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 №7

по курсу «Анализ алгоритмов»

на тему:

«Методы решения задачи коммивояжера»

Студент **Голикова С. М.**

Группа **ИУ7-55Б**

Оценка (баллы)

Преподаватели **Волкова Л. Л., Строганов Ю. В.**

Москва, 2022г.

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.

В рамках выполнения работы необходимо решить следующие задачи:

— описать и реализовать алгоритм полного перебора для решения задачи коммивояжера;
— описать и реализовать муравьиный алгоритм для решения задачи коммивояжера;
— провести параметризацию муравьиного алгоритма на двух классах данных;
— провести сравнительный анализ времени выполнения и трудоемкостей реализаций.

1 Аналитический раздел

В данном разделе представлено теоретическое описание задачи коммивояжера, а также алгоритма полного перебора и муравьиного алгоритма для ее решения.

1.1 Задача коммивояжера

В задаче коммивояжера 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 Алгоритм полного перебора

Суть алгоритма полного перебора для решения задачи коммивояжера заключается в переборе всех вариантов путей и нахождении кратчайшего. Данный алгоритм имеет факториальную сложность $O(n!)$, что приводит к большим временным затратам даже при малых значениях числа вершин в графе.

1.3 Муравьиный алгоритм

Моделирование поведения муравьев связано с распределением феромона на тропе — ребре графа в задаче коммивояжера. Более короткие пути сильнее обогащаются феромоном, вследствие чего являются более предпочтительными для всей колонии. С помощью моделирования испарения феромона, т.е.

отрицательной обратной связи, гарантируется, что найденное локально оптимальное решение не будет единственным: будут предприняты попытки поиска других путей.

Опишем правила поведения муравья при выборе пути применительно к решению задачи коммивояжера [1]:

— муравьи обладают «памятью» в виде списка уже посещенных городов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью k, находящемуся в городе i;

— муравьи обладают «зрением». Видимость есть эвристическое желание посетить город j, если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами i и j, обозначенному через D_{ij}. Так, $\eta_{ij} = \frac{1}{D_{ij}}$;

— муравьи обладают «обонянием». Они могут улавливать след феромона, подтверждающий желание посетить город j из города i, на основании опыта других муравьев. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{ij}(t)$.

Вероятность перехода из города i в город j определяется по формуле (1.1).

$$P_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}, \tag{1.1}$$

где $\tau_{i,j}$ — количество феромонов на ребре от i до j; $\eta_{i,j}$ — эвристическое расстояние от i до j; α — параметр влияния расстояния; 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 движения всеми муравьями происходит обновление феромона.

Если L_k — длина пути k-ого муравья, Q — некоторая константа порядка длины путей, N — количество муравьев, $p \in [0, 1]$ — коэффициент испаре-

ния феромона, то новое значения феромона на ребре (i, j) рассчитывается по формуле (1.2).

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

где

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & \text{ребро посещено } k\text{-ым муравьем,} \\ 0, & \text{иначе.} \end{cases} \tag{1.3}$$

Вывод

В данном разделе были рассмотрены идеи, необходимые для разработки и реализации двух алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

2 Конструкторский раздел

В данном разделе приведены разработанные алгоритмы решения задачи коммивояжера: алгоритм полного перебора и муравьиный алгоритм.

2.1 Разработка алгоритмов

На рисунке 1 приведена схема алгоритма полного перебора для решения задачи коммивояжера, а на рисунках 2–4 — схема муравьиного алгоритма.

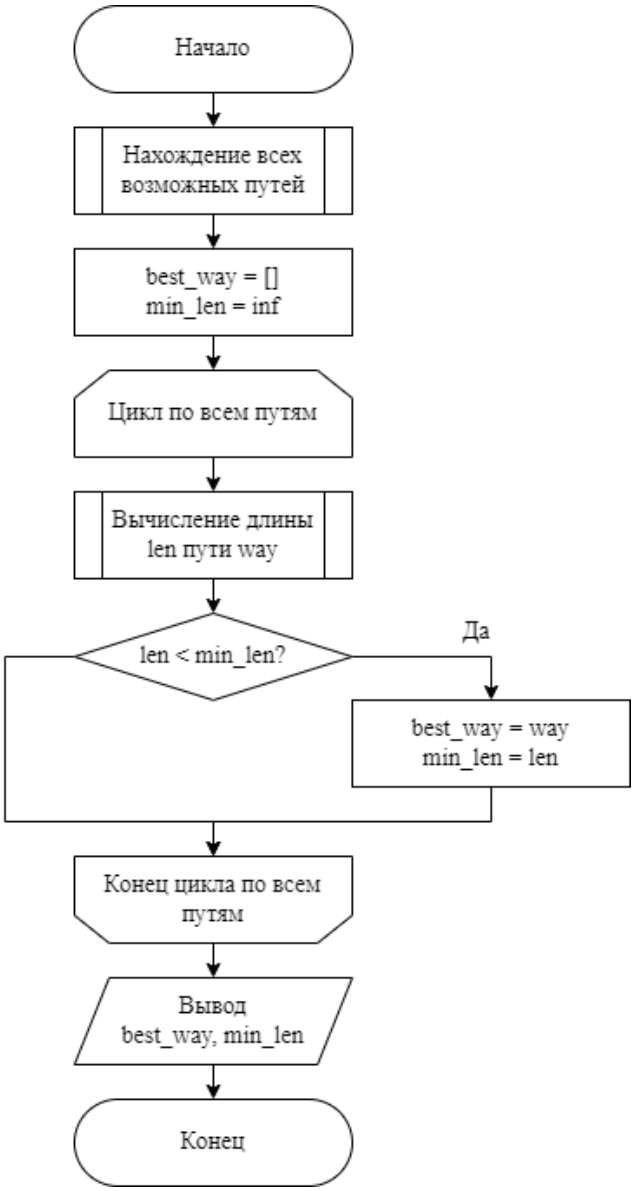

Рисунок 1 — Схема алгоритма полного перебора

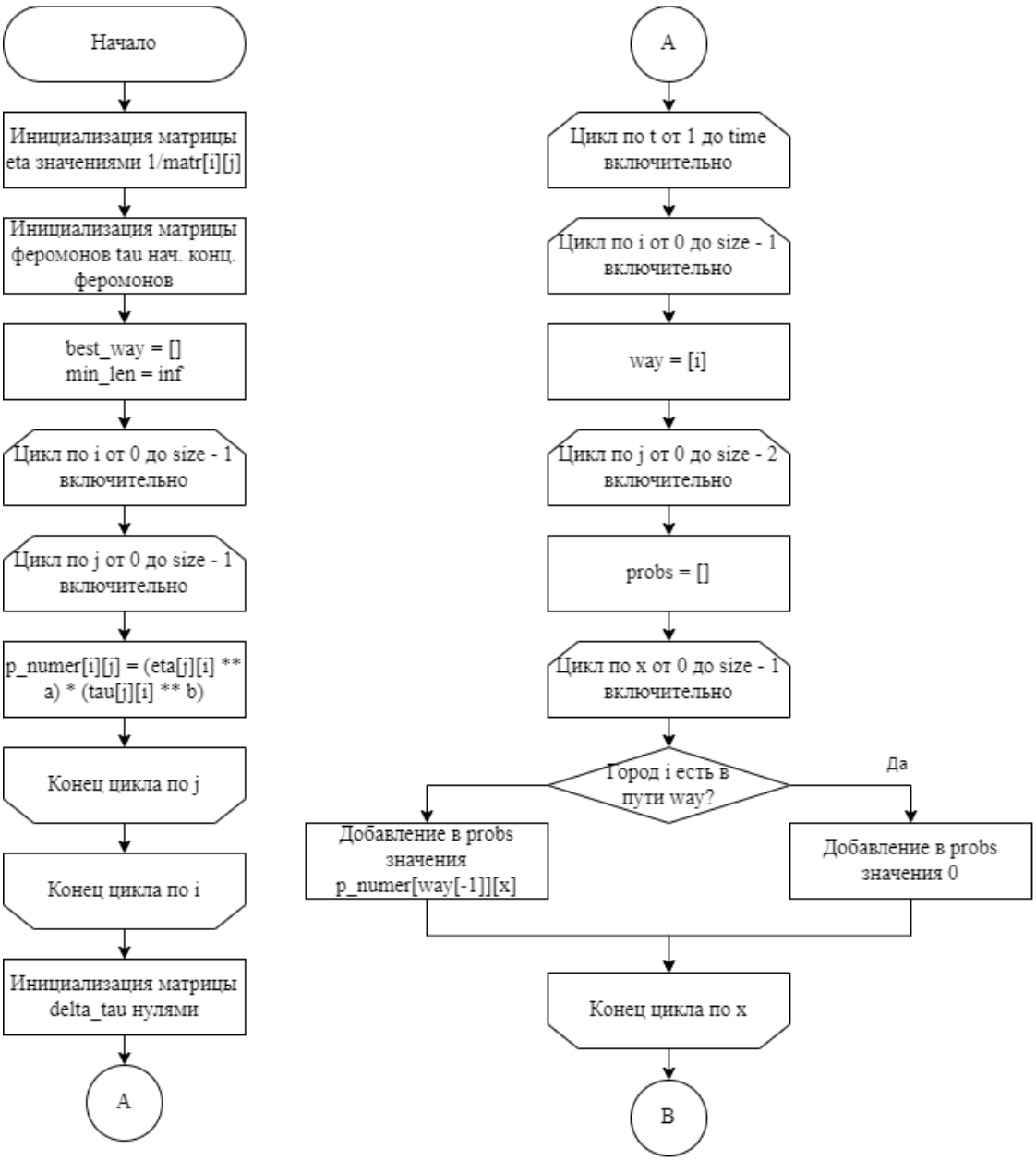

Рисунок 2 — Схема муравьиного алгоритма

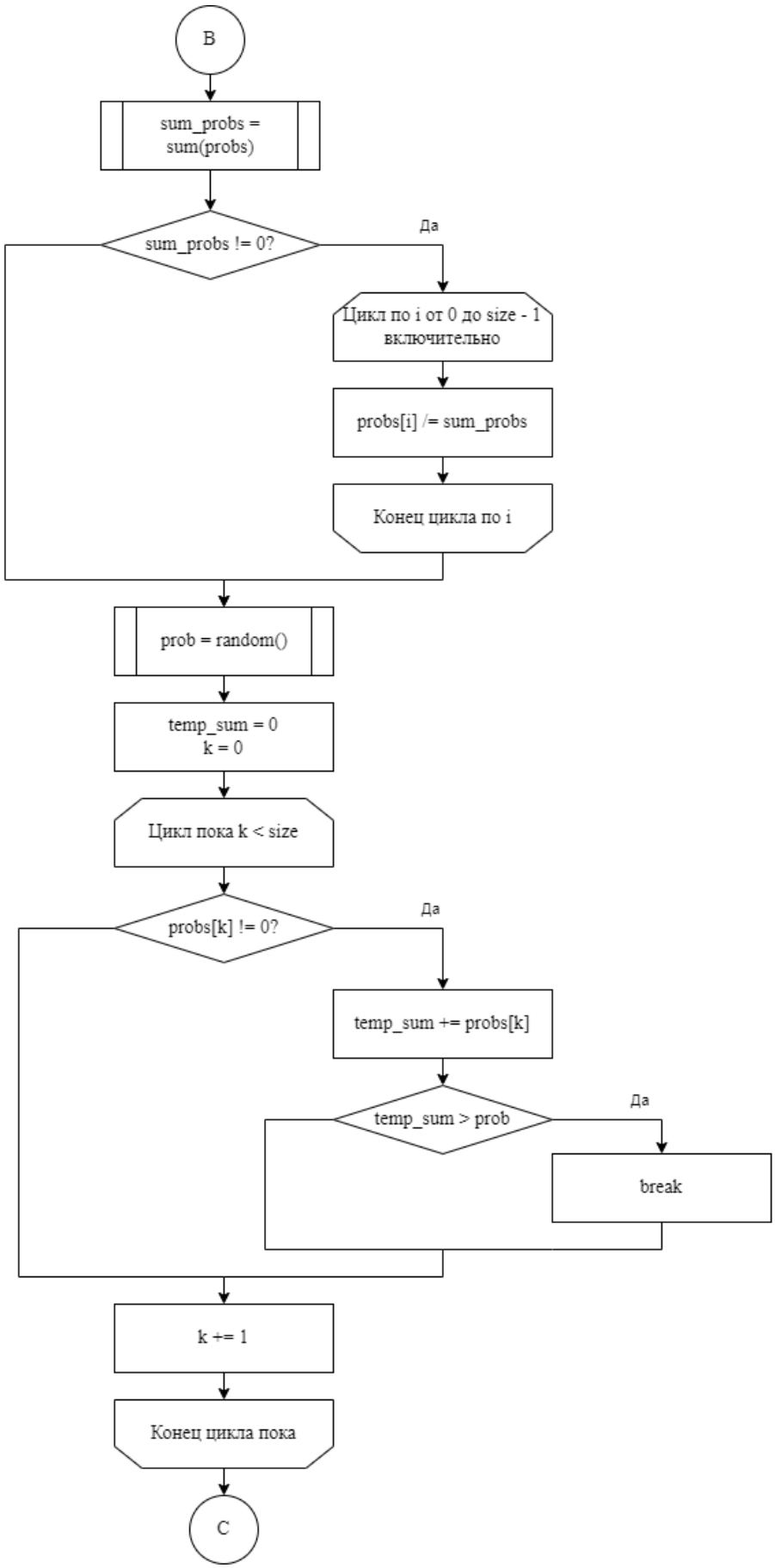

Рисунок 3 — Схема муравьиного алгоритма (продолжение)

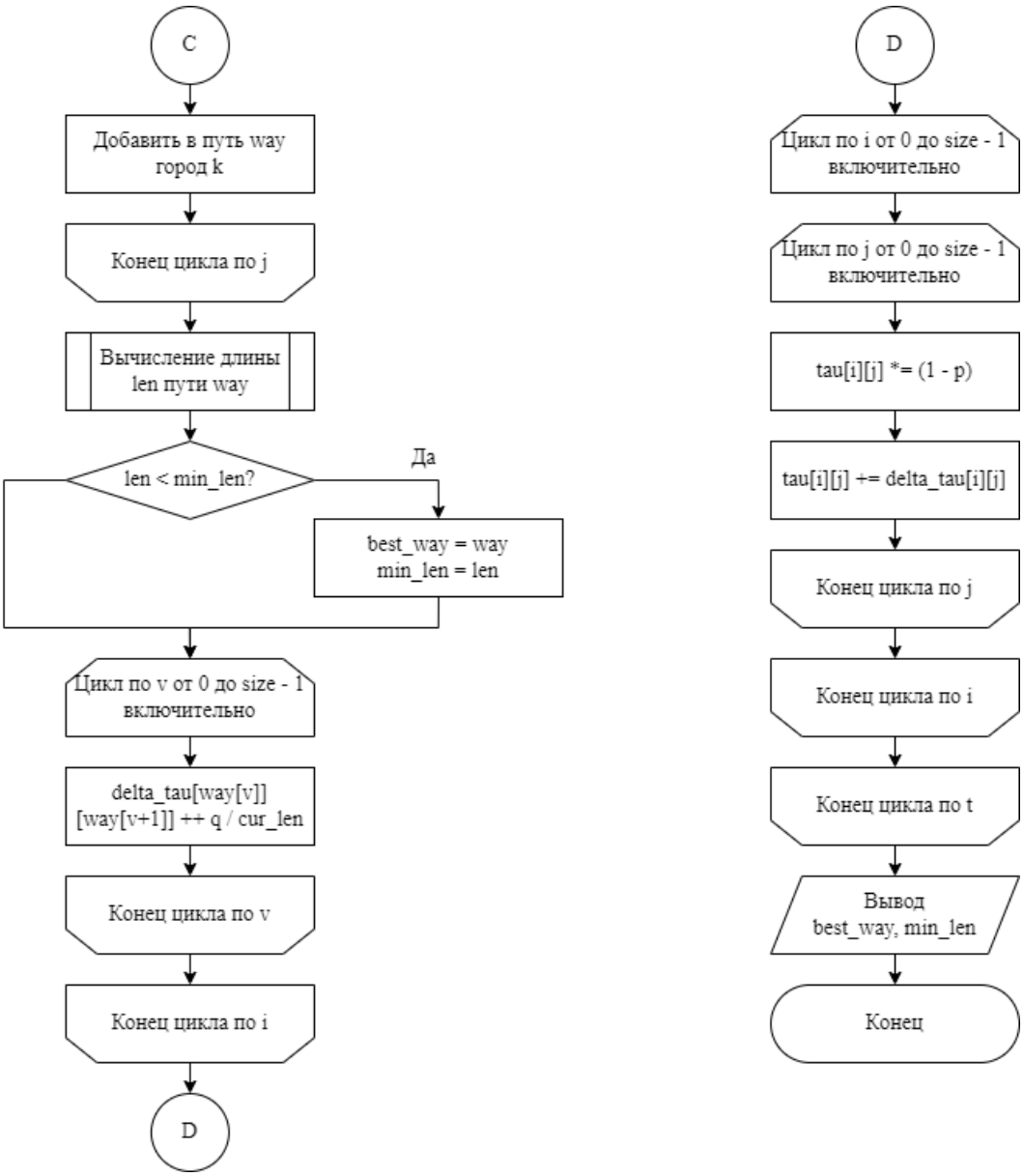

Рисунок 4 — Схема муравьиного алгоритма (продолжение)

2.2 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

— выбор файла с матрицей расстояний;

— ввод параметров α, p и t_{max} для муравьиного алгоритма;

— вывод на экран найденного каждым из алгоритмов кратчайшего пути и его длины;

— измерение времени работы реализаций алгоритмов на матрицах разных

размеров и представление результата в виде графика;

— вывод результатов параметризации муравьиного алгоритма в текстовый файл.

Вывод

Были разработаны алгоритмы для решения задачи коммивояжера: алгоритм полного перебора и муравьиный алгоритм.

3 Технологический раздел

В данном разделе приводятся описание средств реализации и сами реализации алгоритмов, а также функциональные тесты.

3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран Python [4]. Данный язык предоставляет возможности работы с массивами и матрицами.

Для замера процессорного времени использовалась функция process_time библиотеки time [5].

3.2 Реализации алгоритмов

В листинге 1 приведена реализация алгоритма полного перебора для решения задачи коммивояжера. В листингах 2—5 приведена реализация муравьиного алгоритма для решения задачи коммивояжера.

Листинг 1 — Реализация алгоритма полного перебора

```
def full_search(matrix, size):
    all_ways = permutations(range(size))
    best_way = []
    min_len = float("inf")
    for way in all_ways:
        way = list(way) + [way[0]]
        cur_len = 0
        for i in range(size):
            cur_len += matrix[way[i]][way[i + 1]]
        if cur_len < min_len:
            best_way = way
            min_len = cur_len

    return best_way, min_len
```

Листинг 2 — Вычисление вероятностей перехода в следующий город

```
def calc_prob(p_numerator, cur_city, size, root):
    res_probs = []

    for i in range(size):
        if i in root:
            res_probs.append(0)
        else:
            cur_prob = p_numerator[cur_city][i]
            res_probs.append(cur_prob)

    sum_probs = sum(res_probs)

    if sum_probs:
        for i in range(size):
            res_probs[i] /= sum_probs

    return res_probs
```

Листинг 3 — Обновление матрицы обоняния tau

```
def update_tau(tau):
    for i in range(size):
        for j in range(size):
            tau[i][j] *= (1 - p)
            tau[i][j] += delta_tau[i][j]
```

Листинг 4 — Итерация муравьиного алгоритма

```
def ant_iteration(matr, eta, tau, size, a, b, q, p, best, min_len):
    p_numerator = [[0] * size for _ in range(size)]
    for i in range(size):
        for j in range(size):
            p_numerator[i][j] = (eta[j][i] ** a) * (tau[j][i] ** b)
    delta_tau = [[0] * size for _ in range(size)]
    for i in range(size):
        way = [i]
        for j in range(size - 1):
            temp_probs = calc_prob(p_numerator, way[-1], size, way)
            prob = random()
            temp_sum, k = 0, 0
            while k < size:
                if temp_probs[k]:
                    temp_sum += temp_probs[k]
                    if temp_sum > prob:
                        break
                k += 1
            if k == size:
                k = size - 1
            way.append(k)
        way.append(way[0])
        cur_len = 0
        for x in range(len(way) - 1):
            cur_len += matr[way[x]][way[x + 1]]
        if cur_len < min_len:
            best = way
            min_len = cur_len
        for v in range(size):
            delta_tau[way[v]][way[v + 1]] += q / cur_len
    update_tau(tau)

    return tau, best_way, min_len
```

Листинг 5 — Реализация муравьиного алгоритма

```
def ant_algorithm(matrix, size, a, b, q, p, time):
    eta = [[0] * size for _ in range(size)]
    for i in range(size):
        for j in range(i):
            eta[i][j] = 1 / matrix[i][j]
            eta[j][i] = 1 / matrix[j][i]
    tau = [[0.2] * size for _ in range(size)]

    best_way = []
    min_len = float("inf")

    for t in range(time):
        tau, best_way, min_len = ant_iteration(matrix, eta, tau,
            size, a, b, q, p, best_way, min_len)

    return best_way, min_len
```

3.3 Тестирование

В таблице 1 приведены функциональные тесты для реализаций алгоритмов, решающих задачу коммивояжера.

Тесты пройдены успешно каждой реализацией.

Таблица 1 — Функциональное тестирование

Матрица смежности	Кратчайший путь	Длина пути
$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	[0, 1, 0]	2
$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$	[0, 1, 2, 3, 0]	6
$\begin{pmatrix} 0 & 1 & INF & 2 \\ 1 & 0 & 2 & INF \\ INF & 2 & 0 & 3 \\ 2 & INF & 3 & 0 \end{pmatrix}$	[0, 1, 2, 3, 0]	8

Вывод

Были описаны требования к программному обеспечению, средства реализации, приведены реализации алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

В данном разделе приводится пример работы программы, производится сравнение трудоемкостей и времени выполнения реализаций алгоритмов, а также приводятся результаты параметризации муравьиного алгоритма на двух классах данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система Windows 10 [6];

— память 8 ГБ;

— процессор Intel® Core™ i5-6260U @ 1.80 ГГц [7].

Замеры времени выполнения реализаций алгоритмов проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только приложением и средой разработки.

4.2 Демонстрация работы программы

На рисунке 5 представлен пример работы программы. Пользователь вводит название файла с матрицей расстояний и параметры для муравьиного алгоритма. Программа выводит кратчайший путь и его длину, полученные двумя способами: полным перебором и с помощью муравьиного алгоритма.

```
Введите название файла с матрицей расстояний между городами: test.txt
=============================================================================================================
Исходная матрица:
0 1 2 3
1 0 1 2
2 1 0 1
3 2 1 0
=============================================================================================================
ПОЛНЫЙ ПЕРЕБОР
Минимальный путь:  [0, 1, 2, 3, 0]
Длина:  6
=============================================================================================================
Введите параметр alpha: 0.9
Введите коэффициент испарения феромона p: 0.2
Введите максимальное число итераций t: 100
=============================================================================================================
МУРАВЬИНЫЙ АЛГОРИТМ
Минимальный путь:  [1, 3, 2, 0, 1]
Длина:  6
```

Рисунок 5 — Пример работы реализаций алгоритмов

4.3 Сравнение трудоемкостей реализаций

Задача коммивояжера является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность $O(n!)$.

Сложность муравьиного алгоритма равна $O(t_{max} \cdot m \cdot n^2)$, то есть она зависит от времени жизни колонии, количества городов и количества муравьев в колонии [8]. В разработанной реализации количество муравьев равно количеству городов, и трудоемкость муравьиного алгоритма равна $O(t_{max} \cdot n^3)$.

4.4 Сравнение времени выполнения реализаций алгоритмов

Измеряется время работы реализаций алгоритмов в зависимости от размера матрицы. Параметры для муравьиного алгоритма следующие: $\alpha = 1$; $p = 0.5$; $t\_max = 1000$.

На рисунке 6 приведены результаты сравнения времени выполнения реализаций алгоритмов.

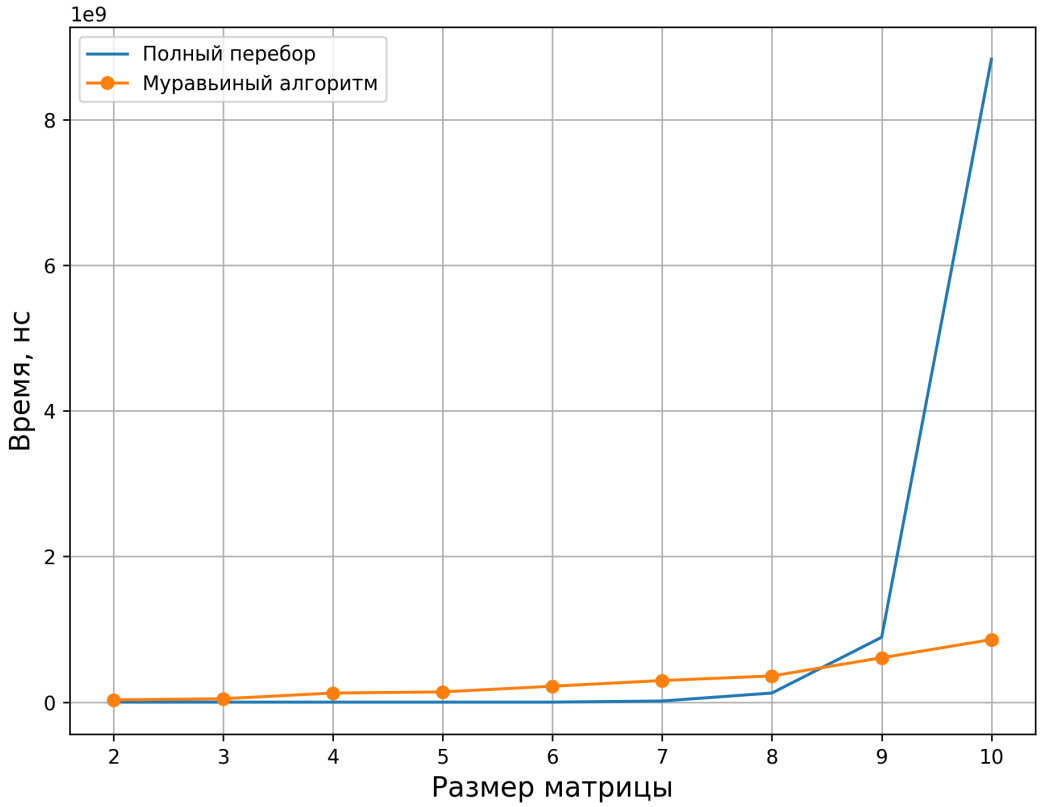

Рисунок 6 — Зависимость времени работы реализаций от размера матрицы расстояний

4.5 Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления проводятся с использованием таких настраиваемых параметров, как α — коэффициент жадности, p — коэффициент испарения феромона, t_{max} — время жизни колонии, количество итераций. Параметризация алгоритма подразумевает подбор таких параметров, при которых алгоритм находит решение, наиболее близкое к эталонному — найденному с помощью полного перебора.

Параметризация проводится на двух классах данных: в первом (small) разброс длин путей равен 10, во втором (large) — 1000.

Рассматриваются матрицы размерности 11 × 11. Для параметров α и p независимо друг от друга задаются значения [0.1, 0.25, 0.5, 0.75, 0.9], а для t_max — [100, 200, 300, 400, 500].

Результаты параметризации приведены в таблице А.1 приложения А, где $diff\_x$ — разница между эталонным решением и решением, полученным с помощью муравьиного алгоритма на классе данных x.

Результаты параметризации в зависимости от количества итераций приведены в таблице 2.

Таблица 2 — Зависимость точности от количества итераций

Количество итераций	Количество ошибок		Средняя ошибка	
	small	large	small	large
100	13	13	1.20	123.04
200	11	9	0.84	87.04
300	7	6	0.56	48.48
400	6	8	0.32	48.44
500	2	1	0.08	6.92

С ростом количества итераций количество ошибок и средняя ошибка падают для обоих классов данных.

От значений α и p зависимостей не обнаружено, что говорит о том, что значения параметров должны регулироваться для каждой конкретной задачи. Лучшие параметры — это параметры, при которых значения ошибок минимальны для обоих классов данных. Например, в рассматриваемом случае лучшими комбинациями аргументов на количестве итераций 100 являются (0.5, 0.1), (0.5, 0.75), (0.75, 0.25), (0.9, >0.1), где первое значение — α, а второе — p.

Вывод

Применимость алгоритмов зависит от того, насколько велик размер матрицы расстояний. При размерах $N < 9$ реализация алгоритма полного перебора работает быстрее, чем реализация муравьиного алгоритма: например, при

$N = 8$ — в 4 раза. Но при дальнейшем увеличении размера матрицы реализация муравьиного алгоритма начинает выигрывать по времени. Так, при $N = 10$ реализация муравьиного алгоритма превосходит реализацию перебора в 13 раз.

Заранее проведенная параметризация может помочь настроить муравьиный алгоритм так, что и он будет в большинстве случаев выдавать точный ответ, несмотря на то, что алгоритм предназначен для приближенного решения задачи коммивояжера.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были решены следующие задачи:

— описан и реализован алгоритм полного перебора для решения задачи коммивояжера;

— описан и реализован муравьиный алгоритм для решения задачи коммивояжера;

— проведена параметризация муравьиного алгоритма на двух классах данных;

— проведен сравнительный анализ времени выполнения и трудоемкостей реализаций.

Применимость алгоритмов зависит от того, насколько велик размер матрицы расстояний. При размерах $N < 9$ реализация алгоритма полного перебора работает быстрее, чем реализация муравьиного алгоритма, но при дальнейшем увеличении размера матрицы реализация муравьиного алгоритма начинает выигрывать по времени.

Заранее проведенная параметризация может помочь настроить муравьиный алгоритм так, что и он будет в большинстве случаев выдавать точный ответ, несмотря на то, что алгоритм предназначен для приближенного решения задачи коммивояжера.

Поставленная цель была достигнута: было разработано программное обеспечения, решающее задачу коммивояжера двумя способами: полным перебором и с помощью муравьиного алгоритма.

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. Лутц, Марк. Изучаем Python, том 1, 5-е изд. — Пер. с англ. – СПб.: ООО “Диалектика”, 2019. — 832 с.

5. time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 22.09.2022).

6. Windows client documentation for IT Pros [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/windows/resources/ (дата обращения: 22.09.2022).

7. Процессор Intel® Core™ i5-6260U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/91160/intel-core-i56260u-processor-4m-cache-up-to-2-90-ghz.html (дата обращения: 22.09.2022).

8. Ульянов, М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ // М.: Наука-Физматлит. — 2007. — С. 201–-207.

ПРИЛОЖЕНИЕ А

Результаты параметризации муравьиного алгоритма

Таблица А.1 — Параметризация

α	p	t_max	diff_small	diff_large
0.1	0.1	100	2	523
0.1	0.25	100	4	0
0.1	0.5	100	2	254
0.1	0.75	100	5	398
0.1	0.9	100	6	358
0.25	0.1	100	2	0
0.25	0.25	100	2	208
0.25	0.5	100	2	0
0.25	0.75	100	0	335
0.25	0.9	100	0	261
0.5	0.1	100	0	0
0.5	0.25	100	1	0
0.5	0.5	100	1	0
0.5	0.75	100	0	0
0.5	0.9	100	0	166
0.75	0.1	100	1	204
0.75	0.25	100	0	0
0.75	0.5	100	0	55
0.75	0.75	100	1	55
0.75	0.9	100	0	55
0.9	0.1	100	1	204
0.9	0.25	100	0	0

Продолжение таблицы А.1

α	p	t_max	diff_small	diff_large
0.9	0.5	100	0	0
0.9	0.75	100	0	0
0.9	0.9	100	0	0
0.1	0.1	200	4	316
0.1	0.25	200	5	208
0.1	0.5	200	1	488
0.1	0.75	200	2	0
0.1	0.9	200	3	204
0.25	0.1	200	1	0
0.25	0.25	200	0	166
0.25	0.5	200	0	166
0.25	0.75	200	1	231
0.25	0.9	200	1	231
0.5	0.1	200	0	166
0.5	0.25	200	0	0
0.5	0.5	200	0	0
0.5	0.75	200	0	0
0.5	0.9	200	0	0
0.75	0.1	200	0	0
0.75	0.25	200	1	0
0.75	0.5	200	0	0
0.75	0.75	200	0	0
0.75	0.9	200	0	0
0.9	0.1	200	0	0
0.9	0.25	200	1	0

Продолжение таблицы A.1

α	p	t_max	diff_small	diff_large
0.9	0.5	200	1	0
0.9	0.75	200	0	0
0.9	0.9	200	0	0
0.1	0.1	300	1	335
0.1	0.25	300	2	55
0.1	0.5	300	3	239
0.1	0.75	300	0	0
0.1	0.9	300	4	355
0.25	0.1	300	2	55
0.25	0.25	300	0	0
0.25	0.5	300	0	0
0.25	0.75	300	0	173
0.25	0.9	300	0	0
0.5	0.1	300	1	0
0.5	0.25	300	0	0
0.5	0.5	300	0	0
0.5	0.75	300	0	0
0.5	0.9	300	0	0
0.75	0.1	300	1	0
0.75	0.25	300	0	0
0.75	0.5	300	0	0
0.75	0.75	300	0	0
0.75	0.9	300	0	0
0.9	0.1	300	0	0
0.9	0.25	300	0	0

Продолжение таблицы А.1

α	p	t_max	diff_small	diff_large
0.9	0.5	300	0	0
0.9	0.75	300	0	0
0.9	0.9	300	0	0
0.1	0.1	400	2	166
0.1	0.25	400	0	386
0.1	0.5	400	1	231
0.1	0.75	400	1	55
0.1	0.9	400	2	208
0.25	0.1	400	0	0
0.25	0.25	400	0	0
0.25	0.5	400	1	55
0.25	0.75	400	0	55
0.25	0.9	400	0	0
0.5	0.1	400	0	0
0.5	0.25	400	0	0
0.5	0.5	400	0	0
0.5	0.75	400	0	0
0.5	0.9	400	0	0
0.75	0.1	400	0	0
0.75	0.25	400	0	0
0.75	0.5	400	0	55
0.75	0.75	400	0	0
0.75	0.9	400	0	0
0.9	0.1	400	0	0
0.9	0.25	400	1	0

Продолжение таблицы А.1

α	p	t_max	diff_small	diff_large
0.9	0.5	400	0	0
0.9	0.75	400	0	0
0.9	0.9	400	0	0
0.1	0.1	500	0	0
0.1	0.25	500	0	0
0.1	0.5	500	1	173
0.1	0.75	500	0	0
0.1	0.9	500	0	0
0.25	0.1	500	0	0
0.25	0.25	500	0	0
0.25	0.5	500	0	0
0.25	0.75	500	0	0
0.25	0.9	500	0	0
0.5	0.1	500	0	0
0.5	0.25	500	1	0
0.5	0.5	500	0	0
0.5	0.75	500	0	0
0.5	0.9	500	0	0
0.75	0.1	500	0	0
0.75	0.25	500	0	0
0.75	0.5	500	0	0
0.75	0.75	500	0	0
0.75	0.9	500	0	0
0.9	0.1	500	0	0
0.9	0.25	500	0	0

Продолжение таблицы А.1

α	p	t_max	diff_small	diff_large
0.9	0.5	500	0	0
0.9	0.75	500	0	0
0.9	0.9	500	0	0