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 Методы решения задачи коммивояжера

Студент Шматко К. М.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут рассмотрены методы решения задачи коммивояжера.

Задача коммивояжёра — одна из задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город [1]. Эта задача имеет различные методы решения, и каждый из них имеет свои преимущества и недостатки.

Целью данной лабораторной работы является исследование методов решения задачи коммивояжера.

Для достижения поставленной цели необходимо выполнить следующие задачи.

1) Описать методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма.

2) Создать программное обеспечение, позволяющее решить задачу коммивояжера исследуемыми методами.

3) Замерить время реализации.

4) Провести анализ затрат работы программы по времени, провести параметризацию муравьиного алгоритма.

Выданный индивидуальный вариант для выполнения лабораторной работы:

— неориентированный граф;

— без элитных муравьев;

— гамильтонов цикл;

— города России от Калининграда до Владивостока.

1 Аналитическая часть

В этом разделе будет представлена информация о задаче коммивояжера, а также о способах её решения — методе полного перебора и методе на основе муравьиного алгоритма.

1.1 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них [1]. Смысл перебора состоит в том, что мы перебираем все варианты объезда городов и выбираем оптимальный. Однако при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (сложность алгоритма равна $n!$).

Алгоритм полного перебора обеспечивает точное решение задачи, но с увеличением числа городов значительно возрастают временные затраты на выполнение.

1.2 Метод на основе муравьиного алгоритма

Муравьиный алгоритм — метод решения задачи оптимизации, основанный на принципе поведения колонии муравьев [1].

Муравьи следуют своим органам чувств, оставляя феромоны на своем пути для навигации других муравьев. При большом числе муравьев наибольшее количество феромонов задерживается на самом популярном пути, и это может быть связано с длинами ребер.

Идея заключается в том, что отдельный муравей обладает ограниченными возможностями, поскольку он способен выполнять только простые задачи. Однако, при большом количестве таких муравьев они могут действовать как автономные вычислительные единицы. Муравьи взаимодействуют непрямым обменом информацией через окружающую среду при помощи феромонов.

Пусть муравей имеет следующие характеристики:

1) зрение — способность определить длину ребра;

2) память — способность запомнить пройденный маршрут;

3) обоняние — способность чуять феромон.

Также введем функцию (1.1), характеризующую привлекательность ребра, которая определяется зрением.

$$\eta_{ij} = 1/D_{ij}, \tag{1.1}$$

где D_{ij} — расстояние от текущего пункта i до заданного пункта j.

Дополнительно понадобится формула вычисления вероятности перехода в заданную точку (1.2).

$$p_{k,ij} = \begin{cases} \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum_{q \notin J_k} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, j \notin J_k \\ 0, j \in J_k \end{cases} \tag{1.2}$$

где a — параметр влияния длины пути, b — параметр влияния феромона, τ_{ij} — количество феромонов на ребре ij, η_{ij} — привлекательность ребра ij, J_k — список посещенных за текущий день городов.

После завершения движения всех муравьев (ночью, перед наступлением следующего дня), феромон обновляется по формуле (1.3).

$$\tau_{ij}(t+1) = \tau_{ij}(t) \cdot (1-p) + \Delta\tau_{ij}(t). \tag{1.3}$$

При этом

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{N} \Delta\tau_{ij}^{k}(t), \tag{1.4}$$

где

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q/L_k, \text{ребро посещено муравьем } k \text{ в текущий день } t, \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

Поскольку вероятность (1.2) перехода в заданную точку не должна быть равна нулю, необходимо обеспечить неравенство $\tau_{ij}(t)$ нулю посред-

ством введения дополнительного минимально возможного значения феромона τ_{min} и в случае, если $\tau_{ij}(t+1)$ принимает значение, меньшее τ_{min}, откатывать феромон до этой величины.

Путь выбирается по следующей схеме.

1) Каждый муравей имеет список запретов — список уже посещенных городов (вершин графа).

2) Муравьиное зрение отвечает за эвристическое желание посетить вершину.

3) Муравьиное обоняние отвечает за ощущение феромона на определенном пути (ребре). При этом количество феромона на пути (ребре) в день t обозначается как $\tau_{i,j}(t)$.

4) После прохождения определенного ребра муравей откладывает на нем некоторое количество феромона, которое показывает оптимальность сделанного выбора, это количество вычисляется по формуле (1.5).

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также способы её решения — полный перебор и муравьиный алгоритм.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора путей. На рисунках 2.2 и 2.3 представлена схема алгоритма муравьиного алгоритма. На рисунках 2.4 и 2.6 представлены алгоритмы дополнительных функций.

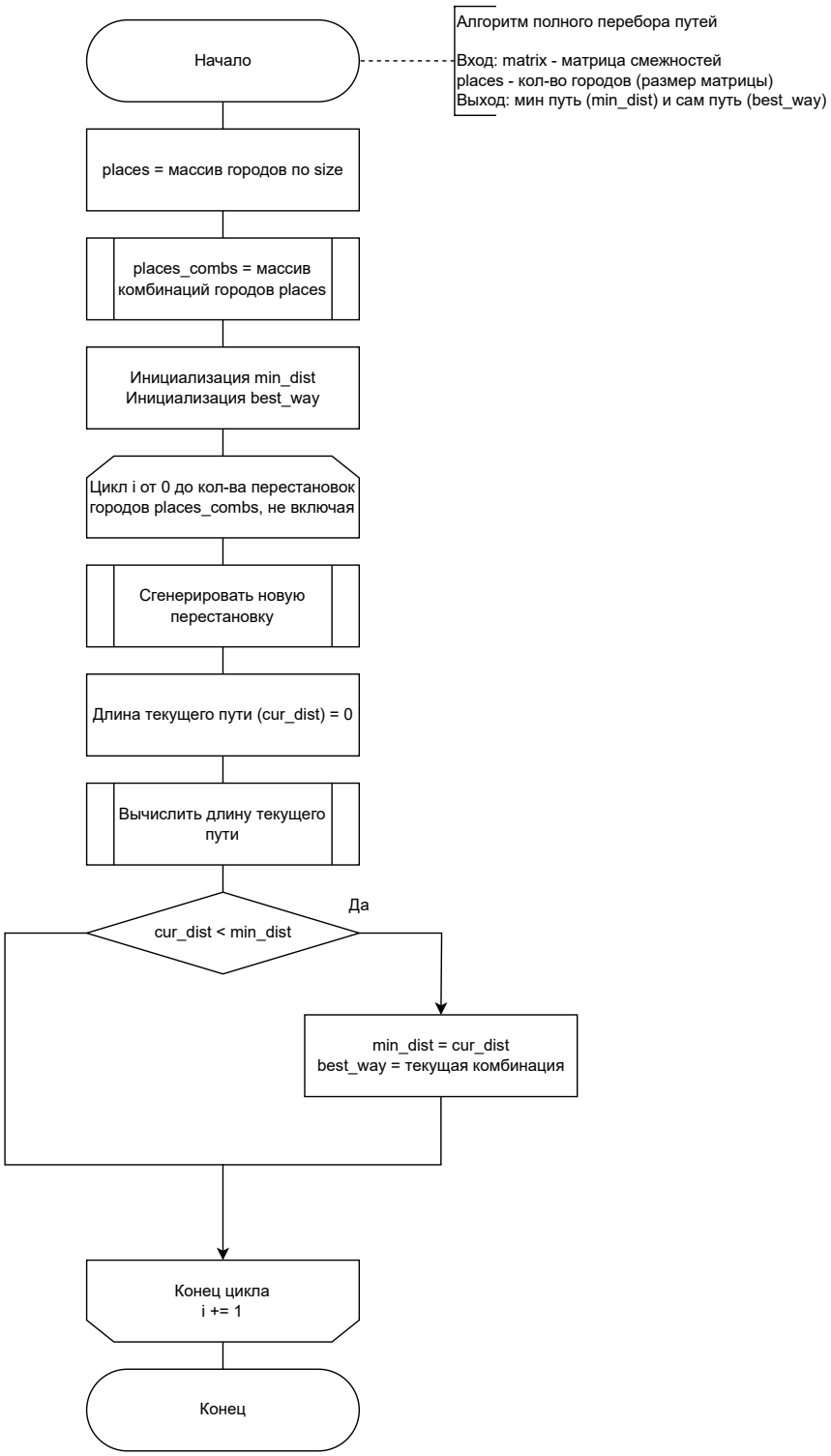

Рисунок 2.1 – Схема алгоритма полного перебора путей

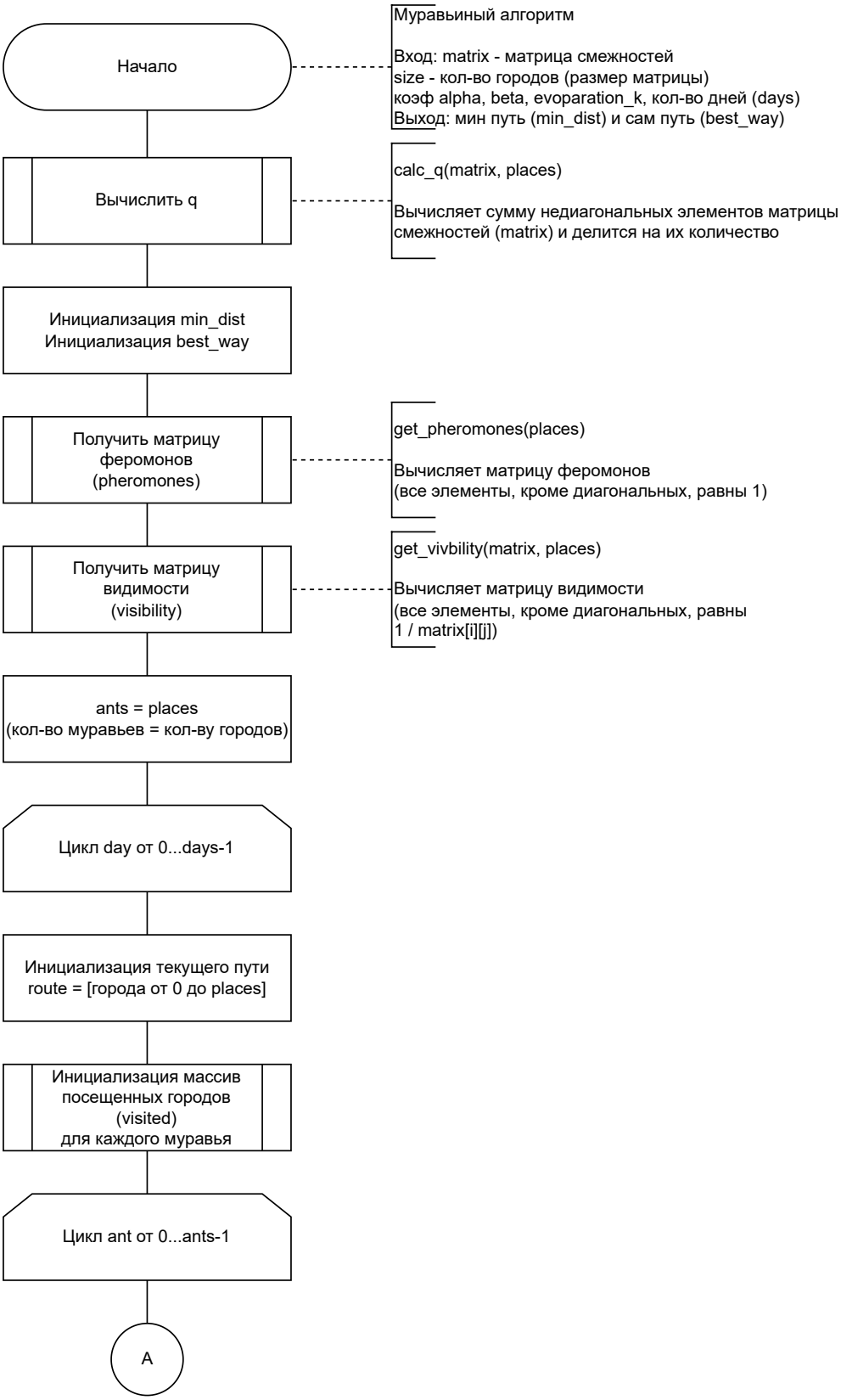

Рисунок 2.2 – Схема муравьиного алгоритма (часть 1)

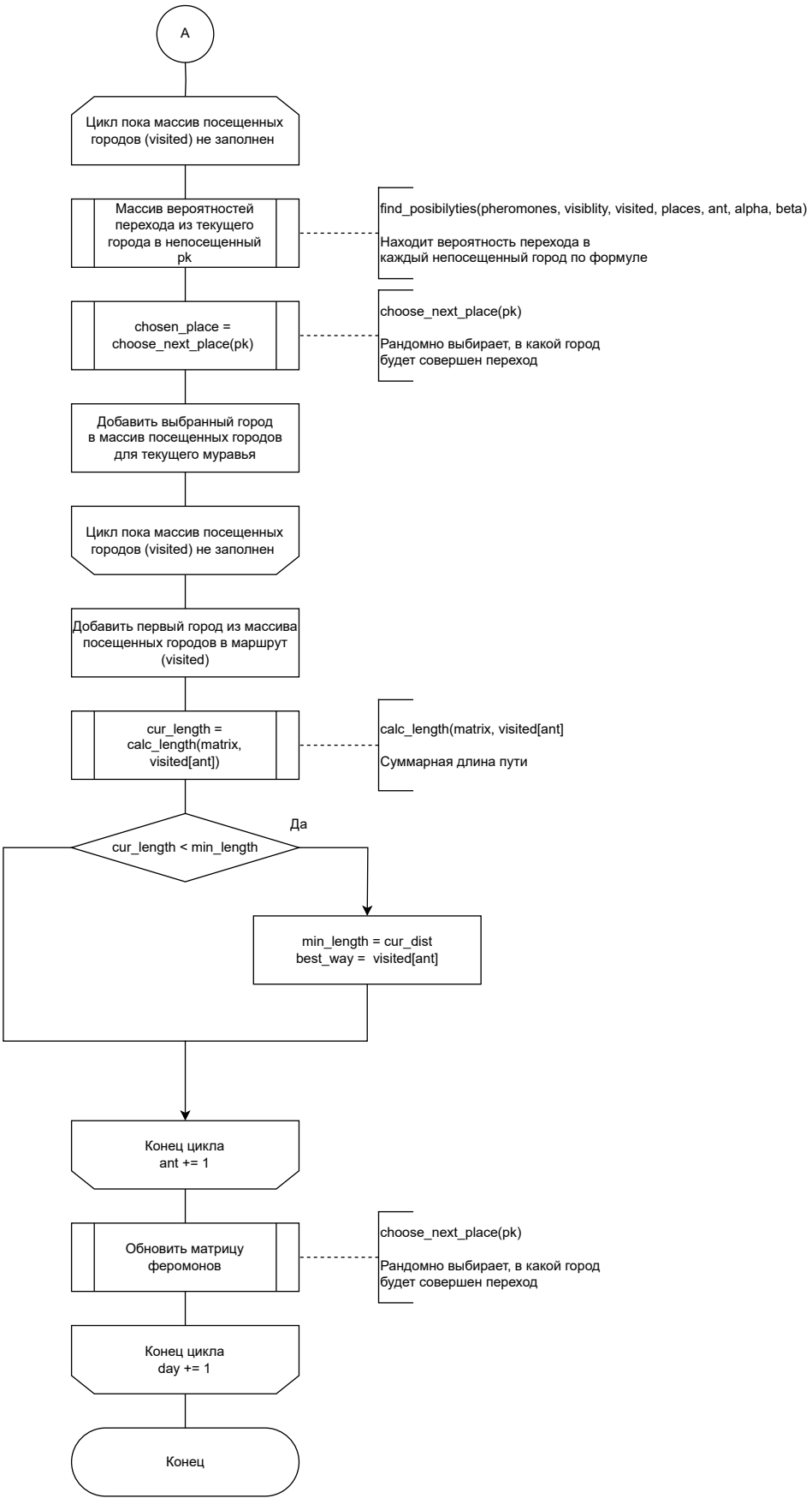

Рисунок 2.3 – Схема муравьиного алгоритма (часть 2)

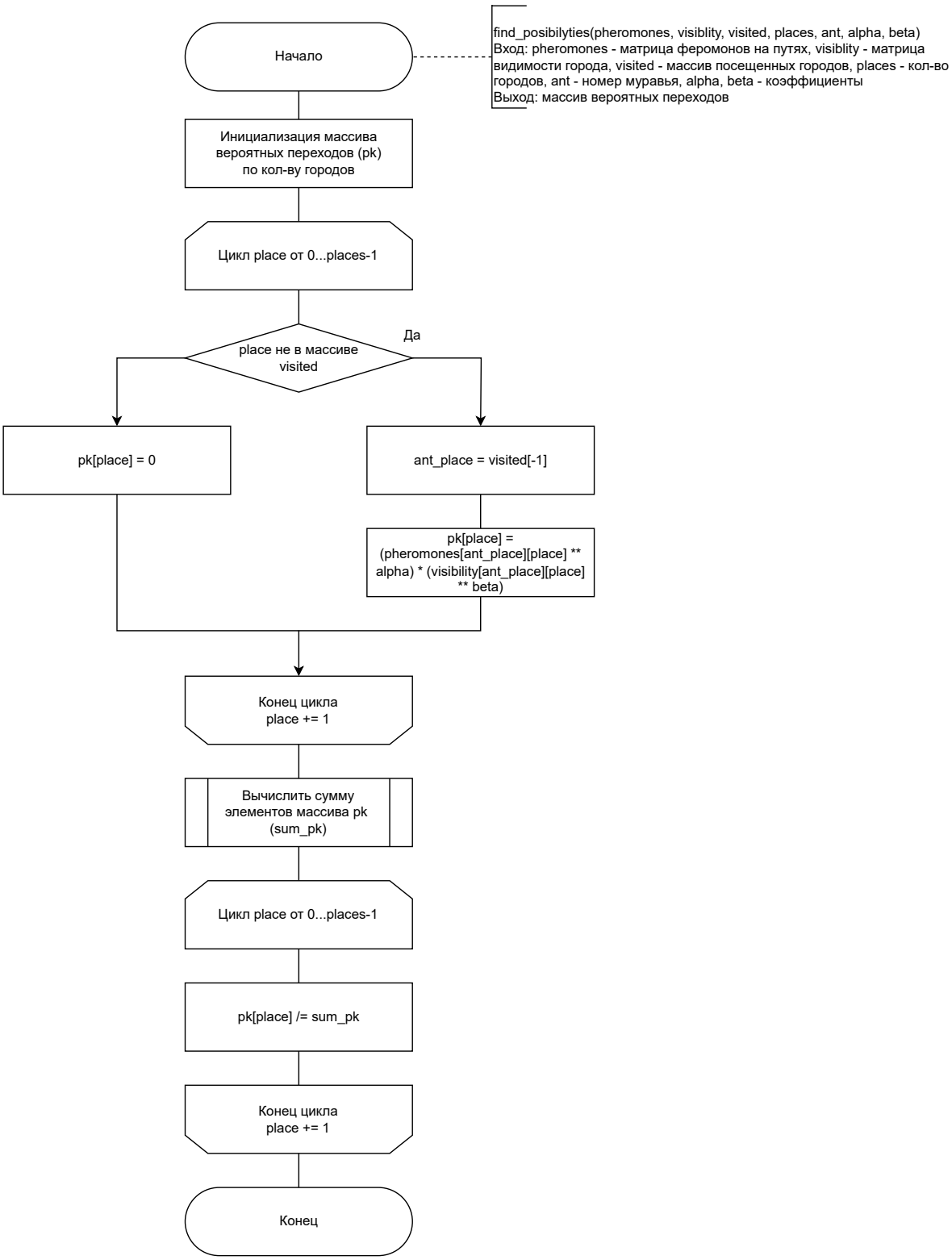

Рисунок 2.4 – Схема алгоритма нахождения массива вероятностей переходов в непосещенные города

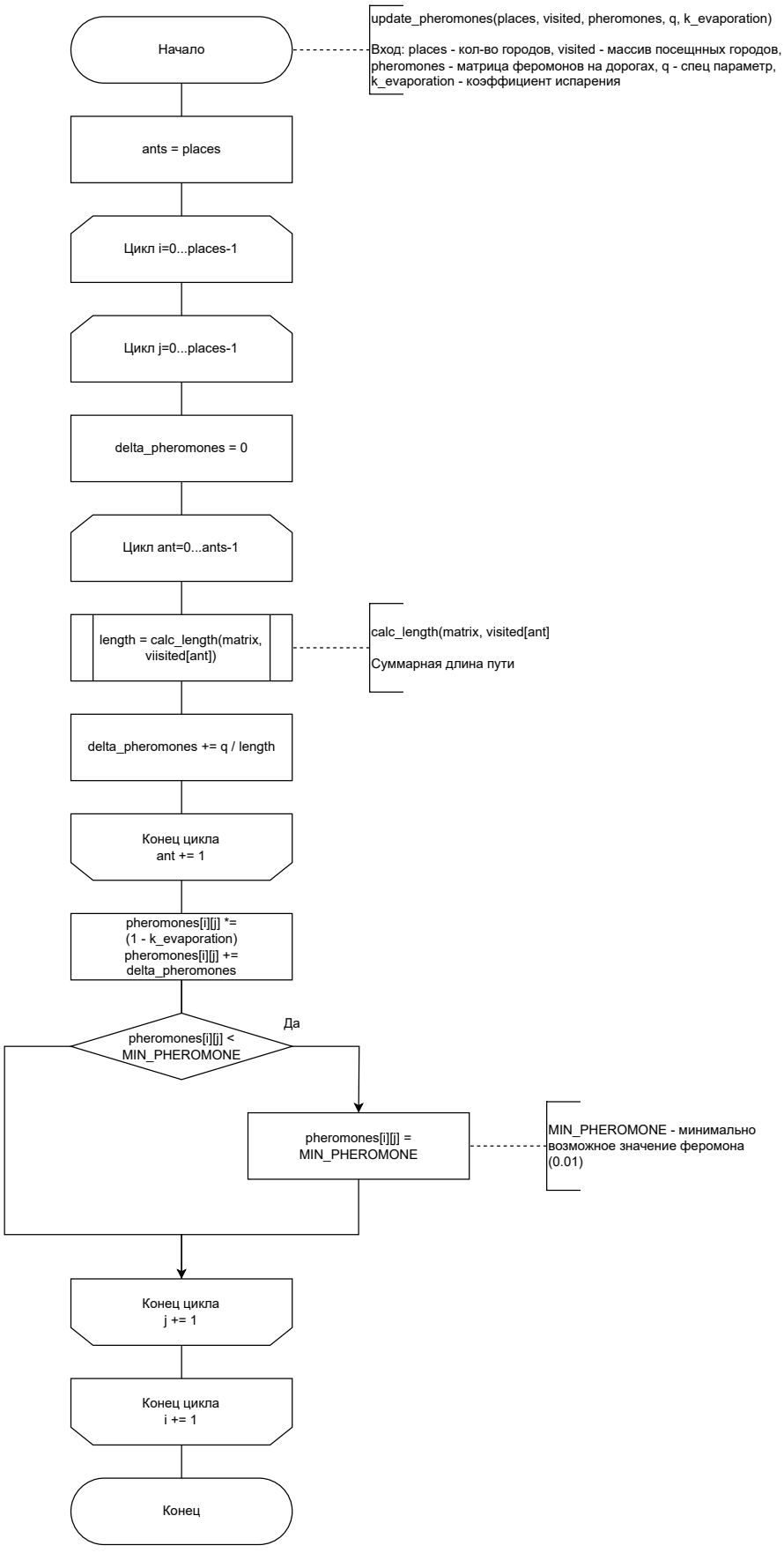

Рисунок 2.5 – Схема алгоритма обновления матрицы феромонов

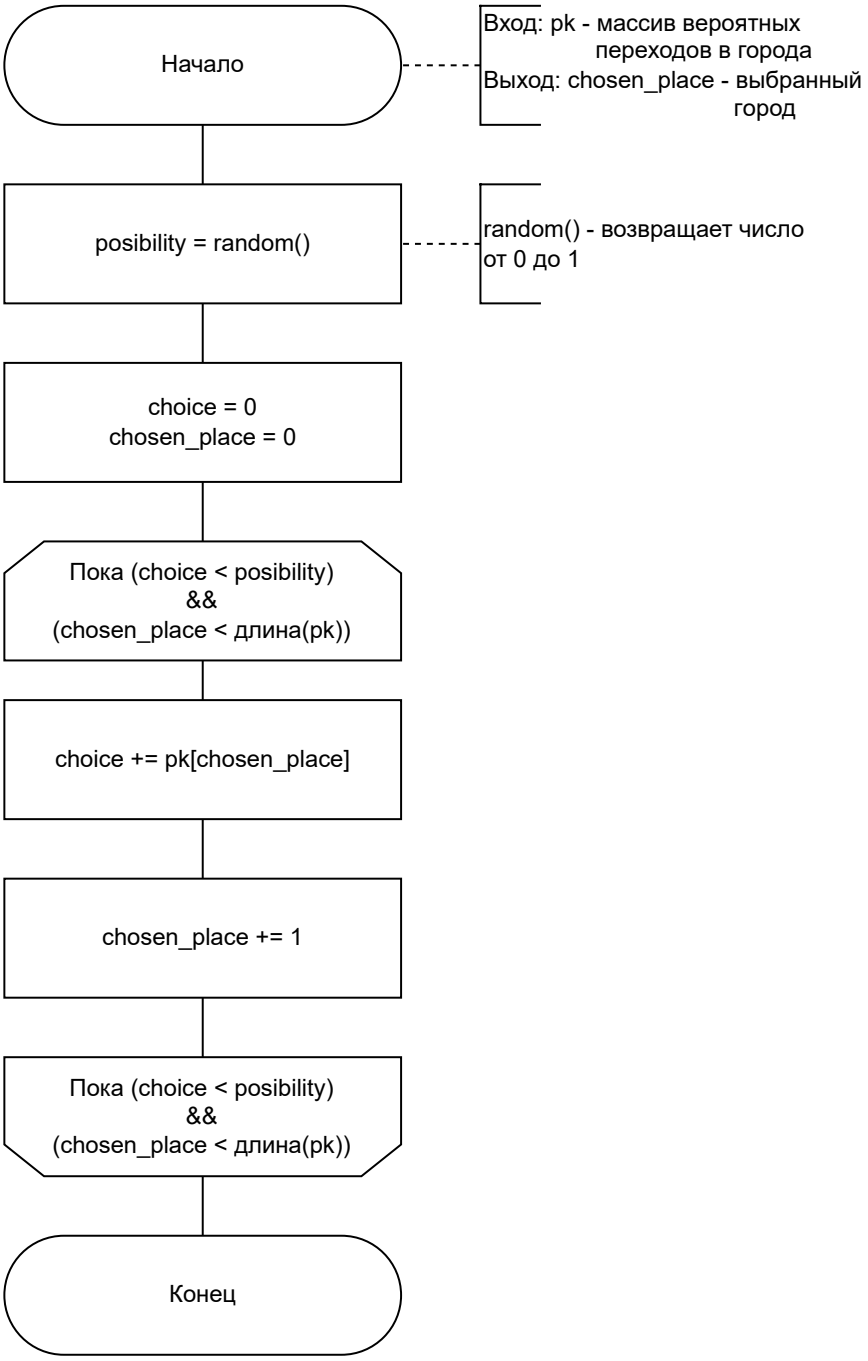

Рисунок 2.6 – Схема алгоритма выбора следующего города

2.2 Модель вычислений

Для вычисления трудоёмкости данных алгоритмов необходимо ввести модель вычислений.

Обозначим трудоёмкость как f_a, где a — индекс, указывающий операцию, блок кода или оператор, для которого вычисляется трудоёмкость.

Определим трудоёмкость базовых операций как:

$$\begin{array}{rrrr} f_{+} = 1 & f_{-} = 1 & f_{+=} = 1 & f_{-=} = 1 \\ f_{:=} = 1 & f_{<<} = 1 & f_{>>} = 1 & f_{[]} = 1 \\ f_{++} = 1 & f_{--} = 1 & f_{>} = 1 & f_{<} = 1 \\ f_{>=} = 1 & f_{<=} = 1 & f_{!=} = 1 & f_{==} = 1 \\ f_{.} = 2 & f_{/} = 2 & f_{\%} = 2 & \end{array} \tag{2.1}$$

Определим трудоёмкость вызова функции как 0.

Определим трудоёмкость условия как

$$f_{if} = f_{cc} + \begin{cases} \min(f_1, f_2), & \text{в лучшем случае,} \\ \max(f_1, f_2), & \text{в худшем случае,} \end{cases} \tag{2.2}$$

где приняты следующие обозначения:

— f_{cc} — трудоёмкость вычисления условия;

— f_1 — трудоёмкость блока после if;

— f_2 — трудоёмкость блока после $else$.

Определим трудоёмкость цикла как

$$f_{loop} = f_{init} + f_{first-cmp} + n \cdot (f_{body} + f_{inc} + f_{cmp}), \tag{2.3}$$

где приняты следующие обозначения:

— f_{init} — трудоёмкость инициализации;

— $f_{first-cmp}$ — трудоёмкость первого сравнения;

— f_{body} — трудоёмкость тела цикла;

— n — количество итераций цикла;

— f_{inc} — трудоёмкость изменения индекса;

— f_{cmp} — трудоёмкость сравнения.

2.2.1 Трудоёмкость алгоритма полного перебора

Трудоемкость алгоритма полного перебора решения задачи коммивояжера состоит из следующих составляющих.

Трудоёмкость внешнего цикла по $i \in [0..N! - 1]$:

$$f_{loop} = 2 + N! \cdot (2 + f_{body}) \tag{2.4}$$

Далее рассмотрена трудоёмкость для тела цикла в зависимости от результата проверки условия на существование заданного пути в графе:

$$f_{body} = f_{if} + \begin{cases} f_{true}, & \text{если путь существует,} \\ 0, & \text{иначе.} \end{cases} \tag{2.5}$$

Трудоёмкость проверки условия:

$$f_{if} = 2 + N \cdot (2 + 8). \tag{2.6}$$

Трудоёмкость действий при выполнении условия:

$$f_{true} = 1 + 2 + N \cdot (2 + 10). \tag{2.7}$$

Тогда трудоемкость алгоритма полного перебора решения задачи коммивояжера для лучшего случая (путь в графе не существует) составит

$$f_{brute\_force} = 2 + N! \cdot (2 + 2 + N \cdot (2 + 8)) = 2 + 4N! + 10N \cdot N! = O(N!), \tag{2.8}$$

а трудоемкость для худшего случая (путь в графе существует):

$$\begin{aligned} f_{brute\_force} = 2 + N! \cdot (2 + 2 + N \cdot (2 + 8) + 1 + 2 + N \cdot (2 + 10)) = \\ = 2 + 7N! + 22N \cdot N! = O(N!). \end{aligned} \tag{2.9}$$

2.2.2 Трудоёмкость муравьиного алгоритма

Введём некоторые обозначения: N — размер графа (равен размеру муравьиной колонии), T — количество дней жизни колонии.

Далее приведён расчёт трудоёмкости муравьиного алгоритма для решения задачи коммивояжёра.

Трудоёмкость внешней функции для муравьиного алгоритма:

$$f_{outer} = f_{ph} + f_{vis} + f_q + 2 + f_{loop-t} + 5. \tag{2.10}$$

Трудоёмкость функции инициализации матрицы феромона:

$$f_{ph} = 1+2+N\cdot(2+3)+2+N\cdot(2+2+N\cdot(2+3)) = 5+7\cdot N+5\cdot N^2. \tag{2.11}$$

Трудоёмкость функции инициализации матрицы видимости:

$$f_{vis} = 1+2+N\cdot(2+3)+2+N\cdot(2+2+N\cdot(2+3)) = 5+7\cdot N+5\cdot N^2. \tag{2.12}$$

Трудоёмкость функции инициализации коэффициента пропорциональности феромона:

$$f_q = 1 + 2 + N\cdot(2+2+N\cdot(2+3)) + 1 = 4 + 4\cdot N + 5\cdot N^2. \tag{2.13}$$

Трудоёмкость цикла по дням:

$$f_{loop-t} = 2 + T\cdot(f_{col} + floop - n + f_{vap} + f_{inc} + f_{cor}). \tag{2.14}$$

Трудоёмкость функции создания колонии:

$$f_{col} = 1 + 2 + N\cdot(2+7) = 3 + 9\cdot N. \tag{2.15}$$

Трудоёмкость функции испарения феромона:

$$f_{vap} = 2 + N\cdot(2+2+N\cdot(2+7)) = 2 + 4\cdot N + 9\cdot N^2. \tag{2.16}$$

Трудоёмкость функции прироста феромона:

$$f_{inc} = 2 + N \cdot (2 + 2 + (N - 1) \cdot (2 + 9)) = 2 - 7 \cdot N + 11 \cdot N^2. \quad (2.17)$$

Трудоёмкость функции коррекции феромона в лучшем случае (не нужна коррекцция) составит:

$$f_{cor} = 2 + N \cdot (2 + 2 + N \cdot 3) = 2 + 4 \cdot N + 3 \cdot N^2, \quad (2.18)$$

а в худшем (нужна коррекция всех):

$$f_{cor} = 2 + N \cdot (2 + 2 + N \cdot 6) = 2 + 4 \cdot N + 6 \cdot N^2. \quad (2.19)$$

Трудоёмкость цикла по количеству муравьёв в колонии:

$$floop - n = 2 + N \cdot (2 + f_{route} + 8). \quad (2.20)$$

Подставив в формулу 2.10 для лучшего случая получается

$$f_{outer} = 23 + 18 \cdot N + 15 \cdot N^2 + 11 \cdot T + 25 \cdot N \cdot T + 14 \cdot N^2 \cdot T + 22 \cdot N^3 \cdot T, \quad (2.21)$$

а для худшего

$$f_{outer} = 23 + 18 \cdot N + 15 \cdot N^2 + 11 \cdot T + 44 \cdot N \cdot T - 1.5 \cdot N^2 \cdot T + 25.5 \cdot N^3 \cdot T. \quad (2.22)$$

Такую трудоёмкость можно оценить как $O(N^3 \cdot T)$.

Вывод

В данном разделе были рассмотрены алгоритм полного перебора и муравьиный алгоритм.

3 Технологическая часть

В данном разделе будут описаны требования к программному обеспечению, средства реализации, выбранные типы данных, листинг кода и функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— получает на вход матрицу смежности, для которой можно будет выбрать один из алгоритмов поиска оптимальных путей (полным перебором или муравьиным алгоритмом);

— позволяет пользователю определять коэффициенты и количество дней для муравьиного алгоритма;

— вычисляет минимальную сумму пути, а также сам путь, используя один из алгоритмов.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык JavaScript [2]. Данный выбор обусловлен наличием у языка встроенной функции $process.cpuUsage()$ для измерения процессорного времени. Это позволяет удовлетворить требованиям для выполнения лабораторной работы.

Время выполнения программы было замерено с использованием функции *process.cpuUsage()* из программной платформы Node.js [3].

3.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— размер матрицы смежности — целое число;

— коэффициенты α, β, ρ — действительные числа;

— матрица смежности — матрица целых чисел.

3.4 Сведения о модулях программы

Данная программа разбита на следующие модули.

— `main.js` — файл, содержащий точку входа в программу, из которой происходит запуск работы алгоритмов.

— `algorithms.js` — файл, содержащий функции реализации всех алгоритмов.

— `test.js` — файл содержит функции, замеряющие процессорное время алгоритмов и проводящий параметризацию.

— `file.js` — файл, содержащий вспомогательные функции.

3.5 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора путей, в листингах 3.2–3.6 представлена реализация муравьиного алгоритма и дополнительные к нему функции.

Листинг 3.1 – Реализаиция алгоритма полного перебора

```
function fullCombinationAlg(matrix, size) {
    const places = Array.from({ length: size }, (_, i) => i);
    const placesCombinations = [];

    for (const combination of itertools.permutations(places)) {
        const combArr = Array.from(combination);
        placesCombinations.push(combArr);
    }

    let minDist = Infinity;
    let bestWay;

    for (let i = 0; i < placesCombinations.length; i++) {
        placesCombinations[i].push(placesCombinations[i][0]);
        let curDist = 0;

        for (let j = 0; j < size; j++) {
            const startCity = placesCombinations[i][j];
            const endCity = placesCombinations[i][j + 1];
            curDist += matrix[startCity][endCity];
        }

        if (curDist < minDist) {
            minDist = curDist;
            bestWay = placesCombinations[i];
        }
    }

    console.log(minDist);
    return [minDist, bestWay];
}
```

Листинг 3.2 – Реализаиция муравьиного алгоритма

```
function antAlgorithm(matrix, places, alpha, beta,
    k_evaporation, days) {
    const q = calcQ(matrix, places);
    let bestWay = [];
    let minDist = Infinity;
    let pheromones = calcPheromones(places);
    const visibility = calcVisibility(matrix, places);
    const ants = places;

    for (let day = 0; day < days; day++) {
        const route = Array.from({ length: places }, (_, i) =>
            i);
        const visited = calcVisitedPlaces(route, ants);

        for (let ant = 0; ant < ants; ant++) {
            while (visited[ant].length !== ants) {
                const pk = findWays(pheromones, visibility,
                    visited, places, ant, alpha, beta);
                const chosenPlace =
                    chooseNextPlaceByPosibility(pk);
                visited[ant].push(chosenPlace - 1);
            }

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

            const curLength = calcLength(matrix, visited[ant]);

            if (curLength < minDist) {
                minDist = curLength;
                bestWay = visited[ant];
            }
        }

        pheromones = updatePheromones(matrix, places, visited,
            pheromones, q, k_evaporation);
    }
    console.log(minDist);
    return [minDist, bestWay];
}
```

Листинг 3.3 – Реализация алгоритма нахождения массива вероятностей переходов в непосещенные города

```
function findWays(pheromones, visibility, visited, places, ant,
    alpha, beta) {
    const pk = Array(places).fill(0);

    for (let place = 0; place < places; place++) {
        if (!visited[ant].includes(place)) {
            const ant_place = visited[ant][visited[ant].length
                - 1];
            pk[place] = Math.pow(pheromones[ant_place][place],
                alpha) * Math.pow(visibility[ant_place][place],
                beta);
        } else {
            pk[place] = 0;
        }
    }

    const sum_pk = pk.reduce((sum, val) => sum + val, 0);

    for (let place = 0; place < places; place++) {
        pk[place] /= sum_pk;
    }

    return pk;
}
```

Листинг 3.4 – Реализация алгоритма выбора следующего города

```
function chooseNextPlaceByPosibility(pk) {
    const posibility = Math.random();
    let choice = 0;
    let chosenPlace = 0;

    while (choice < posibility && chosenPlace < pk.length) {
        choice += pk[chosenPlace];
        chosenPlace++;
    }

    return chosenPlace;
}
```

Листинг 3.5 – Реализация алгоритма вычисления феромона

```
1     function calcPheromones(size) {
2         const min_phero = 1;
3         const pheromones = Array.from({ length: size }, () =>
              Array(size).fill(min_phero));
4         return pheromones;
5     }
```

Листинг 3.6 – Реализация алгоритма обновления матрицы феромонов

```
1  function updatePheromones(matrix, places, visited, pheromones,
       q, k_evaporation) {
2      const ants = places;
3
4      for (let i = 0; i < places; i++) {
5          for (let j = 0; j < places; j++) {
6              let delta = 0;
7              for (let ant = 0; ant < ants; ant++) {
8                  const length = calcLength(matrix, visited[ant]);
9                  delta += q / length;
10             }
11
12             pheromones[i][j] *= 1 - k_evaporation;
13             pheromones[i][j] += delta;
14
15             if (pheromones[i][j] < MIN_PHEROMONE) {
16                 pheromones[i][j] = MIN_PHEROMONE;
17             }
18         }
19     }
20
21     return pheromones;
22 }
```

3.6 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов решения задачи коммивояжера (алгоритма полного перебора) в формате: минимальная стоимость пути, сам путь из вершин. Для муравьиного алгоритма не определены тесты, так как его работа непредсказуема. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 6 & 3 & 9 & 1 \\ 8 & 0 & 3 & 4 & 7 \\ 8 & 4 & 0 & 4 & 6 \\ 8 & 2 & 5 & 0 & 8 \\ 7 & 2 & 4 & 1 & 0 \end{pmatrix}$	7, [0, 4, 3, 1, 2]	7, [0, 4, 3, 1, 2]
$\begin{pmatrix} 0 & 7 & 10 \\ 10 & 0 & 3 \\ 10 & 7 & 0 \end{pmatrix}$	10, [0, 1, 2]	10, [0, 1, 2]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$	44, [0, 1, 2, 3]	44, [0, 1, 2, 3]

Вывод

Были реализованы алгоритмы решения задачи коммивояжера — муравьиный алгоритм и алгоритм полного перебора. Проведено тестирование реализаций алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, проведение исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени.

— Процессор: 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Домашняя 64-разрядная система версии 22H2 [4].

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

4.2 Демонстрация работы программы

Программа получает на вход массив смежностей и выдает стоимость минимального пути и сам путь, вычисленные с помощью муравьиного алгоритма и алгоритма полного перебора.

На рисунке 4.1 представлена демонстрация работы программы.

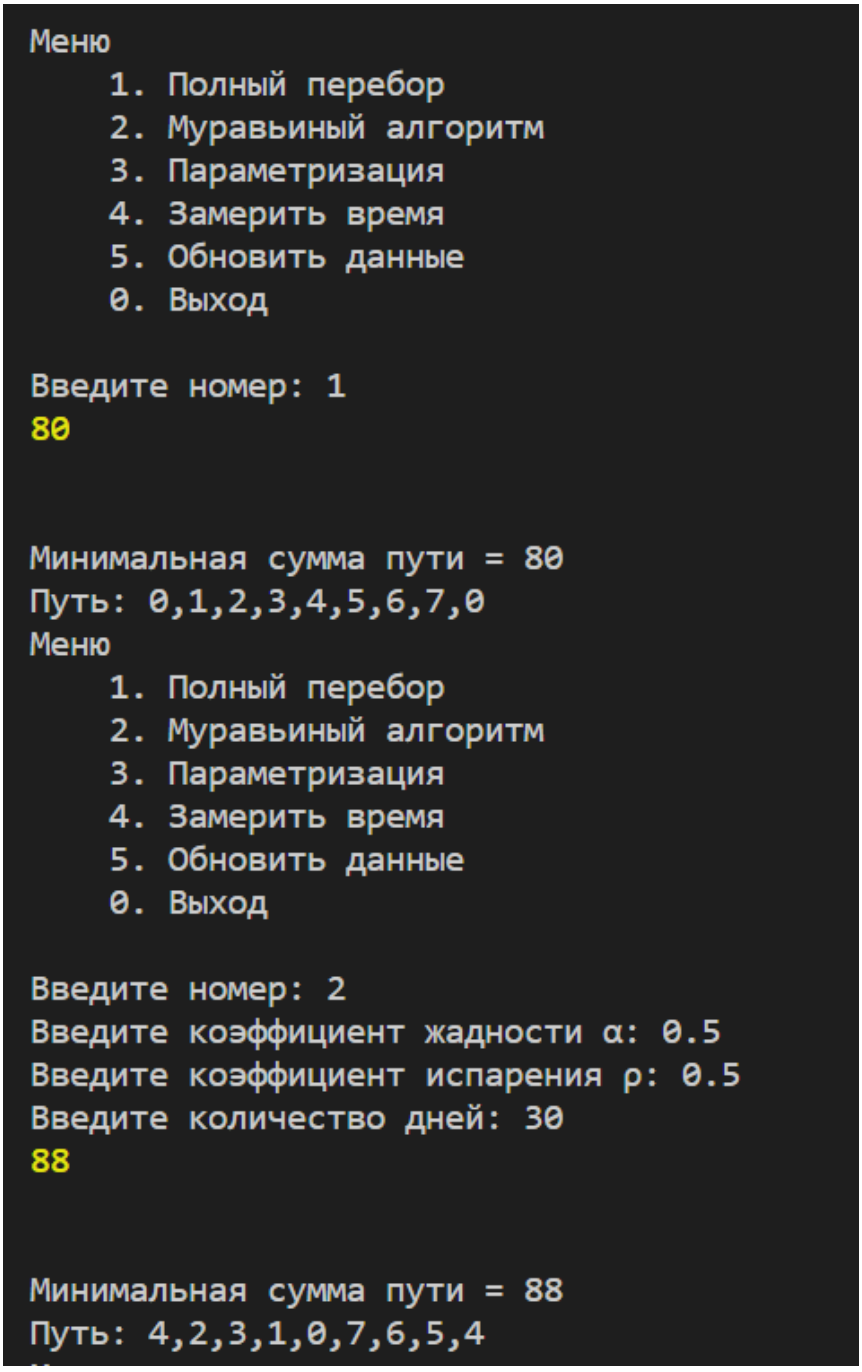

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблице 4.1.

Таблица 4.1 – Результаты замеров времени

Размер матрицы	**Время, мс**	
	Полный перебор	**Муравьиный алгоритм**
1.0	0.086	10.432
2.0	0.145	27.712
3.0	0.403	59.887
4.0	1.673	106.480
5.0	8.893	168.577
6.0	62.227	241.348
7.0	487.975	331.568
8.0	4378.154	449.027

По таблице 4.1 был построен график 4.2. Исходя из этих данных можно понять, что при размере матрицы больше 7 алгоритм полного перебора работает значительно дольше чем муравьиный алгоритм.

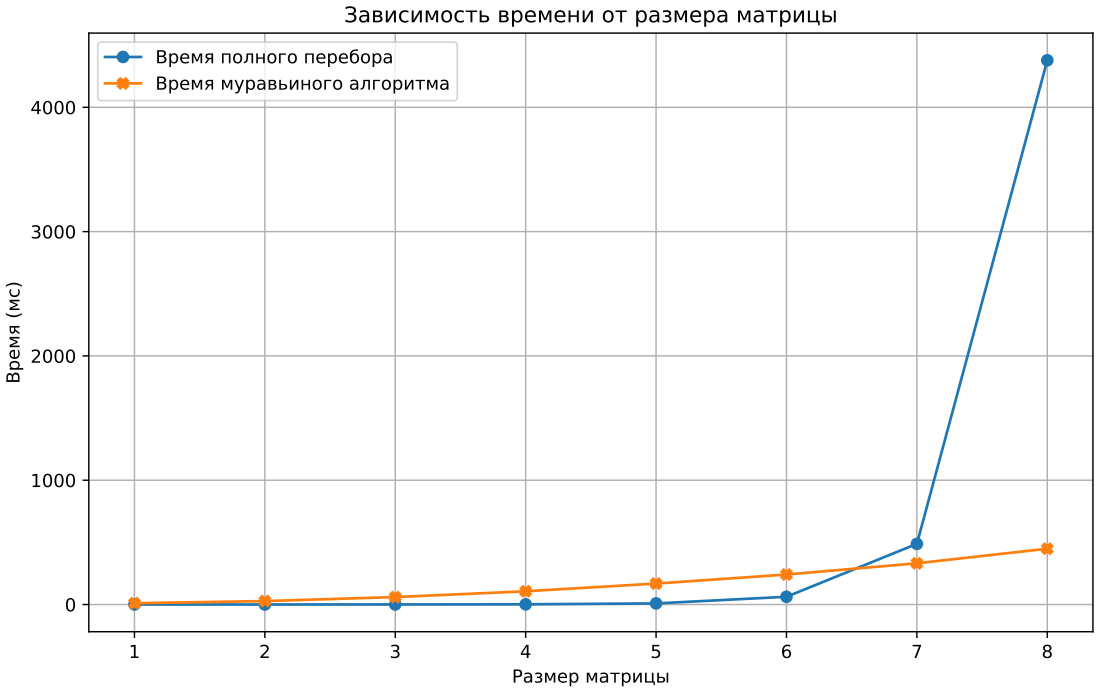

Рисунок 4.2 – Результаты замеров времени работы реализаций алгоритмов для различных линейных размеров матриц

4.4 Постановка эксперимента

Автоматическая параметризация была проведена на двух классах данных — 4.4.1 и 4.4.2. Алгоритм был запущен для набора значений

$\alpha, \rho \in (0, 1)$.

Итоговая таблица значений параметризации состоит из следующих колонок:

— α — коэффициент жадности;

— ρ — коэффициент испарения;

— *Дни* — количество дней жизни колонии муравьев;

— *Ответ* — эталонный результат, полученный методом полного перебора для проведения данного эксперимента;

— *Ошибка* — разность полученного основанным на муравьином алгоритме методом значения и эталонного значения на данных значениях параметров, показатель качества решения.

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

4.4.1 Класс данных 1

Согласно с вариантом представим граф, вершины которого будут является городами России:

1) Калининград;

2) Санкт-Петербург;

3) Москва;

4) Новосибирск;

5) Уфа;

6) Белгород;

7) Хабаровск;

8) Владивосток.

Класс данных 1 представляет собой матрицу смежности размером 8 элементов (разброс значений — от 401 до 8307), которая представлена далее.

$$K_1 = \begin{pmatrix} 0 & 79390 & 716 & 6100 & 1430 & 1534 & 691 & 631 \\ 79390 & 0 & 401 & 3667 & 1761 & 2407 & 1349 & 1254 \\ 716 & 401 & 0 & 3305 & 193904 & 2441 & 1942 & 1850 \\ 6100 & 3667 & 3305 & 0 & 3379 & 3106 & 859 & 4700 \\ 1430 & 1761 & 193904 & 3379 & 0 & 644 & 5159 & 6337 \\ 1534 & 2407 & 2441 & 3106 & 644 & 0 & 4843 & 6000 \\ 691 & 1349 & 1942 & 859 & 5159 & 4843 & 0 & 8307 \\ 631 & 1254 & 1850 & 4700 & 6337 & 6000 & 8307 & 0 \end{pmatrix} \quad (4.1)$$

Для данного класса данных приведена таблица 4.2 – 4.4 с выборкой параметров, которые наилучшим образом решают поставленную задачу. Использованы следующие обозначения: Days — количество дней, Result — результат работы, Mistake — ошибка как отклонение решения от эталонного.

В выборке, разделенной на подгруппы по признаку значения параметра α, для пары (α, ρ) выбран набор значений параметров, обеспечивающих наилучший результат приближения (наименьшее значение параметра mistake). Если одинаковый результат параметра mistake достигается для нескольких кортежей $(\alpha, \rho, days, result, mistake)$, содержащих одинаковые значения параметров (α, ρ), среди них выбирается кортеж, содержащий наименьшее значение параметра days.

Таблица 4.2 – Результаты параметризации на классе данных 1

α	ρ	Days	Result	Mistake
0.1	0.1	1	9390	0
0.1	0.2	5	9390	0
0.1	0.3	5	9390	0
0.1	0.4	10	9390	0
0.1	0.5	5	9390	0
0.1	0.6	50	9390	0
0.1	0.7	50	9390	0
0.1	0.8	5	9390	0
0.2	0.1	10	9390	0
0.2	0.2	50	9390	0
0.2	0.3	10	9390	0
0.2	0.4	10	9390	0
0.2	0.5	50	9390	0
0.2	0.6	50	9390	0
0.2	0.7	10	9390	0
0.2	0.8	50	9390	0
0.3	0.1	10	9390	0
0.3	0.2	100	9390	0
0.3	0.3	50	9390	0
0.3	0.4	10	9390	0
0.3	0.5	100	9390	0
0.3	0.6	50	9390	0
0.3	0.7	100	9390	0
0.3	0.8	50	9390	0

Таблица 4.3 – Результаты параметризации на классе данных 1

α	ρ	Days	Result	Mistake
0.4	0.1	50	9390	0
0.4	0.2	50	9390	0
0.4	0.3	50	9390	0
0.4	0.4	10	9390	0
0.4	0.5	50	9390	0
0.4	0.6	50	9390	0
0.4	0.7	50	9390	0
0.4	0.8	50	9390	0
0.5	0.1	50	9390	0
0.5	0.2	50	9390	0
0.5	0.3	50	9390	0
0.5	0.4	10	9390	0
0.5	0.5	50	9390	0
0.5	0.6	100	9390	0
0.5	0.7	50	9390	0
0.5	0.8	300	9390	0
0.6	0.1	100	9390	0
0.6	0.2	50	9390	0
0.6	0.3	100	9390	0
0.6	0.4	100	9390	0
0.6	0.5	50	9390	0
0.6	0.6	50	9390	0
0.6	0.7	10	9390	0
0.6	0.8	100	9390	0
0.7	0.1	100	9390	0
0.7	0.2	50	9390	0

Таблица 4.4 – Результаты параметризации на классе данных 1

α	ρ	Days	Result	Mistake
0.7	0.3	50	9390	0
0.7	0.4	100	9390	0
0.7	0.5	50	9390	0
0.7	0.6	50	9390	0
0.7	0.7	10	9390	0
0.8	0.8	10	9390	0
0.8	0.1	50	9390	0
0.8	0.2	300	9390	0
0.8	0.3	100	9390	0
0.8	0.4	300	9390	0
0.8	0.5	300	9390	0
0.8	0.6	100	9390	0
0.8	0.7	300	9390	0
0.8	0.8	50	9390	0
0.9	0.1	300	9390	0
0.9	0.2	300	9390	0
0.9	0.3	100	9390	0
0.9	0.4	9390	9390	0
0.9	0.5	300	9390	0
0.9	0.6	50	9390	0
0.9	0.7	100	9390	0
0.9	0.8	50	9390	0

4.4.2 Класс данных 2

Класс данных 2 представляет собой матрицу смежности размером 9 элементов (разброс значений - от 10 до 100), которая представлена далее.

$$K_1 = \begin{pmatrix} 0 & 30 & 20 & 80 & 40 & 50 & 30 & 20 \\ 30 & 0 & 15 & 60 & 35 & 40 & 25 & 20 \\ 20 & 15 & 0 & 50 & 30 & 35 & 30 & 25 \\ 80 & 60 & 50 & 0 & 55 & 45 & 20 & 60 \\ 40 & 35 & 30 & 55 & 0 & 15 & 65 & 80 \\ 50 & 40 & 35 & 45 & 15 & 0 & 50 & 70 \\ 30 & 25 & 30 & 20 & 65 & 50 & 0 & 60 \\ 20 & 20 & 25 & 60 & 80 & 70 & 60 & 0 \end{pmatrix} \tag{4.2}$$

Для данного класса данных приведена таблица 4.5 – 4.7 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.5 – Результаты параметризации на классе данных 2

α	ρ	Days	Result	Mistake
0.1	0.1	50	195	0
0.1	0.2	3	195	0
0.1	0.3	3	195	0
0.1	0.4	1	195	0
0.1	0.5	10	195	0
0.1	0.6	50	195	0
0.1	0.7	100	195	0
0.1	0.8	50	195	0
0.2	0.1	10	195	0
0.2	0.2	5	195	0
0.2	0.3	50	195	0
0.2	0.4	10	195	0
0.2	0.5	3	195	0
0.2	0.6	50	195	0
0.2	0.7	50	195	0
0.2	0.8	10	195	0
0.3	0.1	10	195	0
0.3	0.2	100	195	0
0.3	0.3	5	195	0
0.3	0.4	10	195	0
0.3	0.5	100	195	0
0.3	0.6	50	195	0
0.3	0.7	100	195	0
0.3	0.8	50	195	0

Таблица 4.6 – Результаты параметризации на классе данных 2

α	ρ	Days	Result	Mistake
0.4	0.1	50	195	0
0.4	0.2	100	195	0
0.4	0.3	50	195	0
0.4	0.4	50	195	0
0.4	0.5	100	195	0
0.4	0.6	50	195	0
0.4	0.7	10	195	0
0.4	0.8	50	195	0
0.5	0.1	300	195	0
0.5	0.2	100	195	0
0.5	0.3	50	195	0
0.5	0.4	300	195	0
0.5	0.5	300	195	0
0.5	0.6	300	195	0
0.5	0.7	50	195	0
0.5	0.8	300	195	0
0.6	0.1	50	195	0
0.6	0.2	100	195	0
0.6	0.3	100	195	0
0.6	0.4	300	195	0
0.6	0.5	50	195	0
0.6	0.6	300	195	0
0.6	0.7	10	195	0
0.6	0.8	50	195	0

Таблица 4.7 – Результаты параметризации на классе данных 2

α	ρ	Days	Result	Mistake
0.7	0.1	100	195	0
0.7	0.2	100	195	0
0.7	0.3	50	195	0
0.7	0.4	50	195	0
0.7	0.5	50	195	0
0.7	0.6	100	195	0
0.7	0.7	300	195	0
0.8	0.8	300	195	0
0.8	0.1	50	195	0
0.8	0.2	50	195	0
0.8	0.3	100	195	0
0.8	0.4	100	195	0
0.8	0.5	10	195	0
0.8	0.6	50	195	0
0.8	0.7	300	195	0
0.8	0.8	50	195	0
0.9	0.1	100	195	0
0.9	0.2	300	195	0
0.9	0.3	300	195	0
0.9	0.4	100	195	0
0.9	0.5	100	195	0
0.9	0.6	100	195	0
0.9	0.7	100	195	0
0.9	0.8	300	195	0

4.5 Вывод

В результате эксперимента было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц (больше 7).

Также при проведении эксперимента с классами данных было получено, что на первом классе данных муравьиный алгоритм лучше всего показывает себя при параметрах:

- $\alpha = 0.1, \rho = 0.1, days = 1$;
- $\alpha = 0.2, \rho \in \{0.1, 0.3, 0.4, 0.7\}, days = 10$;
- $\alpha = 0.3, \rho \in \{0.1, 0.4\}, days = 10$;
- $\alpha = 0.4, \rho = 0.4, days = 10$;
- $\alpha = 0.5, \rho = 0.4, days = 10$;
- $\alpha = 0.6, \rho = 0.7, days = 10$;
- $\alpha = 0.7, \rho \in \{0.7, 0.8\}, days = 10$.

Следовательно, для класса данных 1 рекомендуется использовать данные параметры.

Для класса данных 2 было получено, что наилучшим образом алгоритм работает на значениях параметров, которые представлены далее:

- $\alpha = 0.1, \rho = 0.4, days = 1$;
- $\alpha = 0.2, \rho = 0.5, days = 3$;
- $\alpha = 0.3, \rho = 0.3, days = 5$;
- $\alpha = 0.4, \rho = 0.7, days = 10$;
- $\alpha = 0.6, \rho = 0.7, days = 10$;
- $\alpha = 0.8, \rho = 0.5, days = 10$.

Для второго класса данных 2 рекомендуется использовать данные параметры.

Заключение

Было экспериментально подтверждено различие во временной эффективности муравьиного алгоритма и алгоритма полного перебора решения задачи коммивояжера. В результате исследований можно сделать вывод о том, что при матрицах большого размера (больше 7) стоит использовать муравьиный алгоритм решения задачи коммивояжера, а не алгоритм полного перебора.

В ходе выполнения лабораторной работы были решены все задачи:

1) Описаны методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма.

2) Разработан и реализован программный продукт, позволяющий решить задачу коммивояжера исследуемыми методами.

3) Сравнены по времени метод полного перебора и метод на основе муравьиного алгоритма.

4) Описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Исходя из полученных результатов, использование муравьиного алгоритма наиболее эффективно по времени при больших размерах матриц. Это следует из рассчитанной трудоемкости алгоритмов и проведенного замера времени. Следовательно, при размерах матриц больше 7 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует оптимального решения, в отличие от метода полного перебора.

Список использованных источников

1 Ульянов. М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ: учебное пособие. — Москва: Наука, ФИЗМАТЛИТ, 2007. с. 376.

2 Документация по JavaScript [Электронный ресурс]. — Режим доступа: `https://262.ecma-international.org/13.0/` (дата обращения: 10.10.2023).

3 Node.js function process.cpuUsage([previousValue]) [Электронный ресурс]. — Режим доступа: `https://nodejs.org/api/process.html#processcpuusagepreviousvalue` (дата обращения: 10.10.2023).

4 Windows 11 [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows11` (дата обращения: 10.10.2023).