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 курсу «Анализ Алгоритмов»

на тему: «Задача коммивояжера»

Студент группы ИУ7-56Б _____ (Подпись, дата) Чупахин М. Д. (Фамилия И.О.)

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

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут рассмотрены методы решения задачи коммивояжера.

Одной из наиболее известных и значимых задач в области транспортной логистики является задача коммивояжёра, также известная как "задача о странствующем торговце". Суть этой задачи заключается в поиске оптимального пути, который может быть самым коротким, быстрым или экономически выгодным, проходя через промежуточные пункты один раз и возвращаясь в начальную точку. [1] Например, это может быть оптимальный маршрут, позволяющий торговцу посетить определенные города один раз и вернуться обратно с минимальным временем, расходами или длиной пути. В современных условиях, когда стоимость доставки иногда сравнима с стоимостью товара, а скорость доставки играет важную роль, поиск оптимального маршрута становится критически важным.

Муравьиный алгоритм представляет собой один из эффективных методов для приближенного решения задачи коммивояжёра, а также аналогичных задач поиска маршрутов на графах. Этот метод основан на анализе и использовании модели поведения муравьев, которые ищут пути от колонии к источнику питания. Муравьиный алгоритм представляет собой оптимизацию, использующую принципы поведения муравьев для нахождения эффективных решений задачи.

Цель данной лабораторной работы заключается в описании и исследовании методов решения задачи коммивояжера.

Ниже представлены задачи, которые необходимо выполнить для достижения поставленной цели.

1) Описать задачу коммивояжера.

2) Описать метод решения задачи полным перебором.

3) Описать муравьиный алгоритм.

4) Разработать программу, реализующую описанные методы решения.

5) Проанализировать затраты времени работы программы и выявить их зависимость от различных параметров.

1 Аналитическая часть

В данном разделе будут рассмотрены методы решения задачи коммивояжера: полным перебором и с использованием муравьиного алгоритма.

1.1 Полный перебор

Алгоритм полного перебора для задачи коммивояжера характеризуется высокой вычислительной сложностью $(n!)$, где n – число городов. Суть метода заключается в исследовании всех возможных маршрутов в графе с последующим выбором наименьшего. [2] Хотя это позволяет получить оптимальное решение, временные затраты на выполнение значительны, особенно при даже небольшом количестве вершин в графе.

1.2 Муравьиный алгоритм

Метод оптимизации, известный как муравьиный алгоритм, основывается на модели поведения муравьев. Муравьи руководствуются своими органами чувств в процессе действия. Каждый муравей оставляет на своем пути феромоны, создавая таким образом след, который может быть использован другими муравьями для ориентации. При большом количестве муравьев наибольшее количество феромона остается на самых посещаемых путях, причем частота посещения может зависеть от длин ребер.

Идея заключается в том, что отдельный муравей ограничен в своих возможностях, поскольку способен выполнять только простые задачи. Однако, когда их большое количество, они могут действовать как самостоятельные вычислительные единицы. Муравьи взаимодействуют друг с другом, используя непрямой обмен информацией через окружающую среду посредством феромона.

Муравей обладает 3-мя свойствами:

1) зрение – муравей может видеть «длину» (метку) ребра / дуги и оценить привлекательность ребра;

2) обоняние – муравей чует концентрацию ферамона в день t на ребре / дуге;

3) память – у муравья есть список посещенных за текущий день t городов.

В день t, находясь в городе i, муравей k выбирает следующий город на основе следующего вероятностного правила (формула 1.1):

$$P_{ij,k}(t) = \begin{cases} \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum_{q \notin J_k} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, j \notin J_k \\ 0, j \in J_k \end{cases} \tag{1.1}$$

где J_k - список посещенных городов за текущий день; η_{ij} - привлекательность ребра; τ_{ij} - количество ферамонов на ребре; α - коэффициент жадности решения; β - коэффициент стадности; $\alpha + \beta = 1$; α, $\beta \in (0, 1)$. При $\alpha = 0, \beta = 1$ решение стадное. При $\alpha = 1, \beta = 0$ решение жадное.

Перед наступлением нового дня феромон обновляется по формуле 1.2:

$$\tau_{ij}(t+1) = \tau_{ij}(t)(1-\rho) + \Delta\tau_{ij}(t), \Delta\tau_{ij}(t) = \sum_{k=1}^{N} \Delta\tau_{ij,k}(t) \tag{1.2}$$

Модификация с элитными муравьями: перед рассветом элитный муравей (муравьи) усиливают рёбра лучшего(их) маршрута(ов).

Вывод

В данном разделе были рассмотрены методы решения задачи коммивояжера: полным перебором и с использованием муравьиного алгоритма.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программное обеспечение должно удовлетворять следующим функциональным требованиям: на входе – матрица смежности и дополнительные параметры, необходимые для муравьиного алгоритма, на выходе – длина кратчайшего маршрута и сам маршрут.

Программное обеспечение также должно соответствовать следующим требованиям:

— наличие пользовательского интерфейса для выбора действий;

— вывод длины минимального маршрута и самого маршрута;

— предоставление функционала для измерения времени выполнения алгоритмов.

2.2 Разработка алгоритмов

В данном разделе представлены схемы алгоритмов для полного перебора (рисунки 2.1, 2.2) и муравьиного алгоритма (рисунки 2.3 – 2.6)

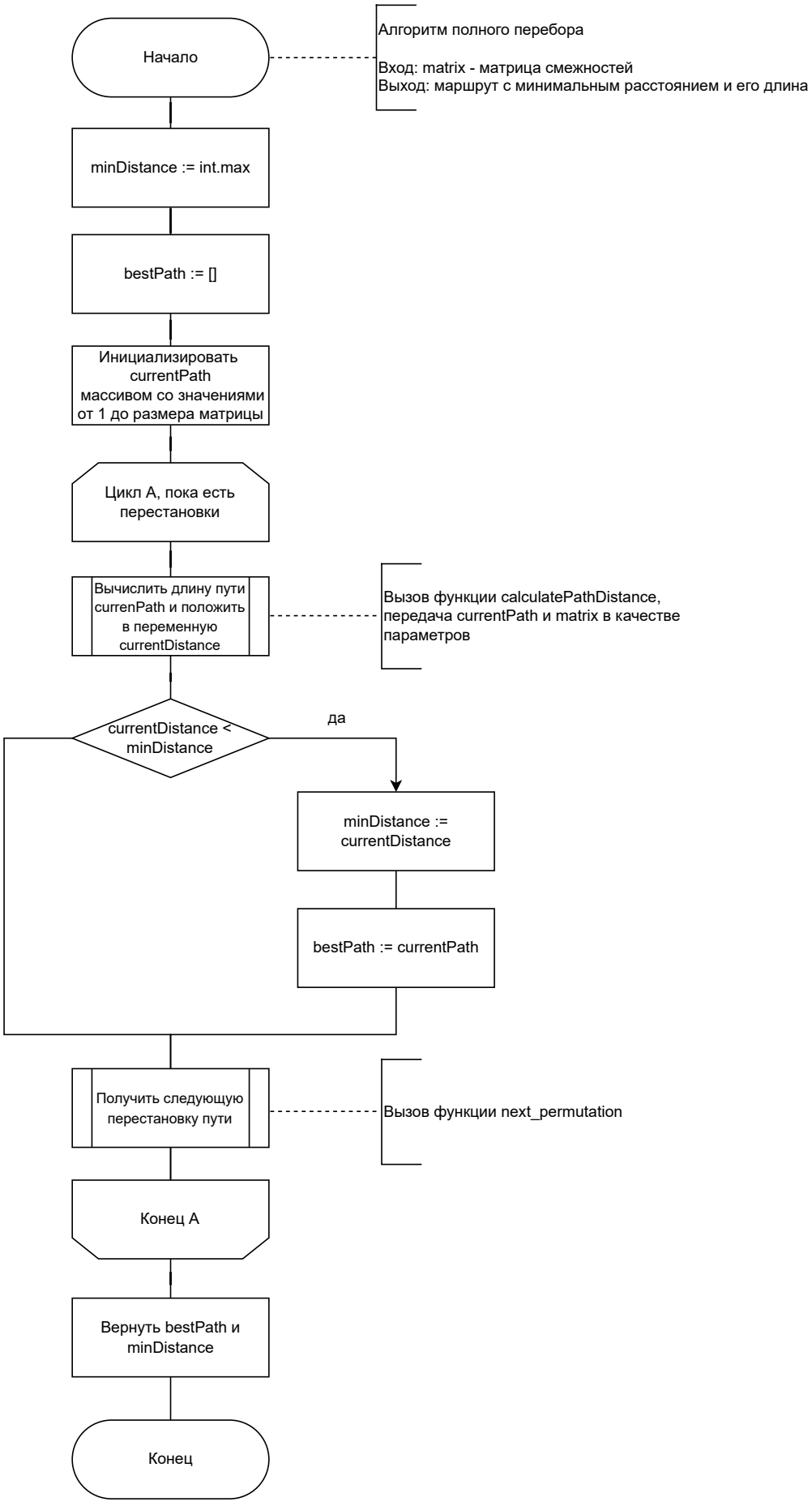

Рисунок 2.1 – Схема метода полным перебора

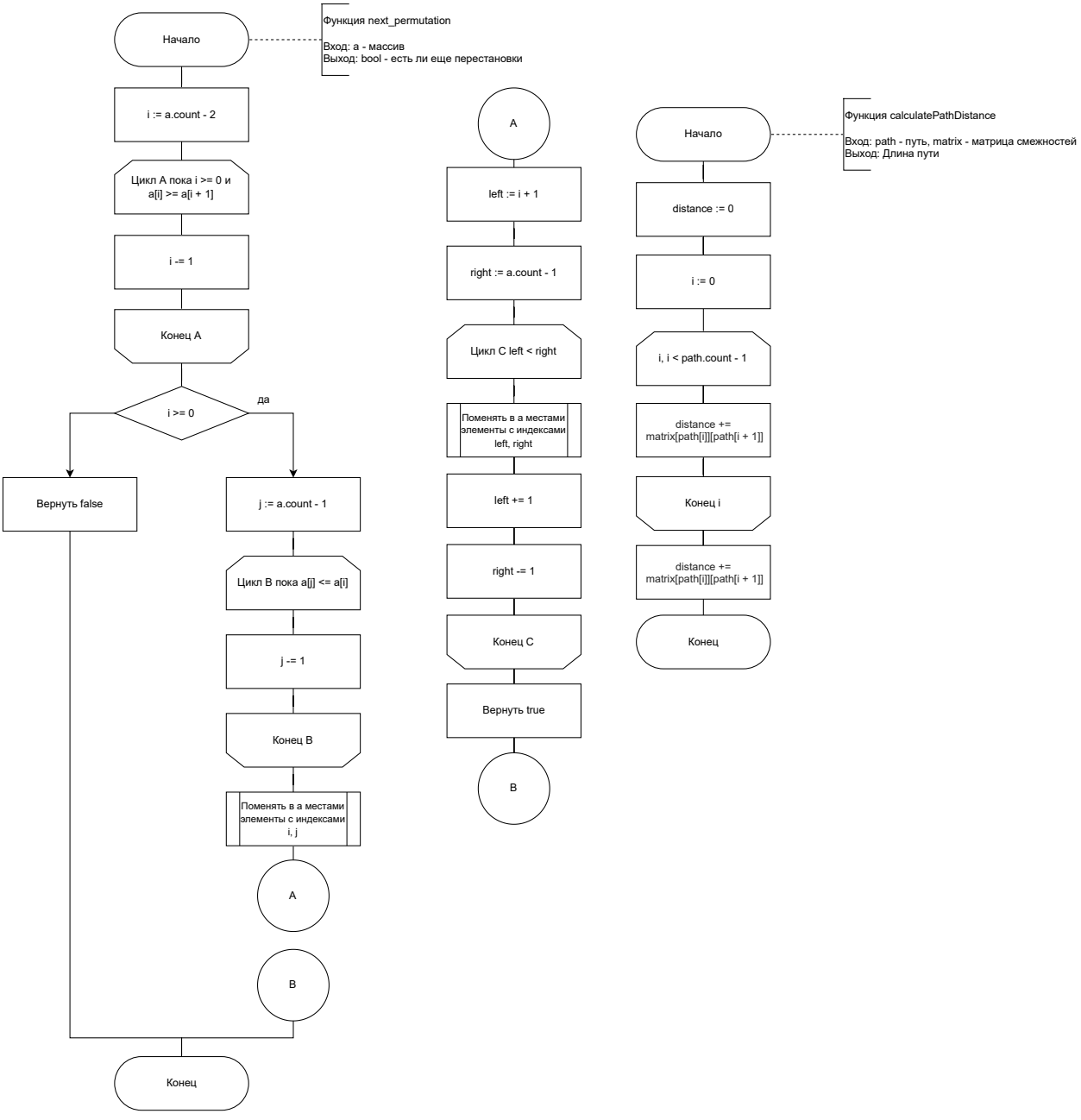

Рисунок 2.2 – Вспомогательные функции для метода полным перебором

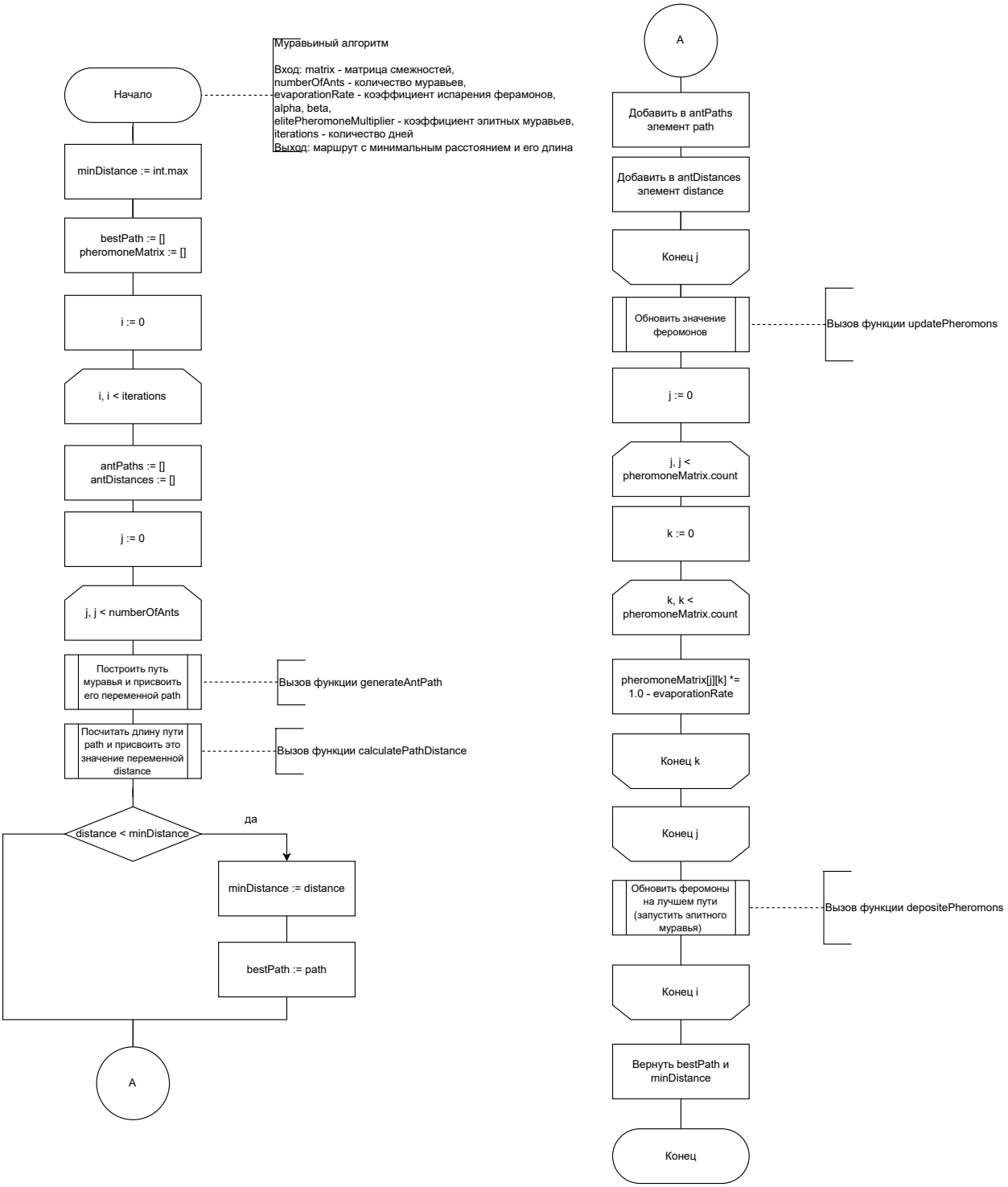

Рисунок 2.3 – Схема метода на основе муравьиного алгоритма

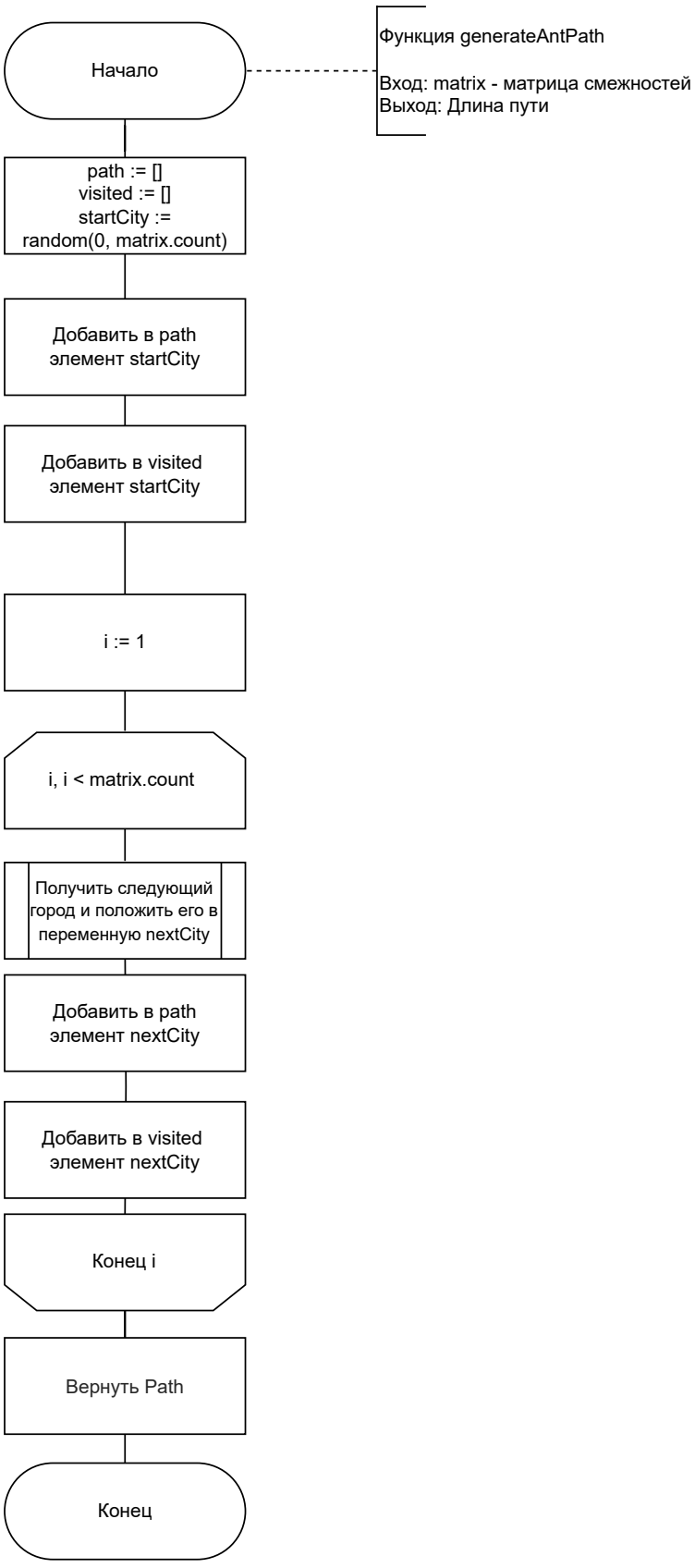

Рисунок 2.4 – Схема функции составления маршрута муравья

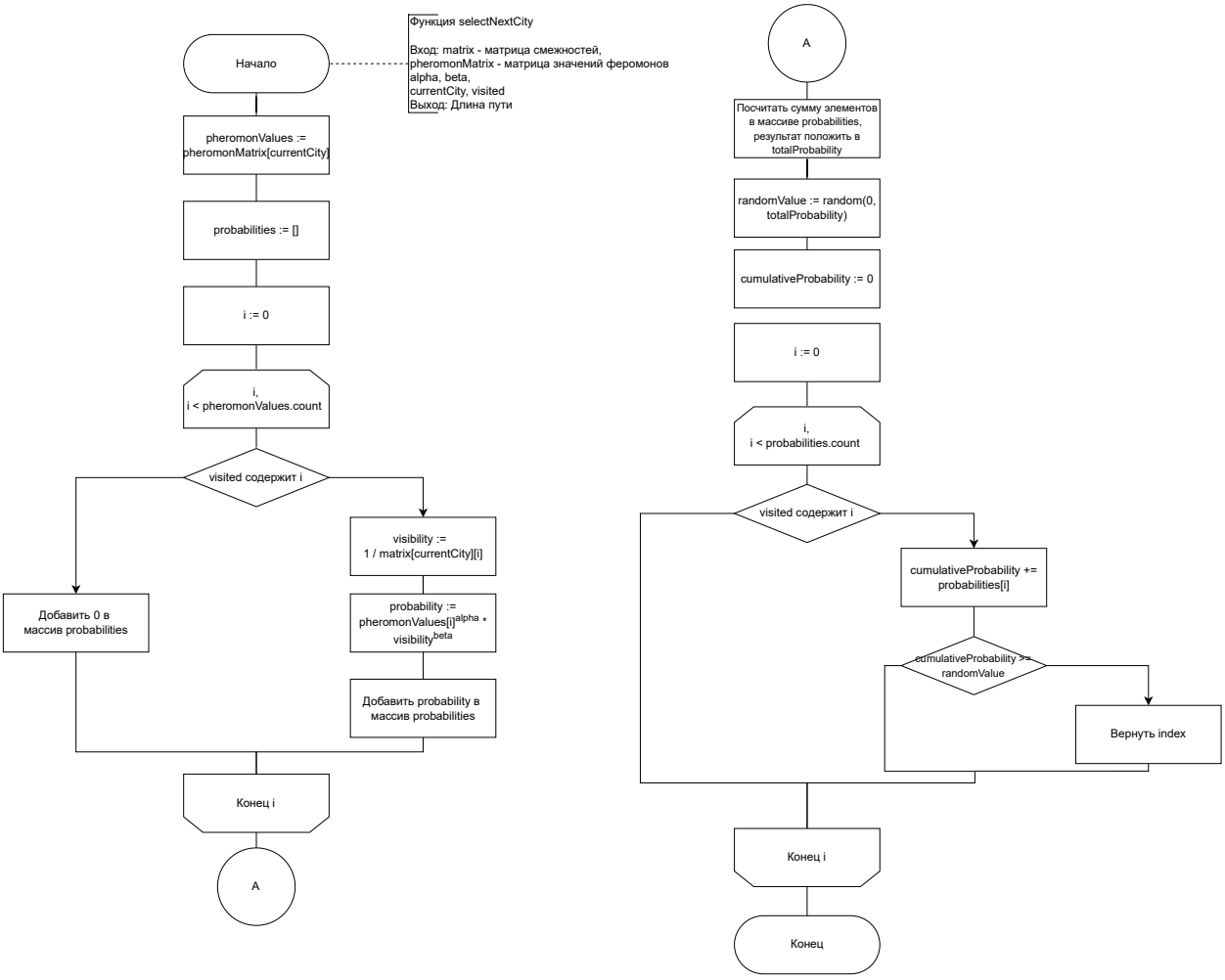

Рисунок 2.5 – Схема функции выбора следующего города

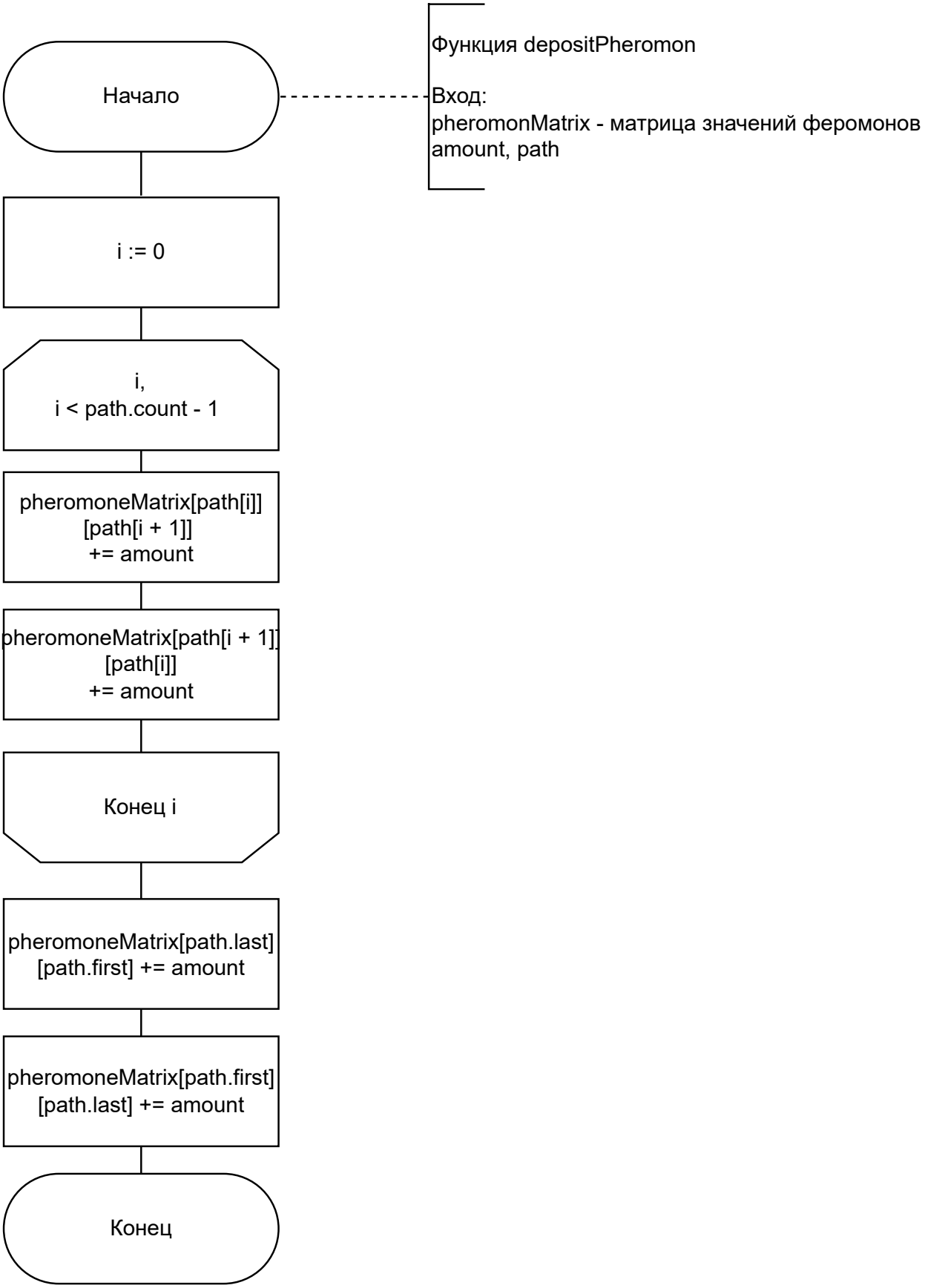

Рисунок 2.6 – Схема функции элитных муравьев

Вывод

В данном разделе были представлены схемы методов решения задачи коммивояжёра: полным перебором и на основе муравьиного алгоритма.

3 Технологическая часть

В данной главе описаны средства реализации, приведены листинги кода и функциональные тесты.

3.1 Средства реализации

Для разработки данной лабораторной работы был выбран язык программирования Swift [3]. Этот выбор обусловлен возможностью измерения процессорного времени [4] и соответствием с выдвинутыми техническими требованиям.

Измерение времени выполнения алгоритмов производится с использованием функции *clock_gettime()* [4].

3.2 Сведения о модулях программы

Программа разбита на следующие модули:

— `main.swift` — точка входа в программу, где происходит вызов алгоритмов через интерфейс;

— `Algorithms.swift` — содержит реализации методов решения задачи коммивояжёра (полным перебором и с использованием муравьиного алгоритма);

— `CPUTimeMeasure.swift` — измеряет время работы алгоритмов с учетом заданного количества повторений;

— `GraphRenderer.swift` — Строит графики для каждого из алгоритмов с учетом заданного количества повторений для каждого алгоритма;

3.3 Реализация алгоритмов

В листингах 3.1 – 3.8 приведены реализации методов решения задачи коммивояжёра: полным перебором (листинги 3.1 – 3.3) и с использованием муравьиного алгоритма (листинги 3.4 – 3.8).

В листинге 3.9 приведена реализация ввода матрицы.

Листинг 3.1 – Полный перебор

```
1     private static func bruteForce(distanceMatrix: [[Int]]) ->
          (path: [Int], distance: Int)? {
2         var minDistance = Int.max
3         var bestPath: [Int] = []
4
5         // Генерируем все возможные перестановки городов
6         var currentPath: [Int] = Array(0..<distanceMatrix.count)
7         repeat {
8             let currentDistance = calculatePathDistance(path:
                  currentPath, distanceMatrix: distanceMatrix)
9             if currentDistance < minDistance {
10                minDistance = currentDistance
11                bestPath = currentPath
12            }
13        } while next_permutation(&currentPath)
14
15        // Возвращаем лучший найденный маршрут и его длину
16        return (bestPath, minDistance)
17    }
```

Листинг 3.2 – Функция вычисления длины маршрута

```
1     private static func calculatePathDistance(path: [Int],
          distanceMatrix: [[Int]]) -> Int {
2         var distance = 0
3         for i in 0..<path.count - 1 {
4             distance += distanceMatrix[path[i]][path[i + 1]]
5         }
6         distance += distanceMatrix[path.last!][path.first!]
7         return distance
8     }
```

Листинг 3.3 – Функция получения нового маршрута

```
1     private static func next_permutation(_ a: inout [Int]) ->
          Bool {
2         var i = a.count - 2
3         while (i >= 0 && a[i] >= a[i + 1]) {
4             i -= 1
5         }
6
7         if i < 0 {
8             return false
9         }
10
11        var j = a.count - 1
12        while a[j] <= a[i] {
13            j -= 1
14        }
15
16        a.swapAt(i, j)
17
18        var left = i + 1
19        var right = a.count - 1
20
21        while left < right {
22            a.swapAt(left, right)
23            left += 1
24            right -= 1
25        }
26
27        return true
28    }
```

Листинг 3.4 – Муравьиный алгоритм

```
1     func runAntColony(iterations: Int) -> (path: [Int],
          distance: Int)? {
2         var globalBestPath: [Int] = []
3         var globalBestDistance = Int.max
4
5         for _ in 0..<iterations {
6             var antPaths: [[Int]] = []
7             var antDistances: [Int] = []
8             for _ in 0..<numberOfAnts {
9                 let path = generateAntPath()
10                let distance = calculatePathDistance(path: path)
11
12                // Обновление лучшего пути текущего муравья
13                if distance < globalBestDistance {
14                    globalBestPath = path
15                    globalBestDistance = distance
16                }
17                antPaths.append(path)
18                antDistances.append(distance)
19            }
20
21            updatePheromones(antPaths: antPaths, antDistances:
                  antDistances)
22
23            // Испарение феромонов
24            for i in 0..<pheromoneMatrix.count {
25                for j in 0..<pheromoneMatrix[i].count {
26                    pheromoneMatrix[i][j] *= (1.0 -
                          evaporationRate)
27                }
28            }
29
30            // Обновление феромонов на лучшем пути
31            depositPheromone(path: globalBestPath, amount:
                  elitePheromoneMultiplier /
                  Double(globalBestDistance))
32        }
33
34        return (globalBestPath, globalBestDistance)
35    }
```

Листинг 3.5 – Функция построения пути муравья

```
1     func generateAntPath() -> [Int] {
2         var path: [Int] = []
3         var visited: Set<Int> = []
4
5         // Начальный город
6         let startCity = Int.random(in: 0..<distanceMatrix.count)
7         path.append(startCity)
8         visited.insert(startCity)
9
10        // Построение пути
11        for _ in 1..<distanceMatrix.count {
12            let nextCity = selectNextCity(currentCity:
                  path.last!, visited: visited)
13            path.append(nextCity)
14            visited.insert(nextCity)
15        }
16
17        return path
18    }
```

Листинг 3.6 – Функция выбора следующего города

```
1     func selectNextCity(currentCity: Int, visited: Set<Int>) ->
          Int {
2         let pheromoneValues = pheromoneMatrix[currentCity]
3
4         // Вычисление вероятностей выбора следующего города
5         var probabilities: [Double] = []
6         for (index, pheromone) in pheromoneValues.enumerated() {
7             if !visited.contains(index) {
8                 let visibility = 1.0 /
                      Double(distanceMatrix[currentCity][index])
9                 let probability = pow(pheromone, alpha) *
                      pow(visibility, beta)
10                probabilities.append(probability)
11            } else {
12                probabilities.append(0.0)
13            }
14        }
15
16        // Выбор следующего города на основе вероятностей
17        let totalProbability = probabilities.reduce(0, +)
18        let randomValue = Double.random(in:
              0.0..<totalProbability)
19        var cumulativeProbability = 0.0
20
21        for (index, probability) in probabilities.enumerated() {
22            if !visited.contains(index) {
23                cumulativeProbability += probability
24                if cumulativeProbability >= randomValue {
25                    return index
26                }
27            }
28        }
29
30        // В случае ошибки возвращаем последний посещенный город
31        return visited.first!
32    }
```

Листинг 3.7 – Функция работы с элитным муравьем

```
    func depositPheromone(path: [Int], amount: Double) {
        for i in 0..<path.count - 1 {
            pheromoneMatrix[path[i]][path[i + 1]] += amount
            pheromoneMatrix[path[i + 1]][path[i]] += amount
        }
        pheromoneMatrix[path.last!][path.first!] += amount
        pheromoneMatrix[path.first!][path.last!] += amount
    }

    func updatePheromones(antPaths: [[Int]], antDistances:
       [Int]) {
        for i in 0..<pheromoneMatrix.count {
            for j in 0..<pheromoneMatrix[i].count {
                var pheromoneDeposit = 0.0
                for k in 0..<antPaths.count {
                    if antPaths[k].contains(i) &&
                       antPaths[k].contains(j) {
                        pheromoneDeposit += 1.0 /
                           Double(antDistances[k])
                    }
                }
                pheromoneMatrix[i][j] += pheromoneDeposit
                pheromoneMatrix[j][i] += pheromoneDeposit
            }
        }
    }
}
```

Листинг 3.8 – Функция обновления феромонов

```
1     func updatePheromones(antPaths: [[Int]], antDistances:
        [Int]) {
2         for i in 0..<pheromoneMatrix.count {
3             for j in 0..<pheromoneMatrix[i].count {
4                 var pheromoneDeposit = 0.0
5                 for k in 0..<antPaths.count {
6                     if antPaths[k].contains(i) &&
                        antPaths[k].contains(j) {
7                         pheromoneDeposit += 1.0 /
                            Double(antDistances[k])
8                     }
9                 }
10                pheromoneMatrix[i][j] += pheromoneDeposit
11                pheromoneMatrix[j][i] += pheromoneDeposit
12            }
13        }
14    }
```

Листинг 3.9 – Функция ввода матрицы

```
1     private static func inputMatrix() -> [[Int]]? {
2         print(Welcome.enterNumberOfCities)
3         let numCities = Int(readLine()!)
4         guard let numCities else {
5             print(Errors.invalidNumberInput)
6             return nil
7         }
8         guard numCities > 0 else {
9             print(Errors.belowZero)
10            return nil
11        }
12
13        var matrix = Array(repeating: Array(repeating: 0,
              count: numCities), count: numCities)
14        for i in 0..<numCities {
15            for j in 0..<numCities {
16                print("Введите элемент \(i + 1) строки, \(j +
                      1) столбца: ")
17                let value = Int(readLine()!)
18                guard let value else {
19                    print(Errors.invalidNumberInput)
20                    return nil
21                }
22                matrix[i][j] = value
23            }
24        }
25
26        return matrix
27    }
```

3.4 Функциональные тесты

В таблице приведены функциональные тесты для методов решения задачи коммивояжёра. Все тесты были успешно пройдены.

Таблица 3.1 – Тесты для умножения матриц

Матрица смежности	Ожидаемый результат
$\begin{pmatrix} 0 & 4 & 2 & 1 & 7 \\ 4 & 0 & 3 & 7 & 2 \\ 2 & 3 & 0 & 10 & 3 \\ 1 & 7 & 10 & 0 & 9 \\ 7 & 2 & 3 & 9 & 0 \end{pmatrix}$	15, [0, 2, 4, 1, 3, 0]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$	4, [0, 1, 2, 0]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$	64, [0, 1, 2, 3, 0]

Вывод

В данной главе были представлены требования к программному обеспечению, описаны средства реализации, приведены листинги кода алгоритмов и функциональные тесты, подтверждающие корректность работы алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Apple M1 Pro [5]

— Оперативная память: 32 ГБайт.

— Операционная система: macOS Ventura 13.5.2. [6]

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

4.2 Демонстрация работы программы

На изображении 4.1 представлена иллюстрация работы разработанного программного продукта.

```
Меню:
1. Решение задачи коммивояжёра
2. Измерить время
3. Построить графики
0. Выйти

Выберите пункт меню: 1
Введите количество городов
3
Введите элемент 1 строки, 1 столбца:
0
Введите элемент 1 строки, 2 столбца:
1
Введите элемент 1 строки, 3 столбца:
2
Введите элемент 2 строки, 1 столбца:
1
Введите элемент 2 строки, 2 столбца:
0
Введите элемент 2 строки, 3 столбца:
1
Введите элемент 3 строки, 1 столбца:
2
Введите элемент 3 строки, 2 столбца:
1
Введите элемент 3 строки, 3 столбца:
0
Введите коэффициент alpha
0.3
Введите коэффициент beta
0.7
Введите коэффициент усиления элитным муравьем
1.2
Введите количество дней
10
Полный перебор
Расстояние: 4, путь: [0, 1, 2]
Муравьиный алгоритм
Расстояние: 4, путь: [0, 1, 2]
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Анализ временных характеристик

В данном разделе представлены результаты экспериментов, в которых измерялось время решения задачи коммивояжёра. Данные результаты представлены в таблице 4.1.

Таблица 4.1 содержит результаты замеров времени выполнения методов решения задачи коммивояжёра: полным перебором и с использованием муравьиного алгоритма. Замеры проводились для размеров от 1 до 8. Все замеры проводились 10 раз. Брался усредненный результат.

Таблица 4.1 – Результаты замеров времени (массив уже отсортирован)

	Время, мс	
Размер матрицы	**Полный перебор**	**Муравьиный алгоритм**
1.0	0.080	10.448
2.0	0.151	29.716
3.0	0.409	61.887
4.0	1.673	106.480
5.0	8.903	168.573
6.0	61.217	241.343
7.0	486.975	334.548
8.0	4388.134	448.048

На основе данных из таблицы 4.1 был построен график (см. рисунок 4.2).

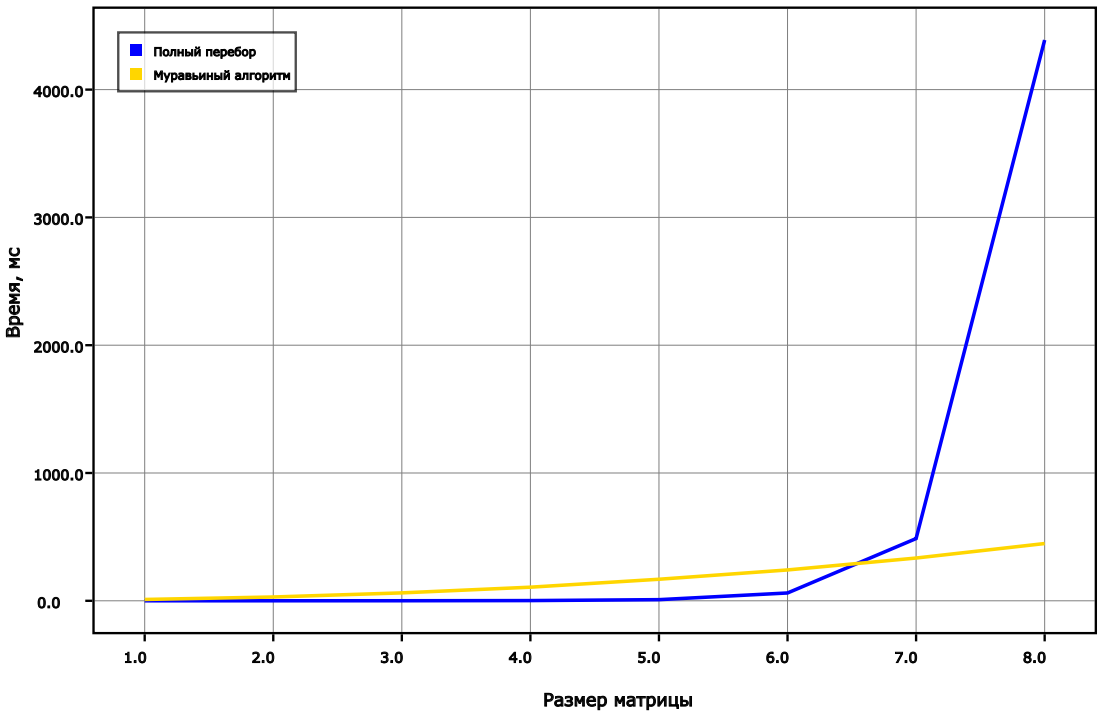

Рисунок 4.2 – Сравнение времени выполнения методов решения задачи коммивояжёра

4.4 Постановка эксперимента

Цель эксперимента – найти такие параметры, при которых для выбранного класса данных алгоритм будет возвращать наилучший результат, то есть такой, при котором величина ошибки минимальна. Таблица значений параметризации будет состоять из следующих значений:

1) α – коэффициент жадности;

2) ρ – коэффициент испарения;

3) $days$ – количество дней;

4) $elite$ – коэффициент усиления элитным муравьем;

5) $distance$ – результат, полученный муравьиным алгоритмом при данных параметрах;

6) $mistake$ – величина ошибки (разность эталонного результата, полученного методом полного перебора и значения $distance$)

4.4.1 Класс данных 1

Класс данных 1 представляет собой матрицу смежности 4.1 размером 10x10. Элементы матрицы - числа от 0 до 100

$$M_1 = \begin{pmatrix} 0 & 29 & 20 & 21 & 16 & 31 & 100 & 12 & 4 & 31 \\ 29 & 0 & 15 & 35 & 29 & 28 & 40 & 21 & 29 & 41 \\ 20 & 15 & 0 & 18 & 26 & 14 & 43 & 21 & 39 & 40 \\ 21 & 35 & 18 & 0 & 17 & 31 & 38 & 26 & 36 & 27 \\ 16 & 29 & 26 & 17 & 0 & 23 & 45 & 32 & 19 & 21 \\ 31 & 28 & 14 & 31 & 23 & 0 & 27 & 15 & 25 & 37 \\ 100 & 40 & 43 & 38 & 45 & 27 & 0 & 39 & 16 & 18 \\ 12 & 21 & 21 & 26 & 32 & 15 & 39 & 0 & 22 & 23 \\ 4 & 29 & 39 & 36 & 19 & 25 & 16 & 22 & 0 & 26 \\ 31 & 41 & 40 & 27 & 21 & 37 & 18 & 23 & 26 & 0 \end{pmatrix} \quad (4.1)$$

В таблице 4.2 представлена выборка параметров, при которых муравьиный алгоритм показывает наилучшие результаты.

Таблица 4.2 – Результаты параметризации на классе данных 1

α	ρ	$days$	$elite$	$distance$	$mistake$
0.00	0.00	10.00	0.90	164.00	0.00
0.00	0.00	50.00	0.00	164.00	0.00
0.00	0.00	50.00	0.60	164.00	0.00
0.00	0.00	100.00	1.20	164.00	0.00
0.00	0.10	50.00	2.00	164.00	0.00
0.00	0.10	100.00	0.00	164.00	0.00
0.00	0.10	100.00	0.50	164.00	0.00
0.00	0.10	100.00	0.80	164.00	0.00
0.00	0.10	100.00	1.10	164.00	0.00
0.00	0.10	100.00	1.40	164.00	0.00
0.00	0.10	100.00	1.50	164.00	0.00
0.00	0.20	50.00	0.30	164.00	0.00
0.00	0.20	50.00	0.70	164.00	0.00
0.00	0.30	50.00	0.50	164.00	0.00
0.00	0.30	50.00	0.70	164.00	0.00
0.00	0.30	100.00	1.10	164.00	0.00
0.00	0.30	100.00	1.20	164.00	0.00
0.00	0.40	100.00	0.60	164.00	0.00
0.00	0.40	100.00	0.70	164.00	0.00
0.00	0.50	50.00	1.40	164.00	0.00
0.00	0.50	50.00	2.00	164.00	0.00
0.10	0.00	100.00	0.10	164.00	0.00
0.10	0.10	100.00	0.20	164.00	0.00
0.10	0.20	100.00	0.00	164.00	0.00
0.10	0.30	100.00	0.80	164.00	0.00
0.10	0.40	100.00	0.60	164.00	0.00
0.10	0.50	50.00	0.20	164.00	0.00
0.10	0.50	100.00	0.10	164.00	0.00
0.10	0.60	100.00	0.20	164.00	0.00
0.20	0.00	50.00	1.80	164.00	0.00
0.20	0.10	50.00	0.20	164.00	0.00
0.20	0.10	100.00	0.20	164.00	0.00
0.20	0.20	50.00	0.40	164.00	0.00
0.20	0.20	100.00	0.00	164.00	0.00
0.20	0.30	50.00	1.30	164.00	0.00
0.20	0.60	10.00	1.90	164.00	0.00
0.20	0.60	50.00	0.00	164.00	0.00
0.30	0.00	100.00	0.50	164.00	0.00

Таблица 4.3 – Продолжение таблицы 4.2

α	ρ	$days$	$elite$	$distance$	$mistake$
0.30	0.10	50.00	0.70	164.00	0.00
0.30	0.30	10.00	0.40	164.00	0.00
0.30	0.40	50.00	0.20	164.00	0.00
0.40	0.10	10.00	1.00	164.00	0.00
0.40	0.20	50.00	0.10	164.00	0.00
0.50	0.00	100.00	0.30	164.00	0.00
0.50	0.10	100.00	0.80	164.00	0.00
0.50	0.40	100.00	0.90	164.00	0.00
0.50	0.60	10.00	1.90	164.00	0.00
0.50	0.70	100.00	0.60	164.00	0.00
0.60	0.10	50.00	0.50	164.00	0.00
0.60	0.40	100.00	0.10	164.00	0.00
0.60	0.60	50.00	0.30	164.00	0.00
0.60	0.70	1.00	2.00	164.00	0.00
0.60	0.70	100.00	0.70	164.00	0.00
0.70	0.00	10.00	0.90	164.00	0.00
0.70	0.10	100.00	0.60	164.00	0.00
0.70	0.20	50.00	1.60	164.00	0.00
0.70	0.50	10.00	1.60	164.00	0.00
0.70	0.70	100.00	1.70	164.00	0.00
0.80	0.00	100.00	1.40	164.00	0.00
0.80	0.50	100.00	1.70	164.00	0.00
0.80	0.70	100.00	1.70	164.00	0.00
0.90	0.20	100.00	1.60	164.00	0.00
0.90	0.30	100.00	0.50	164.00	0.00
0.90	0.30	100.00	1.90	164.00	0.00
0.90	0.50	100.00	0.80	164.00	0.00
1.00	0.60	5.00	0.90	164.00	0.00
1.00	0.70	50.00	1.70	164.00	0.00

4.4.2 Класс данных 2

Класс данных 2 представляет собой матрицу смежности 4.2 размером 10x10. Элементы матрицы - числа от 1000 до 10000

$$M_2 = \begin{pmatrix} 0 & 8200 & 6500 & 7300 & 4800 & 9100 & 10000 & 3200 & 4200 & 8900 \\ 8200 & 0 & 4500 & 9500 & 8100 & 7900 & 9200 & 6600 & 8600 & 9700 \\ 6500 & 4500 & 0 & 6700 & 7800 & 6400 & 9300 & 5200 & 9100 & 9400 \\ 7300 & 9500 & 6700 & 0 & 6200 & 8300 & 9600 & 7400 & 8400 & 9300 \\ 4800 & 8100 & 7800 & 6200 & 0 & 7200 & 9800 & 8500 & 5500 & 5700 \\ 9100 & 7900 & 6400 & 8300 & 7200 & 0 & 7900 & 6300 & 7300 & 8600 \\ 10000 & 9200 & 9300 & 9600 & 9800 & 7900 & 0 & 9700 & 6400 & 6700 \\ 3200 & 6600 & 5200 & 7400 & 8500 & 6300 & 9700 & 0 & 6800 & 7200 \\ 4200 & 8600 & 9100 & 8400 & 5500 & 7300 & 6400 & 6800 & 0 & 7500 \\ 8900 & 9700 & 9400 & 9300 & 5700 & 8600 & 6700 & 7200 & 7500 & 0 \end{pmatrix} \tag{4.2}$$

В таблице 4.4 представлена выборка параметров, при которых муравьиный алгоритм показывает наилучшие результаты.

Таблица 4.4 – Резальтаты параметризации на классе данных 2

α	ρ	$days$	$elite$	$distance$	$mistake$
0.00	0.00	5.00	0.70	57800.00	0.00
0.00	0.00	100.00	1.50	57800.00	0.00
0.00	0.10	50.00	0.90	57800.00	0.00
0.00	0.20	10.00	0.10	57800.00	0.00
0.00	0.40	50.00	1.70	57800.00	0.00
0.00	0.40	100.00	0.60	57800.00	0.00
0.00	0.40	100.00	0.70	57800.00	0.00
0.00	0.50	50.00	0.70	57800.00	0.00
0.00	0.50	100.00	0.20	57800.00	0.00
0.00	0.50	100.00	1.00	57800.00	0.00
0.00	0.50	100.00	1.10	57800.00	0.00
0.00	0.60	100.00	1.50	57800.00	0.00
0.10	0.10	50.00	0.90	57800.00	0.00
0.10	0.30	50.00	1.70	57800.00	0.00
0.10	0.40	100.00	1.20	57800.00	0.00
0.10	0.40	100.00	2.00	57800.00	0.00
0.10	0.50	100.00	0.70	57800.00	0.00
0.10	0.60	100.00	1.80	57800.00	0.00
0.10	0.70	50.00	0.50	57800.00	0.00
0.10	0.70	50.00	0.90	57800.00	0.00
0.10	0.70	100.00	0.90	57800.00	0.00
0.10	0.70	100.00	1.60	57800.00	0.00
0.10	0.70	100.00	1.70	57800.00	0.00
0.20	0.10	100.00	2.00	57800.00	0.00
0.20	0.20	50.00	1.60	57800.00	0.00
0.20	0.40	100.00	1.50	57800.00	0.00
0.20	0.50	50.00	1.80	57800.00	0.00
0.20	0.50	100.00	1.00	57800.00	0.00
0.20	0.60	100.00	0.50	57800.00	0.00
0.30	0.20	10.00	1.60	57800.00	0.00
0.30	0.30	100.00	0.40	57800.00	0.00
0.30	0.50	50.00	0.10	57800.00	0.00
0.40	0.00	100.00	0.40	57800.00	0.00
0.40	0.10	100.00	1.30	57800.00	0.00
0.40	0.20	100.00	1.00	57800.00	0.00
0.40	0.20	100.00	1.60	57800.00	0.00
0.40	0.30	100.00	0.00	57800.00	0.00
0.40	0.30	100.00	0.10	57800.00	0.00
0.40	0.50	5.00	0.70	57800.00	0.00
0.40	0.60	100.00	2.00	57800.00	0.00
0.40	0.70	50.00	0.00	57800.00	0.00

Таблица 4.5 – Продолжение таблицы 4.4

α	ρ	$days$	$elite$	$distance$	$mistake$
0.50	0.00	100.00	0.20	57800.00	0.00
0.50	0.20	50.00	1.70	57800.00	0.00
0.50	0.30	100.00	0.90	57800.00	0.00
0.50	0.50	100.00	1.80	57800.00	0.00
0.50	0.60	100.00	0.90	57800.00	0.00
0.50	0.70	5.00	1.00	57800.00	0.00
0.60	0.10	100.00	1.60	57800.00	0.00
0.60	0.30	50.00	1.50	57800.00	0.00
0.60	0.50	100.00	0.30	57800.00	0.00
0.70	0.00	100.00	0.20	57800.00	0.00
0.70	0.60	50.00	0.80	57800.00	0.00
0.70	0.70	50.00	1.50	57800.00	0.00
0.80	0.40	100.00	1.80	57800.00	0.00
0.90	0.00	50.00	2.00	57800.00	0.00
0.90	0.20	50.00	1.20	57800.00	0.00
0.90	0.60	100.00	1.40	57800.00	0.00
0.90	0.60	100.00	1.50	57800.00	0.00
1.00	0.00	50.00	0.80	57800.00	0.00
1.00	0.50	100.00	1.20	57800.00	0.00

4.5 Выводы

Исходя из полученных результатов, использование муравьиного алгоритма эффективнее по времени использования метода полным перебором при размере входной матрице больше 7. При этом при размере матрицы равным 7, муравьиный алгоритм эффективнее по времени в 1.5 раза, а при размере равным 8 – уже в 10 раз.

При проведении экспериментов с классом данных 1, муравьиный алгоритм показывает наилучшие результаты при следующих параметрах:

1) $\alpha = 0.0, \rho \in \{0.0, 0.1, 0.3\}$
2) $\alpha = 0.1, \rho = 0.5$
3) $\alpha = 0.2, \rho = 0.2$

Для класса данных 2 муравьиный алгоритм показывает наилучшие результаты при параметрах, которые представлены далее:

1) $\alpha = 0.0, \rho \in \{0.0, 0.4, 0.5\}$
2) $\alpha = 0.1, \rho \in \{0.4, 0.7\}$
3) $\alpha = 0.2, \rho = 0.5$
4) $\alpha = 0.4, \rho = 0.3$
5) $\alpha = 0.9, \rho = 0.6$

Именно такие параметры следуюет использовать в муравьином алгоритме на выбранных классах данных.

Заключение

Исходя из полученных результатов, рекомендуется использовать муравьиный алгоритм при размере входной матрицы большим 7. При размере матрицы равной 7, муравьиный алгоритм эффективнее метода полным перебором в 1.5 раза, а при размере матрицы, равной 8 – уже в 10 раз.

Основной целью данной лабораторной работы было изучение и описание методов решениия задачи коммивояжёра. Ниже представлены выполненные задачи.

1) Была описана задача коммивояжёра.
2) Был описан метод решения задачи полным перебором.
3) Был описан муравьиный алгоритм.
4) Была разработана программа, реализующая описанные методы решения.
5) Были проанализированы затраты времени работы программы.

Список использованных источников

1 О. Борознов В. Исследование решения задачи коммивояжера. — АГТУ, Вестник Астраханского государственного технического университета. [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera/viewer (дата обращения: 26.11.2023).

2 Семёнов С. С. Педан А. В. Воловиков В. С. Климов И. С. Анализ трудоёмкости различных алгоритмических подходов для решения задачи коммивояжёра — ООО «Корпорация «Интел Групп» [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/analiz-trudoemkosti-razlichnyh-algoritmicheskih-podhodov-dlya-\resheniya-zadachi-kommivoyazhera (дата обращения: 23.01.2023).

3 Документация к Swift [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/swift (дата обращения: 12.09.2023).

4 Измерение процессорного времени [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man3/clock_gettime.3.html (дата обращения: 14.09.2023).

5 Процессоры M1 [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/apple-silicon (дата обращения: 15.09.2023).

6 Операционная система macOS. [Электронный ресурс]. — Режим доступа: https://developer.apple.com/macos/ (дата обращения: 15.09.2023).