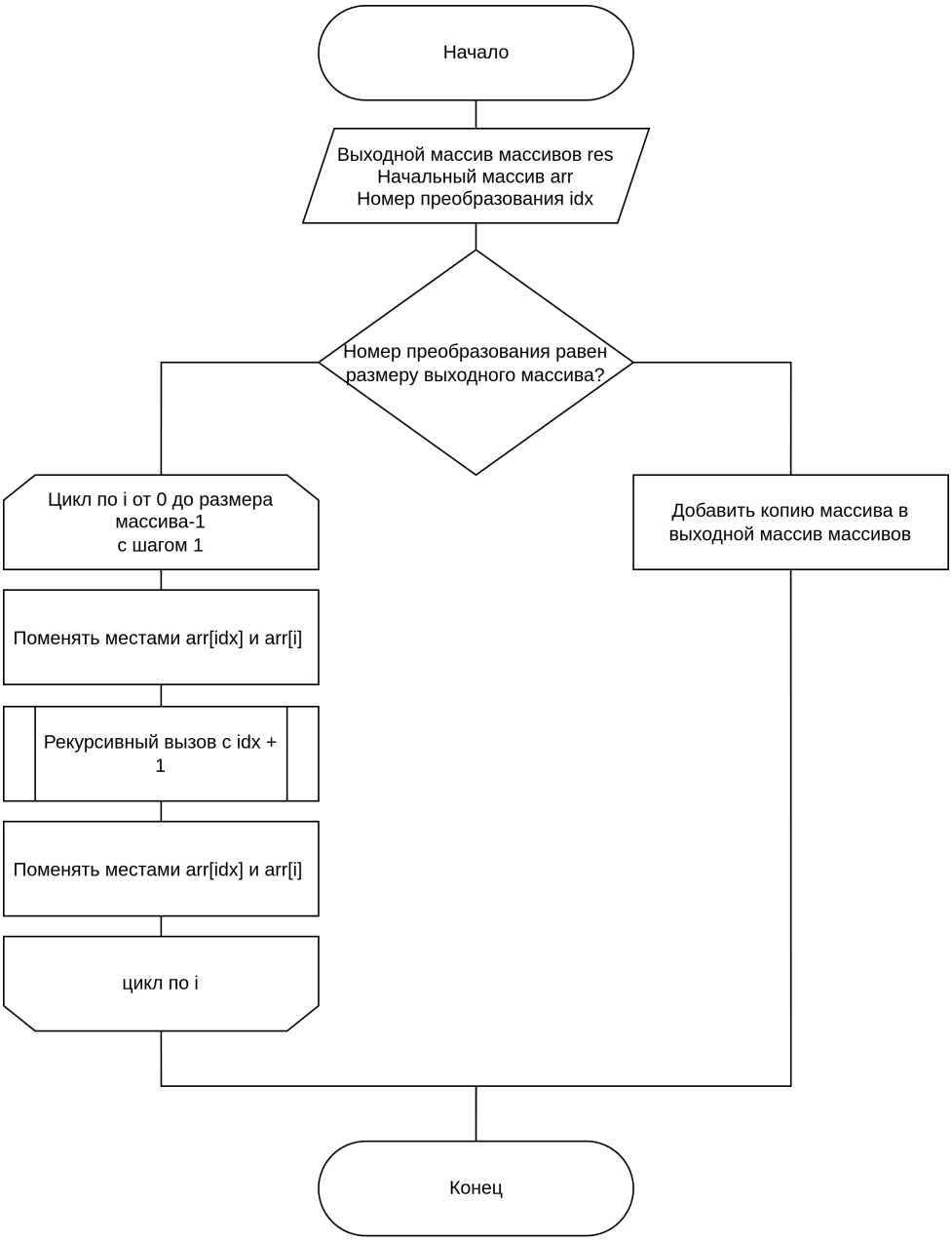

Рисунок 1 — Алгоритм получения преобразований

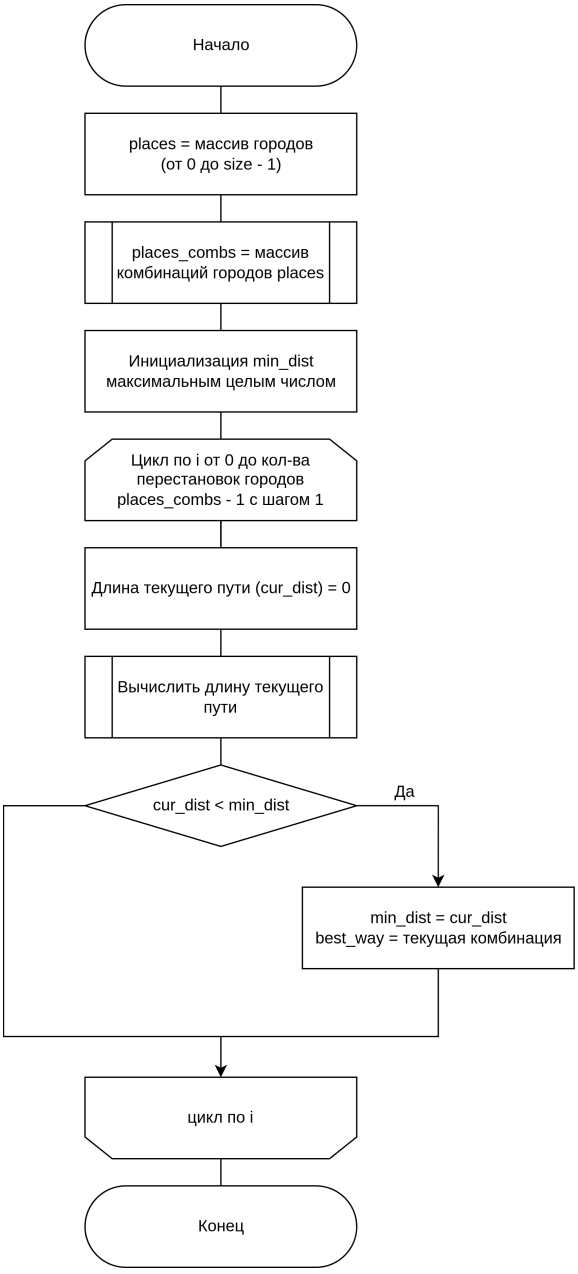

Рисунок 2 — Алгоритм полного перебора

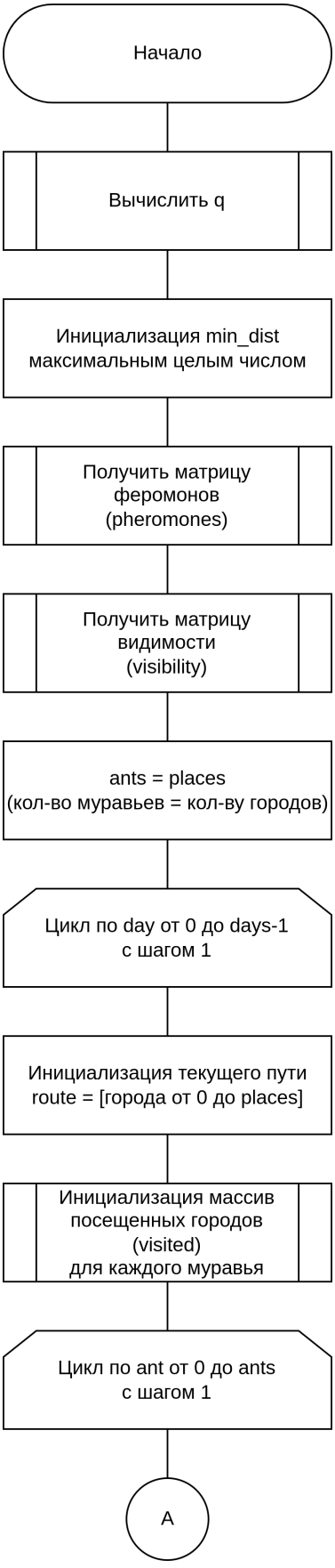

Рисунок 3 — Муравьиный алгоритм (часть 1)

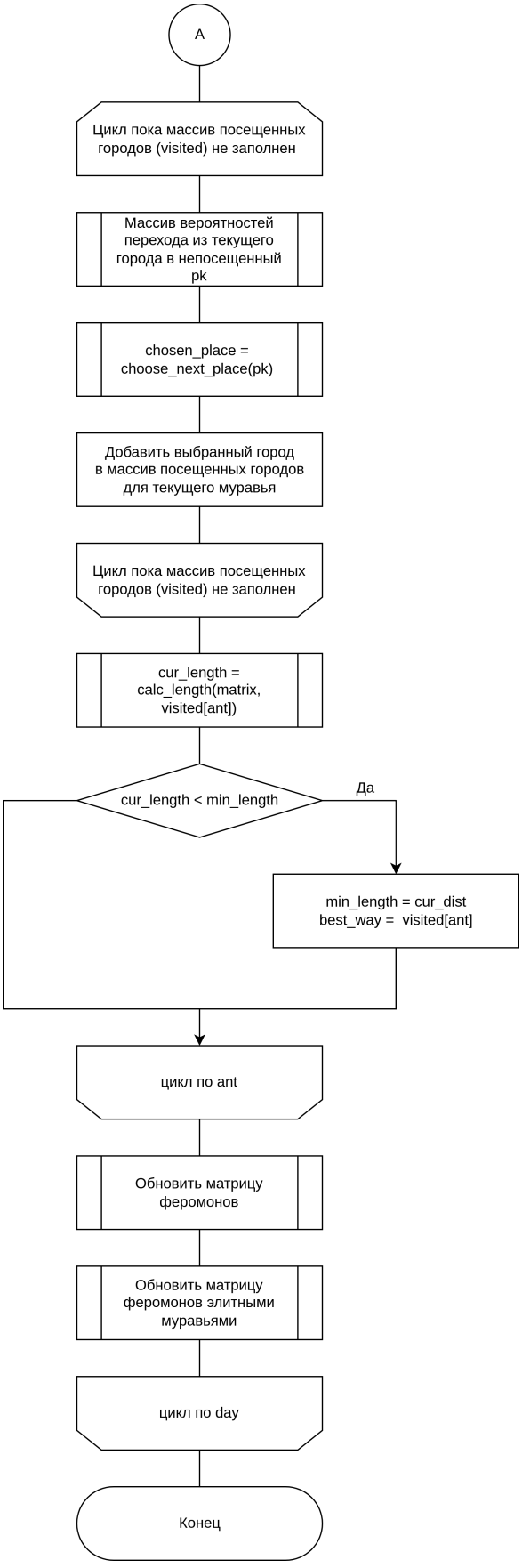

Рисунок 4 — Муравьиный алгоритм (часть 2)

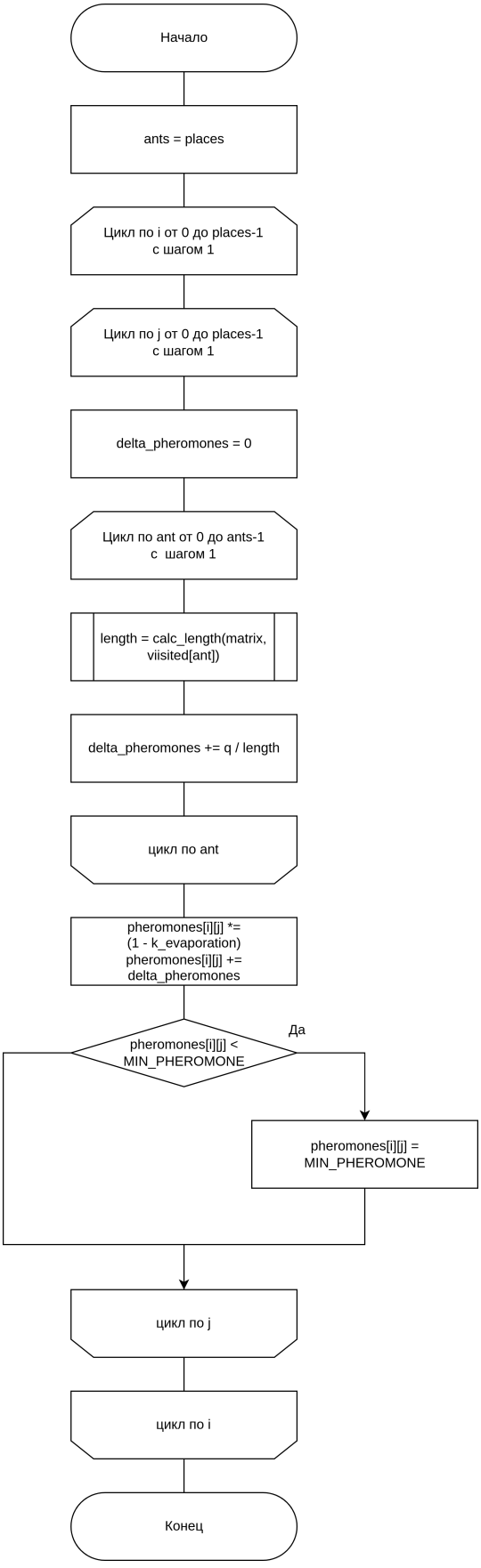

Рисунок 5 — Алгоритм обновления матрицы феромонов

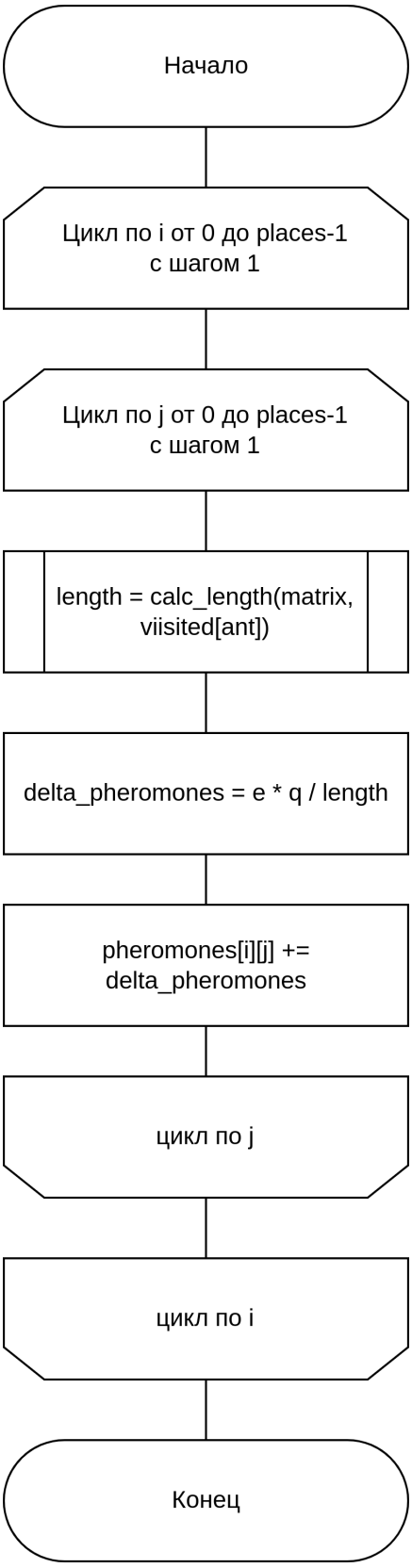

Рисунок 6 — Алгоритм обновления матрицы феромонов для элитных муравьев

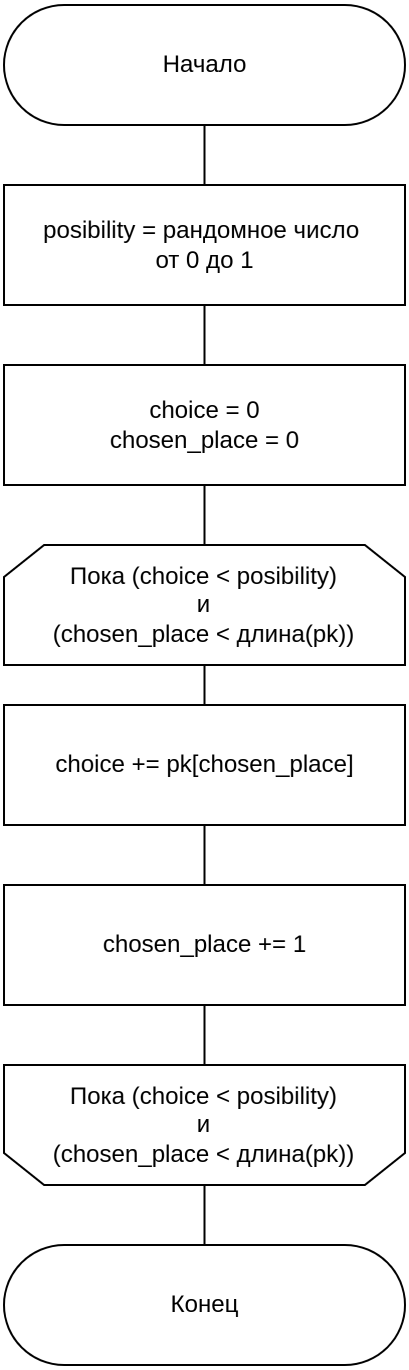

Рисунок 7 — Алгоритм выбора следующего города

3 Технологическая часть

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— входные данные — матрица расстояний между городами;

— выходные данные — наилучший маршрут, найденный алгоритмами.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык C++ стандарта C++20 [4]. Выбор сделан в пользу данного языка в связи с его высоким быстродействием, в связи с наличием у него встроенного модуля $ctime$ для измерения процессорного времени, а также из-за наличия всех необходимых структур данных для реализации алгоритмов.

3.3 Реализация алгоритмов

Реализация алгоритма полного перебора изображена на листинге 1

Реализация муравьиного алгоритма перебора изображена на листинге 2

Листинг 1 — Реализация алгоритма полного перебора

```
#include "FullPF.hpp"

#include <algorithm>
#include <limits>
#include <ranges>

std::pair<double, std::vector<int>>
FullCombinationPathFinder::find(const Matrix &mat) {
    std::vector<int> places(mat.size());
    for (const int &i : std::views::iota(0ull, mat.size()))
    places[i] = i;

    std::vector<std::vector<int>> placesCombinations;
    while (std::next_permutation(places.begin(), places.end()))
    placesCombinations.push_back(places);

    double minDist = std::numeric_limits<double>::infinity();
    std::vector<int> bestWay;

    for (const auto &combination : placesCombinations) {
    double curDist = 0;
    for (const int &j : std::views::iota(0ull, mat.size() - 1)) {
        int startCity = combination[j];
        int endCity = combination[j + 1];
        curDist += mat[startCity][endCity];
    }

    curDist += mat[combination[mat.size() - 1]][combination[0]];

    if (curDist < minDist) {
        minDist = curDist;
        bestWay = combination;
        bestWay.push_back(bestWay[0]);
    }
```

```
    }

    return {minDist, bestWay};
}
```

Листинг 2 — Реализация муравьиного алгоритма

```
#include "AntPF.hpp"

#include <algorithm>
#include <cmath>
#include <iostream>
#include <limits>
#include <random>
#include <ranges>
#include <vector>

const double MIN_PHEROMONE = 1e-10;

std::pair<double, std::vector<int>> AntPathFinder::find(const Matrix
    &mat) {
    double q = calc_q(mat);
    std::vector<int> best_way;
    double min_length = std::numeric_limits<double>::infinity();

    Matrix pheromones = get_pheromones(mat);
    Matrix visibility = get_visibility(mat);
    int ants = mat.size();

    for (const auto &_ : std::views::iota(0, m_days)) {
    std::vector<int> route(mat.size());
    std::iota(route.begin(), route.end(), 0);

    std::vector<std::vector<int>> visited = get_visited_places(route
        , ants);

    for (const auto &ant : std::views::iota(0, ants)) {
        while (visited[ant].size() != mat.size()) {
        std::vector<double> pk =
            find_possibilities(pheromones, visibility, visited, ant,
                mat);
```

```
        int chosen_place = choose_next(pk);
        visited[ant].push_back(chosen_place);
        }

        visited[ant].push_back(visited[ant][0]);

        double cur_length = calc_length(visited[ant], mat);
        if (cur_length < min_length) {
        min_length = cur_length;
        best_way = visited[ant];
        }
    }

    pheromones = update_pheromones(visited, pheromones, q, mat);
    pheromones = update_pheromones_elite(pheromones, q, best_way,
       mat);
    }

    return {min_length, best_way};
}

double AntPathFinder::calc_q(const Matrix &mat) {
    double q = 0;
    int count = 0;

    for (const auto &i : std::views::iota(0ull, mat.size()))
    for (const auto &j :
            std::views::iota(0ull, mat.size()) |
                std::views::filter([&i](int j) { return i != j; }))
                   {
        q += mat[i][j];
        count++;
    }

    return q / count;
}

AntPathFinder::Matrix AntPathFinder::get_pheromones(const Matrix &
   mat) {
    double min_phero = 1.0;
```

```
    return Matrix(mat.size(), std::vector<double>(mat.size(),
        min_phero));
}

AntPathFinder::Matrix AntPathFinder::get_visibility(const Matrix &
    mat) {
    Matrix visibility(mat.size(), std::vector<double>(mat.size(),
        0.));

    for (const auto &i : std::views::iota(0ull, mat.size()))
    for (const auto &j : std::views::iota(0ull, mat.size()) |
                                std::views::filter([&i](int j) {
                                    return i != j; }))
        visibility[i][j] = 1.0 / mat[i][j];

    return visibility;
}

std::vector<std::vector<int>>
AntPathFinder::get_visited_places(const std::vector<int> &route, int
    ants) {
    std::vector<std::vector<int>> visited(ants);
    for (const auto &ant : std::views::iota(0, ants))
    visited[ant].push_back(route[ant]);

    return visited;
}

double AntPathFinder::calc_length(const std::vector<int> &route,
                                  const Matrix &mat) {
    double length = 0;
    for (size_t way_len = 1; way_len < route.size(); ++way_len) {
    length += mat[route[way_len - 1]][route[way_len]];
    }
    return length;
}

AntPathFinder::Matrix
AntPathFinder::update_pheromones(const std::vector<std::vector<int>>
    &visited,
```

```
                                        Matrix &pheromones, double q,
                                        const Matrix &mat) {
    for (const auto &i : std::views::iota(0ull, mat.size()))
    for (const auto &j : std::views::iota(0ull, mat.size())) {
        double delta_pheromones = 0;
        for (int ant = 0; ant < mat.size(); ++ant) {
        double length = calc_length(visited[ant], mat);
        delta_pheromones += q / length;
        }
        pheromones[i][j] *= (1 - m_k_evaporation);
        pheromones[i][j] += delta_pheromones;

        if (pheromones[i][j] < MIN_PHEROMONE) {
        pheromones[i][j] = MIN_PHEROMONE;
        }
    }

    return pheromones;
}

AntPathFinder::Matrix
AntPathFinder::update_pheromones_elite(Matrix &pheromones, double q,
                                       const std::vector<int> &
                                          best_way,
                                       const Matrix &mat) {
    double length = calc_length(best_way, mat);
    for (const auto &i : std::views::iota(0ull, mat.size()))
    for (const auto &j : std::views::iota(0ull, mat.size())) {
        double delta_pheromones = m_e * q / length;
        pheromones[i][j] += delta_pheromones;
    }

    return pheromones;
}

std::vector<double> AntPathFinder::find_possibilities(
    const Matrix &pheromones, const Matrix &visibility,
    const std::vector<std::vector<int>> &visited, int ant, const
       Matrix &mat) {
    std::vector<double> pk(mat.size(), 0.);
```

```
    int ant_place = visited[ant].back();

    for (const auto &place : std::views::iota(0ull, mat.size())) {
    if (std::find(visited[ant].begin(), visited[ant].end(), place)
      ==
        visited[ant].end()) {
        pk[place] = std::pow(pheromones[ant_place][place], m_alpha)
          *
                    std::pow(visibility[ant_place][place], m_beta);
    }
    }

    if (visited[ant].size() == mat.size()) {
    int start_place = visited[ant][0];
    pk[start_place] = std::pow(pheromones[ant_place][start_place],
       m_alpha) *
                        std::pow(visibility[ant_place][start_place],
                            m_beta);
    }

    double sum_pk = std::accumulate(pk.begin(), pk.end(), 0.0);
    for (const auto &place : std::views::iota(0ull, mat.size()))
    pk[place] /= sum_pk;

    return pk;
}

int AntPathFinder::choose_next(const std::vector<double> &pk) {
    double possibility = static_cast<double>(rand()) / RAND_MAX;
    double choice = 0;
    int chosen_place = 0;

    while ((choice < possibility) && (chosen_place < pk.size())) {
    choice += pk[chosen_place];
    chosen_place++;
    }

    return chosen_place - 1;
}
```

3.4 Функциональные тесты

В таблице 1 приведены функциональные тесты для алгоритмов. Все тесты пройдены успешно.

В таблице используются следующие обозначения.

1) Длина пути, полученная методом полного перебора — МПП.
2) Длина пути, полученная методом на основе муравьиного алгоритма — ММА.
3) Оптимальный результат (длина пути) — ОР.

Таблица 1 — Функциональные тесты

Матрица смежности	ОР	МПП	ММА
$\begin{pmatrix} 0 & 3 & 8 & 7 & 10 \\ 3 & 0 & 7 & 5 & 4 \\ 8 & 7 & 0 & 5 & 5 \\ 7 & 5 & 5 & 0 & 3 \\ 10 & 4 & 5 & 3 & 0 \end{pmatrix}$	23	23	23
$\begin{pmatrix} 0 & 3 & 4 \\ 3 & 0 & 5 \\ 4 & 5 & 0 \end{pmatrix}$	12	12	12
$\begin{pmatrix} 0 & 27 & 12 & 21 \\ 27 & 0 & 16 & 20 \\ 12 & 16 & 0 & 11 \\ 21 & 20 & 11 & 0 \end{pmatrix}$	69	69	69

Получение оптимального результата для муравьиного алгоритма не гарантировано, поэтому тест считается пройденным реализацией муравьиного алгоритма, если результат входит в лучшие 10%, то есть ОР $*$ 1.1.

4 Исследовательская часть

4.1 Технические характеристики

— процессор: 13th Gen Intel(R) Core(TM) i5-13500H (16 ядер) @ 4.70ГГц;

— оперативная память: 16 ГБайт;

— операционная система: EndeavourOS x86_64.

При замерах ноутбук был включен в сеть и был нагружен только системными приложениями.

4.2 Результаты проводимых исследований

В ходе исследования сравнивается время, необходимое на поиск пути.

Результаты исследования изображены на рисунке 8.

В процессе исследования для алгоритма муравьиного использовался промежуток времени: 200 дней.

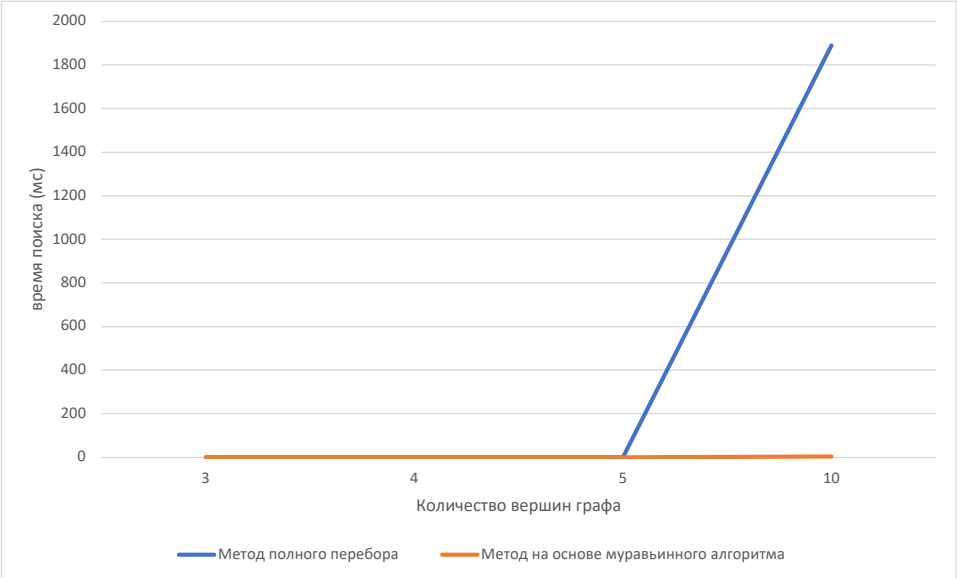

Рисунок 8 — Сравнение временных затрат на поиск пути

4.3 Параметризация

Параметризация проведена на классе данных состоящем из трех графов — 6–8. Алгоритм запущен для набора значений $\alpha, \rho \in (0, 1), days = \{1, 3, 5, 10, 50, 100, 300, 500\}, e = \{0.5, 1, 2\}$.

Полученная таблица значений в ходе параметризации состоит из колонок:

— α — коэффициент жадности;

— ρ — коэффициент испарения;

— *days* — количество дней жизни колонии муравьев;

— *Result* — эталонный результат, полученный методом полного перебора для проведения данного исследования;

— *Mistake* — показатель качества решения, полученный путем разности эталонного решения и решения методом на основе муравьиного алгоритма.

Цель параметризации — определить комбинацию параметров, которые позволяют решить задачу наилучшим образом для выбранного графа. Качество решения зависит от количества дней и погрешности измерений.

4.3 Класс данных

Класс данных состоит из 3 матриц смежности размером 10 элементов (см. таблицы 6–8).

$$K_1 = \begin{pmatrix} 0 & 45 & 0 & 2 & 39 & 3 & 46 & 87 & 95 & 39 \\ 45 & 0 & 71 & 54 & 42 & 69 & 76 & 27 & 10 & 66 \\ 0 & 71 & 0 & 51 & 10 & 3 & 9 & 42 & 22 & 72 \\ 2 & 54 & 51 & 0 & 98 & 6 & 6 & 45 & 62 & 78 \\ 39 & 42 & 10 & 98 & 0 & 21 & 78 & 67 & 65 & 64 \\ 3 & 69 & 3 & 6 & 21 & 0 & 64 & 37 & 44 & 48 \\ 46 & 76 & 9 & 6 & 78 & 64 & 0 & 55 & 65 & 88 \\ 87 & 27 & 42 & 45 & 67 & 37 & 55 & 0 & 23 & 70 \\ 95 & 10 & 22 & 62 & 65 & 44 & 65 & 23 & 0 & 78 \\ 39 & 66 & 72 & 78 & 64 & 48 & 88 & 70 & 78 & 0 \end{pmatrix} \quad (6)$$

$$K_2 = \begin{pmatrix} 0 & 85 & 18 & 25 & 42 & 97 & 77 & 80 & 69 & 2 \\ 85 & 0 & 18 & 88 & 1 & 26 & 55 & 11 & 19 & 28 \\ 18 & 18 & 0 & 51 & 44 & 36 & 49 & 54 & 83 & 61 \\ 25 & 88 & 51 & 0 & 27 & 22 & 45 & 71 & 7 & 11 \\ 42 & 1 & 44 & 27 & 0 & 96 & 80 & 53 & 16 & 5 \\ 97 & 26 & 36 & 22 & 96 & 0 & 89 & 7 & 9 & 16 \\ 77 & 55 & 49 & 45 & 80 & 89 & 0 & 68 & 95 & 12 \\ 80 & 11 & 54 & 71 & 53 & 7 & 68 & 0 & 91 & 3 \\ 69 & 19 & 83 & 7 & 16 & 9 & 95 & 91 & 0 & 48 \\ 2 & 28 & 61 & 11 & 5 & 16 & 12 & 3 & 48 & 0 \end{pmatrix} \tag{7}$$

$$K_3 = \begin{pmatrix} 0 & 20 & 45 & 89 & 91 & 99 & 91 & 58 & 95 & 44 \\ 20 & 0 & 29 & 62 & 59 & 8 & 83 & 38 & 35 & 21 \\ 45 & 29 & 0 & 11 & 58 & 1 & 2 & 3 & 5 & 47 \\ 89 & 62 & 11 & 0 & 57 & 23 & 82 & 48 & 9 & 71 \\ 91 & 59 & 58 & 57 & 0 & 53 & 72 & 81 & 86 & 12 \\ 99 & 8 & 1 & 23 & 53 & 0 & 16 & 14 & 13 & 30 \\ 91 & 83 & 2 & 82 & 72 & 16 & 0 & 7 & 10 & 25 \\ 58 & 38 & 3 & 48 & 81 & 14 & 7 & 0 & 9 & 46 \\ 95 & 35 & 5 & 9 & 86 & 13 & 10 & 9 & 0 & 54 \\ 44 & 21 & 47 & 71 & 12 & 30 & 25 & 46 & 54 & 0 \end{pmatrix} \tag{8}$$

Результаты параметризации представлены в приложении А.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы были достигнуты, а именно были рассмотрены методы решения задачи коммивояжера, такие как метод полного перебора и муравьиный алгоритм.

Для достижения цели были выполнены следующие задачи:

1) были рассмотрены методы решения задачи коммивояжера;
2) были разработаны рассмотренные методы;
3) были реализованы разработанные алгоритмы;
4) был проведён анализ временных затрат реализованных алгоритмов;
5) была проведена параметризация для муравьиного (роевого) алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The Traveling Salesman Problem: A Computational Study / D.L. Applegate, R.E. Bixby, V. Chvátal [и др.]. Princeton Series in Applied Mathematics. Princeton University Press, 2011. URL: https://books.google.ru/books?id=zfIm94nNqPoC.

2. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ : учебное пособие. — Москва : ФИЗМАТЛИТ, 2008.

3. Штовба С. Д. Муравьиные алгоритмы. [Электронный ресурс]. — Режим доступа: `https://masters.donntu.ru/2010/fknt/chorniy/library/article7.htm` (дата обращения: 01.12.2024).

4. ISO International Standard ISO/IEC 14882:2020(E) [Working Draft] – Programming Language C++.

Приложение А