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 Домашней работе

по курсу «Анализ алгоритмов»

на тему: «Методы решения задачи коммивояжёра»

Студент ИУ7-54Б (Группа) _____ (Подпись, дата) Разин А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

Введение

В последние два десятилетия при оптимизации сложных систем исследователи все чаще применяют природные механизмы поиска наилучших решений. Один из таких механизмов — это муравьиные алгоритмы, представляющие собой новый перспективный метод оптимизации, базирующийся на моделировании поведения колонии муравьев [1].

Первый вариант муравьиного алгоритма был предназначен для приближенного решения задачи коммивояжера [2].

Целью данной работы является разработка программного обеспечения, решающего задачу коммивояжера двумя способами: полным перебором и с помощью муравьиного алгоритма. Для поставленной цели необходимо выполнить следующие задачи.

1. Описать задачу коммивояжера.
2. Описать методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма.
3. Привести схемы муравьиного алгоритма и алгоритма, позволяющего решить задачу коммивояжера методом полного перебора.
4. Разработать и реализовать программный продукт, позволяющий решить задачу коммивояжера исследуемыми методами.
5. Сравнить по времени метод полного перебора и метод на основе муравьиного алгоритма.
6. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

Выданный индивидуальный вариант для выполнения лабораторной работы:

- неориентированный граф;
- без элитных муравьев;
- незамкнутый маршрут;
- карта перемещения по Африке.

1 Аналитическая часть

В данной части работы будет описана задача коммивояжера, а также будут описаны способы ее решения — метод полного перебора и метод на основе муравьиного алгоритма.

1.1 Задача коммивояжера

Задача коммивояжера занимает особое место в комбинаторной оптимизации и исследовании операций. Исторически она была одной из тех задач, которые послужили толчком для развития этих направлений. В данной задаче рассматривается n городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса [3].

1.2 Решение с использованием полного перебора

Суть данного решения состоит в переборе всех возможных вариантов замкнутых путей и в выборе кратчайшего из них. Данное решение имеет оценку в $O(n!)$, что затрудняет его использование в графах с большим количеством вершин.

1.3 Решение с использованием муравьиного алгоритма

При поиске путей до пищи муравьи общаются друг с другом с помощью феромонов. В случае если путь длинный, феромоны испарятся и последующие сородичи предпочтут иной путь с большим числом феромонов, таким образом наибольшее число феромона будет оставлено на кратчайших путях.

Муравей имеет следующие органы чувств:

1) зрение — муравей способен определить длину ребра;

2) память — муравей способен запомнить пройденный маршрут;

3) обоняние — муравей способен чуять феромон.

Введем функцию (1.1), характеризующую привлекательность ребра, определяемую благодаря зрению.

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

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

Вероятность перехода из пункта i в пункт j определяется по формуле (1.2).

$$P_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}, \tag{1.2}$$

где:

1. $\tau_{i,j}$ — количество феромонов на ребре от i до j;
2. $\eta_{i,j}$ — привлекательность пути от i до j;
3. α — параметр влияния расстояния;
4. β — параметр влияния феромона.

При $\alpha = 0$ будет выбран ближайший город, что соответствует жадному алгоритму в классической теории оптимизации. Если $\beta = 0$, работает лишь феромонное усиление, что влечет за собой быстрое вырождение маршрутов к одному субоптимальному решению.

После завершения пути, ночью, происходит обновление феромона на пройденных путях по формуле (1.3), в случае, если p — коэффициент испарения феромона, N — количество феромонов, Q — некоторая константа порядка длины путей, L_k — длина пути муравья с номером k.

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}, \quad \Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^{k}, \tag{1.3}$$

где

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & \text{ребро посещено } k\text{-ым муравьем,} \\ 0, & \text{иначе.} \end{cases} \tag{1.4}$$

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

Вывод

В данной части работы были рассмотрены идеи, необходимые для разработки и реализации двух рассматриваемых алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

2 Конструкторская часть

В данной части работы будут рассмотрены схемы муравьиного алгоритма и алгоритма полного перебора.

2.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- выбор файла с матрицей расстояний;
- ввод параметров α, β и $days$ для муравьиного алгоритма;
- вывод на экран найденного каждым из алгоритмов кратчайшего пути и его длины;
- измерение времени получения результатов каждого из рассматриваемых алгоритмов.

2.2 Схемы алгоритмов

На рисунке 2.1 представлен алгоритм перебора всех возможных путей. На рисунках 2.2–2.5, представлен муравьиный алгоритм и схемы алгоритмов расчета данных для муравьиного алгоритма соотвественно.

Вывод

В данной части работы были рассмотрены схемы муравьиного алгоритма и алгоритма полного перебора.

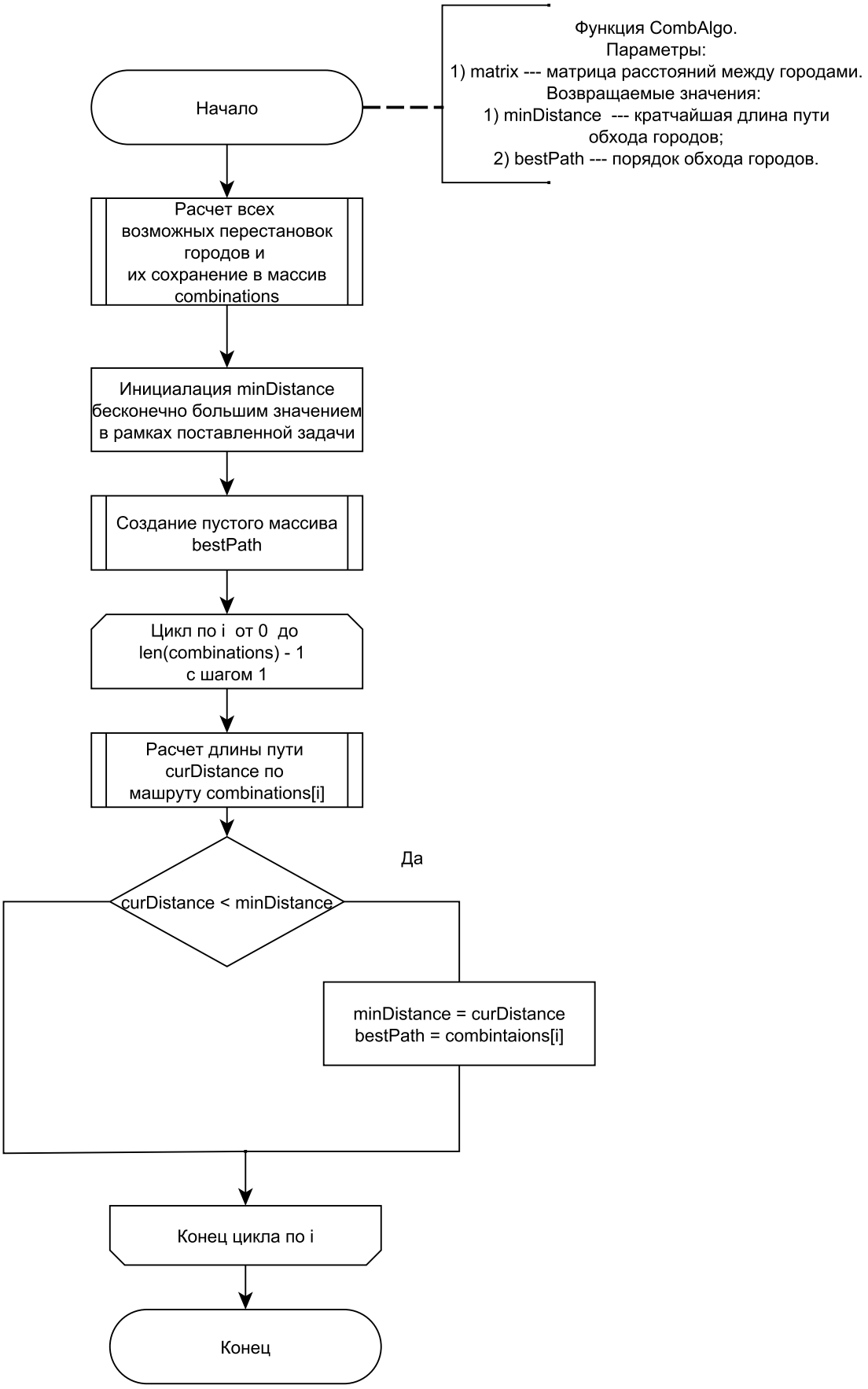

Рисунок 2.1 – Схема алгоритма перебора всех возможных путей

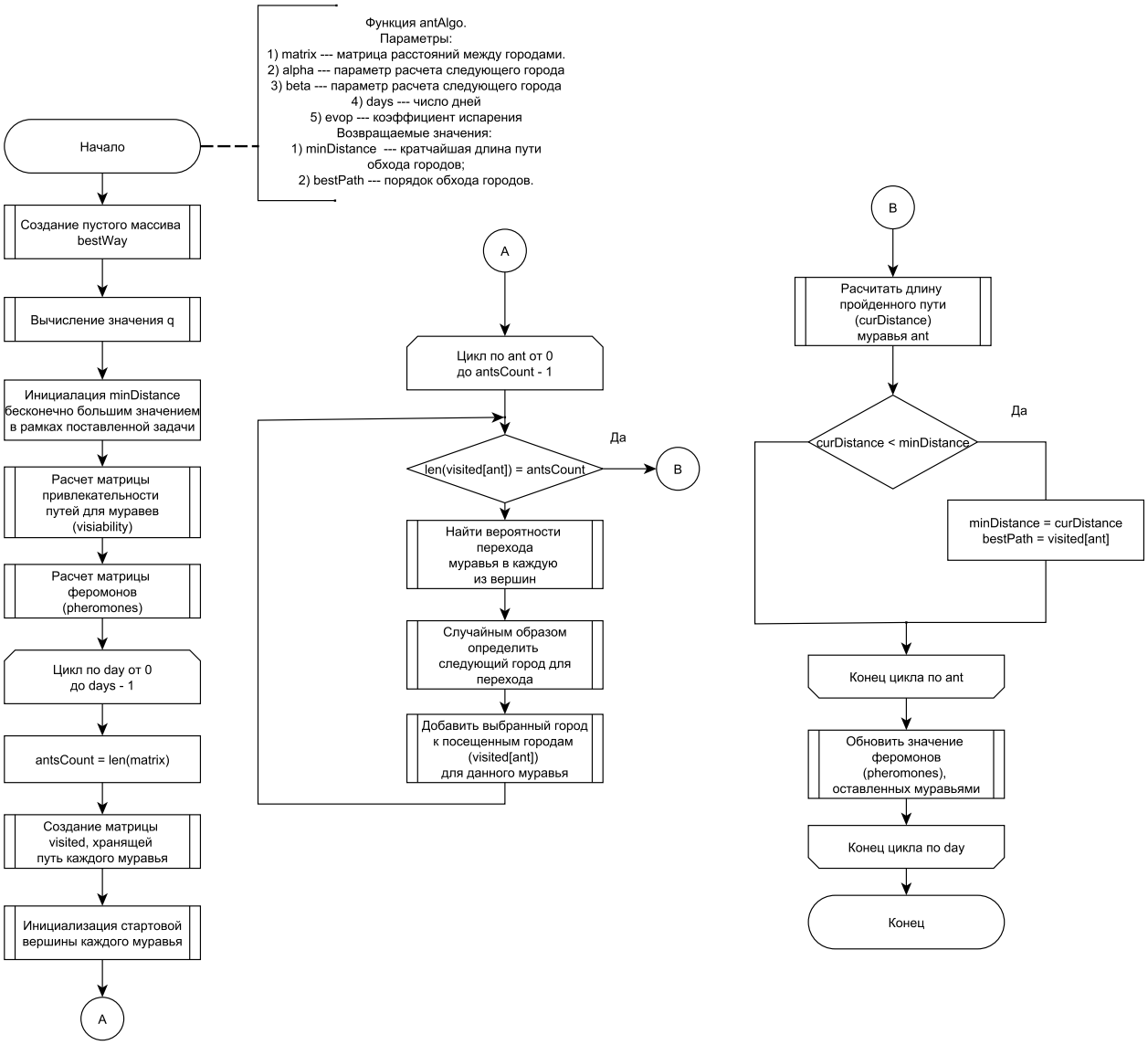

Рисунок 2.2 – Схема муравьиного алгоритма

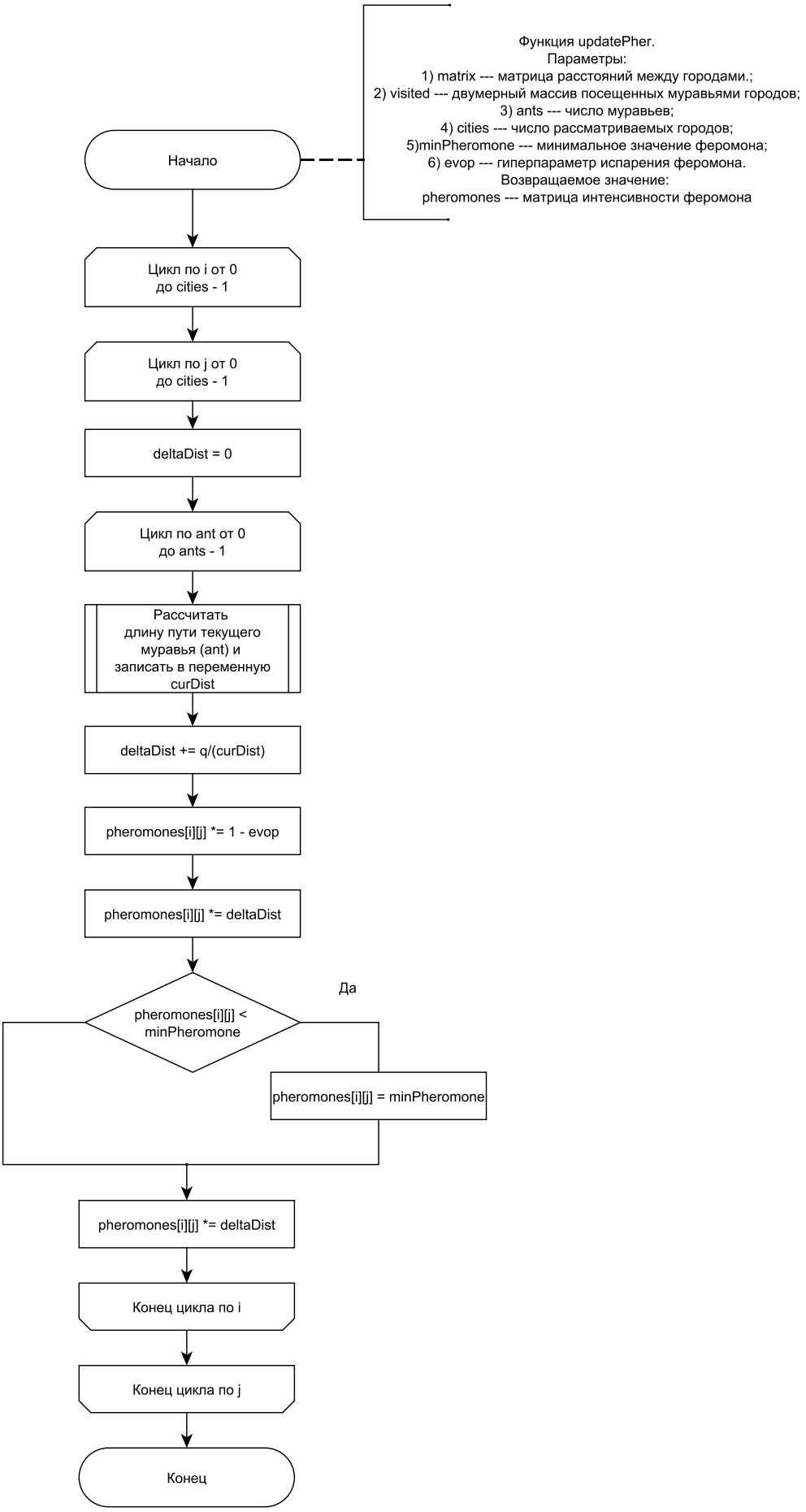

Рисунок 2.3 – Схема алгоритма обновления феромонов

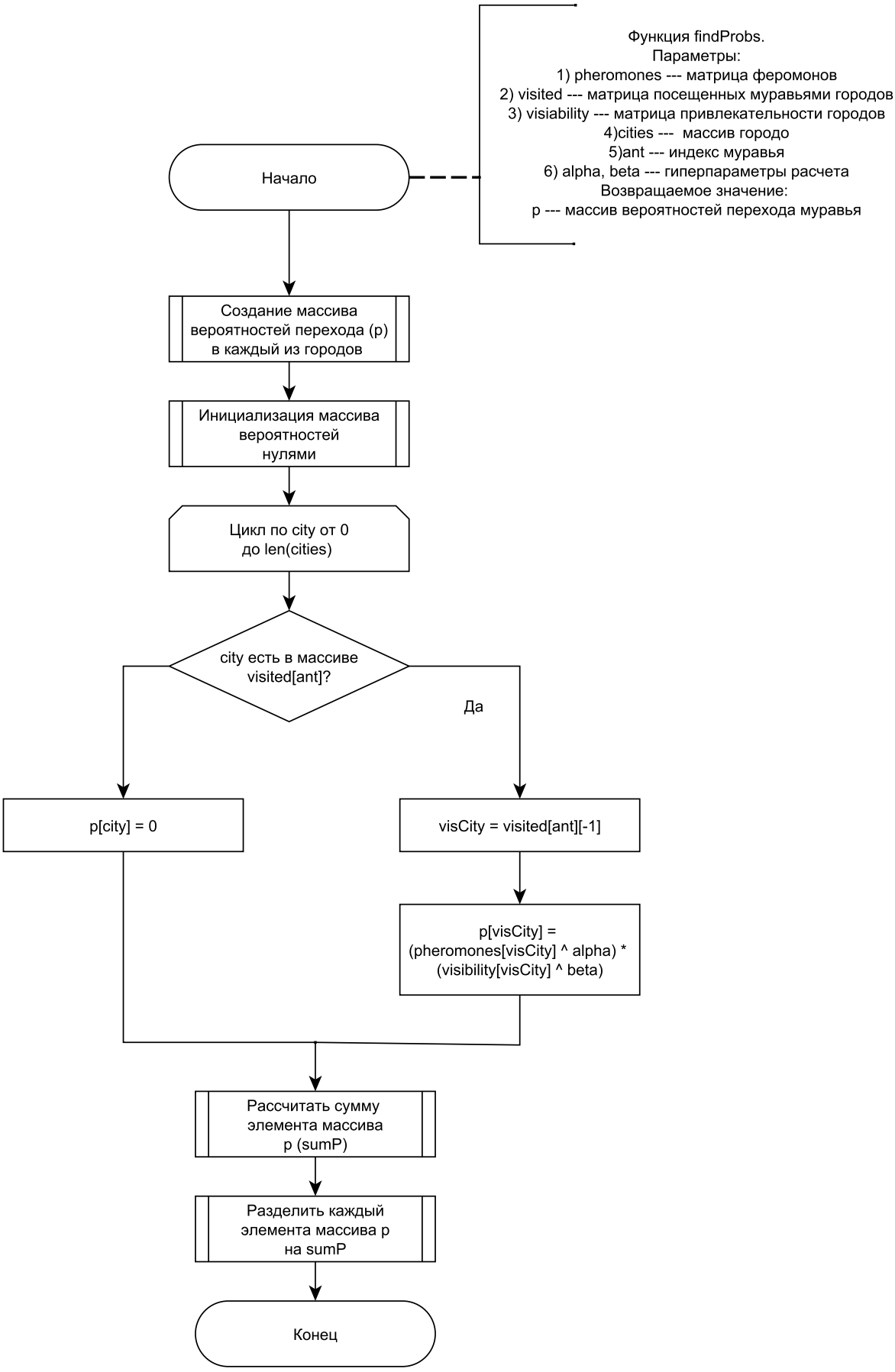

Рисунок 2.4 – Схема алгоритма расчета вероятностей перехода муравья в вершины

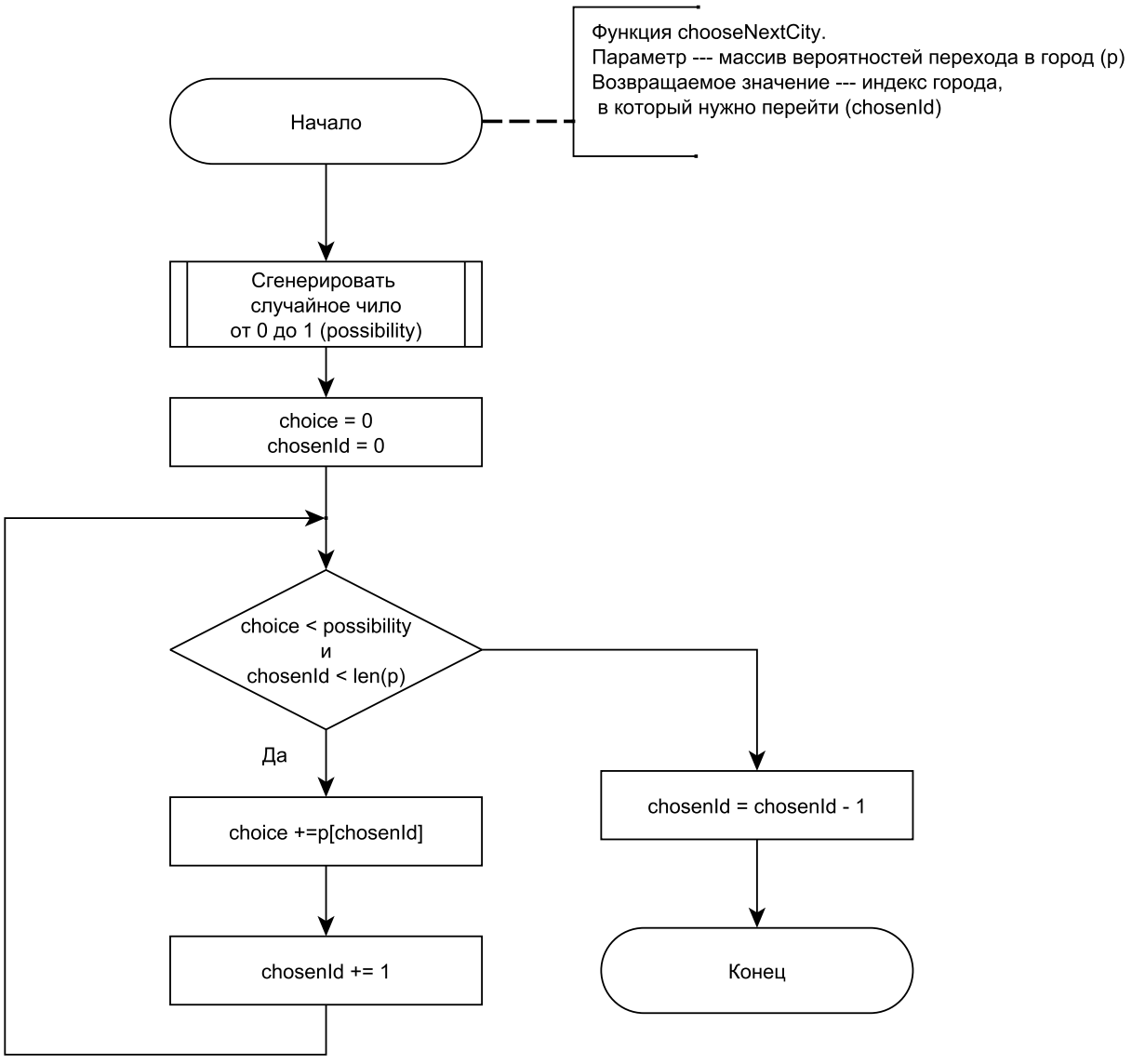

Рисунок 2.5 – Схема алгоритма выбора следующей вершины для перехода

3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги и функциональные тесты.

3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке `Python`, при использовании компилятора, так как в стандартной библиотеке приведенного языка присутствует функция `process_time`, позволяющая измерять время процессорного времени [4].

3.2 Реализация алгоритмов

Листинги исходных кодов программ А.1–А.4 приведены в приложении.

3.3 Тестирование

В таблице 3.1, приведены функциональные тесты программы, значения в столбце «Результат программы», обозначает минимальное расстояние и порядок обхода городов, значения в данном столбце были получены при использовании обоих алгоритмов.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 1 & 5 & 6 \\ 1 & 0 & 8 & 8 \\ 5 & 8 & 0 & 10 \\ 6 & 8 & 10 & 0 \end{pmatrix}$	14, [2, 0, 1, 3]	14, [2, 0, 1, 3]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix}$	3, [1, 0, 2]	3, [1, 0, 2]
$\begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$	3, [0, 1]	3, [0, 1]

Вывод

В данной части работы были описаны средства реализации и представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательская часть

В данном разделе будет описано исследование зависимости среднего числа генерируемых кадров от числа и типа примитивов на сцене. Также будет описаны технические характеристики устройства, на котором проводились замеры и приведен анализ полученных результатов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Майкрософт Windows 10 Pro [5].
4. Использованная подсистема: WSL2 [**WSL2**].

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Штовба, С. Д. Муравьиные алгоритмы // Exponenta Pro. — 2003. — № 4. — С. 70–75.
2. Ершов, Н. М. Лекция 10. Муравьиные алгоритмы // ВМК МГУ. — 2011. — С. 1–14.
3. Задача коммивояжера [Электронный ресурс]. — Режим доступа: `http://old.math.nsc.ru/LBRT/k5/OR-MMF/TSPr.pdf` (дата обращения: 28.11.2022).
4. time — Time access and conversions [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/library/time.html` (дата обращения: 22.09.2022).
5. Windows client documentation for IT Pros [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/en-us/windows/resources/` (дата обращения: 22.09.2022).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация муравьиного алгоритма

```
def antAlg(matrix, cities, alpha, beta, k_evaporation, days):
    q = calcQ(matrix, cities)
    bestWay = []
    minDist = float("inf")
    pheromones = calcPheromones(cities)
    visibility = calcVisibility(matrix, cities)
    ants = cities
    for day in range(days):
        route = np.arange(cities)
        visited = calcVisitedPlaces(route, ants)

        for ant in range(ants):
            while (len(visited[ant]) != ants):
                pk = findProbs(pheromones, visibility, visited,
                   cities, ant, alpha, beta)
                chosenPlace = chooseNextCity(pk)
                visited[ant].append(chosenPlace - 1)

            curLength = calcLength(matrix, visited[ant])

            if (curLength < minDist):
                minDist = curLength
                bestWay = visited[ant]

        pheromones = updPher(matrix, cities, visited,
           pheromones, q, k_evaporation)

    return minDist, bestWay
```

Листинг А.2 – Реализация алгоритма полного перебора

```
def CombAlg(matrix, size):

    places = np.arange(size)
    placesCombinations = list()

    for combination in it.permutations(places):
        combArr = list(combination)
```

```
        placesCombinations.append(combArr)

    minDist = float("inf")

    for i in range(len(placesCombinations)):
        curDist = 0
        for j in range(size - 1):
            startCity = placesCombinations[i][j]
            endCity = placesCombinations[i][j + 1]
            curDist += matrix[startCity][endCity]

        if (curDist < minDist):
            minDist = curDist
            bestWay = placesCombinations[i]

    return minDist, bestWay
```

Листинг А.3 – Реализация алгоритма расчета вероятностей перехода в не посещенную вершину графа

```
def findProbs(pheromones, visibility, visited, places, ant,
   alpha, beta):
    pk = [0] * places

    for place in range(places):
        if place not in visited[ant]:
            ant_place = visited[ant][-1]
            pk[place] = pow(pheromones[ant_place][place], alpha)
               * \
                pow(visibility[ant_place][place], beta)
        else:
            pk[place] = 0

    sum_pk = sum(pk)

    for place in range(places):
        pk[place] /= sum_pk

    return pk
```

Листинг А.4 – Реализация алгоритма расчета значения феромонов

```
def updPher(matrix, cities, visited, pheromones, q,
   k_evaporation):
```

```
    ants = cities

    for i in range(cities):
        for j in range(cities):
            delta = 0
            for ant in range(ants):
                cur_dist = calcLength(matrix, visited[ant])
                delta += q / cur_dist

            pheromones[i][j] *= (1 - k_evaporation)
            pheromones[i][j] += delta
            if (pheromones[i][j] < MIN_PHEROMONE):
                pheromones[i][j] = MIN_PHEROMONE

    return pheromones
```