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"

Тема Муравьиный алгоритм

Студент Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение **3**

1 Аналитическая часть **4**

1.1 Задача коммивояжера 4

1.2 Алгоритм полного перебора для решения задачи коммивояжера 4

1.3 Муравьиный алгоритм для решения задачи коммивояжера . 4

1.4 Вывод из аналитической части 6

2 Конструкторская часть **7**

2.1 Схема алгоритма полного перебора 7

2.2 Схема муравьиного алгоритма 12

2.3 Вывод из конструкторской части 13

3 Технологическая часть **14**

3.1 Требования к ПО 14

3.2 Выбор средств реализации 14

3.3 Листинги кода 14

3.4 Тестирование 19

4 Исследовательская часть **21**

4.1 Технические характеристики 21

4.2 Сравнение времени выполнения реализаций алгоритмов . . . 21

4.3 Анализ статистики параллельного конвейера 22

4.4 Вывод из исследовательской части 24

Заключение **25**

Список использованной литературы **26**

Введение

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев. Колония представляет собой систему с очень простыми правилами автономного поведения особей. Однако, несмотря на примитивность поведения каждого отдельного муравья, поведение всей колонии оказывается достаточно разумным. Эти принципы проверены временем — удачная адаптация к окружающему миру на протяжении миллионов лет означает, что природа выработала очень удачный механизм поведения [1].

Целью данной работы является реализация муравьиного алгоритма для решения задачи коммивояжера и приобретение навыков параметризации алгоритмов.

В рамках выполнения работы необходимо решить следующие задачи:

1) реализовать алгоритм полного перебора для решения задачи коммивояжера;

2) изучить и реализовать муравьиный алгоритм для решения задачи коммивояжера;

3) провести параметризацию муравьиного алгоритма на трех классах данных и подобрать оптимальные параметры;

4) провести сравнительный анализ трудоемкостей реализаций.

1 Аналитическая часть

В данном разделе будет приведена теория, необходимая для разработки и реализации двух алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

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 [2].

1.2 Алгоритм полного перебора для решения задачи коммивояжера

Суть алгоритма полного перебора для решения задачи коммивояжера заключается в переборе всех вариантов путей и нахождении кратчайшего. Преимуществом является его результат - точное решение, недостатком - длительность вычислений при относительно небольшом пространстве поиска [3]. Эта задача является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность [1].

1.3 Муравьиный алгоритм для решения задачи коммивояжера

Моделирование поведения муравьев связано с распределением феромона на тропе — ребре графа в задаче коммивояжера. При этом вероятность

включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его ребрах. При этом избежать преждевременной сходимости можно, моделируя отрицательную обратную связь в виде испарения феромона. С учетом особенностей задачи коммивояжера, мы можем описать локальные правила поведения муравьев при выборе пути.

- Муравьи имеют собственную «память» 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 — D_{ij}: $\eta_{ij} = \frac{1}{D_{ij}}$.

- 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 $\tau_{ij}(t)$.

На этом основании мы можем сформулировать вероятностно-пропорционально правило 1.1, определяющее вероятность перехода k-ого муравья из города i в город j:

$$P_{ij,k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum\limits_{l \in J_{i,k}} (\tau_{il}(t))^{\alpha}(\eta_{il}(t))^{\beta}}, j \in J_{i,k}, \\ 0, \text{иначе} \end{cases}, \tag{1.1}$$

где α— параметр влияния длины пути, при $\alpha = 0$ алгоритм вырождается до жадного алгоритма (будет выбран ближайший город), β— параметр влияния феромона.

Пройдя ребро (i, j), муравей откладывает на нем некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , а $L_k(t)$ — длина этого маршрута. Пусть также Q — параметр,

имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде 1.2:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i,j) \in T_k(t), \\ 0, \text{иначе.} \end{cases} \tag{1.2}$$

Правила внешней среды определяют, в первую очередь, испарение феромона. Пусть $p \in [0,1]$ есть коэффициент испарения, тогда правило испарения имеет вид 1.3:

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

где m — количество муравьев в колонии.

В начале алгоритма количество феромона на ребрах принимается равным небольшому положительному числу. При этом необходимо следить, чтобы количество феромона на существующем ребре не обнулилось в ходе испарения. Общее количество муравьев остается постоянным и равным количеству городов, каждый муравей начинает маршрут из своего города.

1.4 Вывод из аналитической части

Были рассмотрены идеи и материалы, необходимые для разработки и реализации двух алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритма полного перебора и муравьиного алгоритма для решения задачи коммивояжера.

2.1 Схема алгоритма полного перебора

На рисунке 2.1 приведена схема вспомогательной функции next_set, которая позволяет сгенерировать все возможные перестановки первых n неотрицательных чисел.

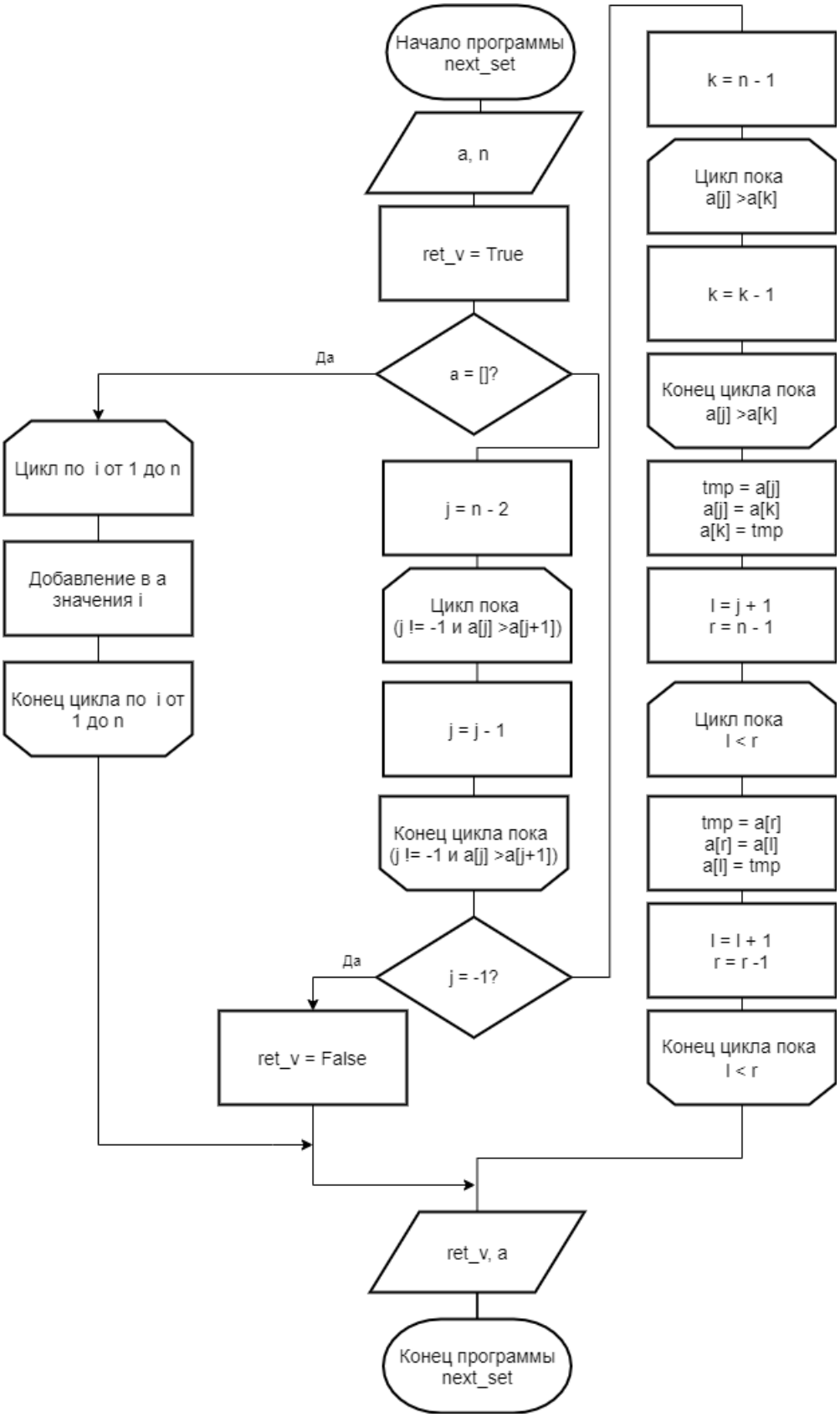

Рисунок 2.1 – Схема вспомогательной функции next_set

На рисунке 2.2 приведена схема вспомогательной функции count_way_len, которая с помощью матрицы смежности D вычисляет длину пути при проходе по городам visited_cities.

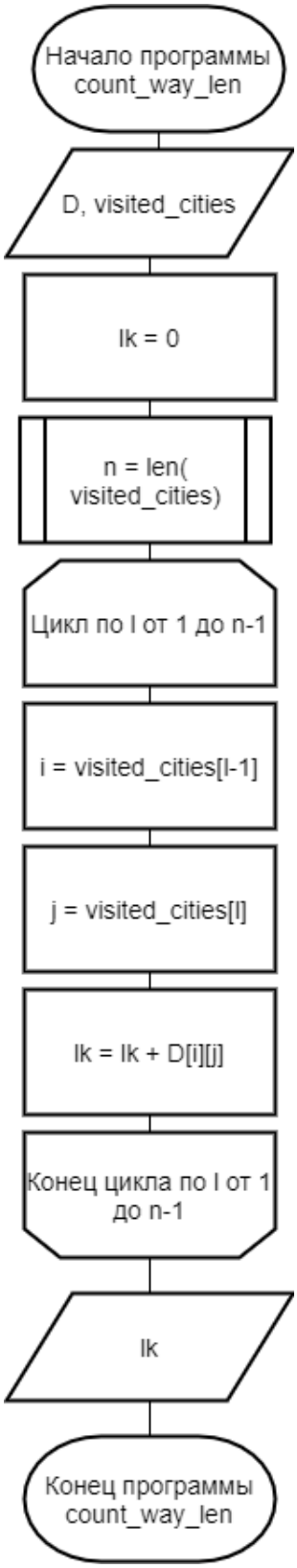

Рисунок 2.2 – Схема вспомогательной функции count_way_len

На рисунке 2.3 приведена схема алгоритма полного перебора для решения задачи коммивояжера.

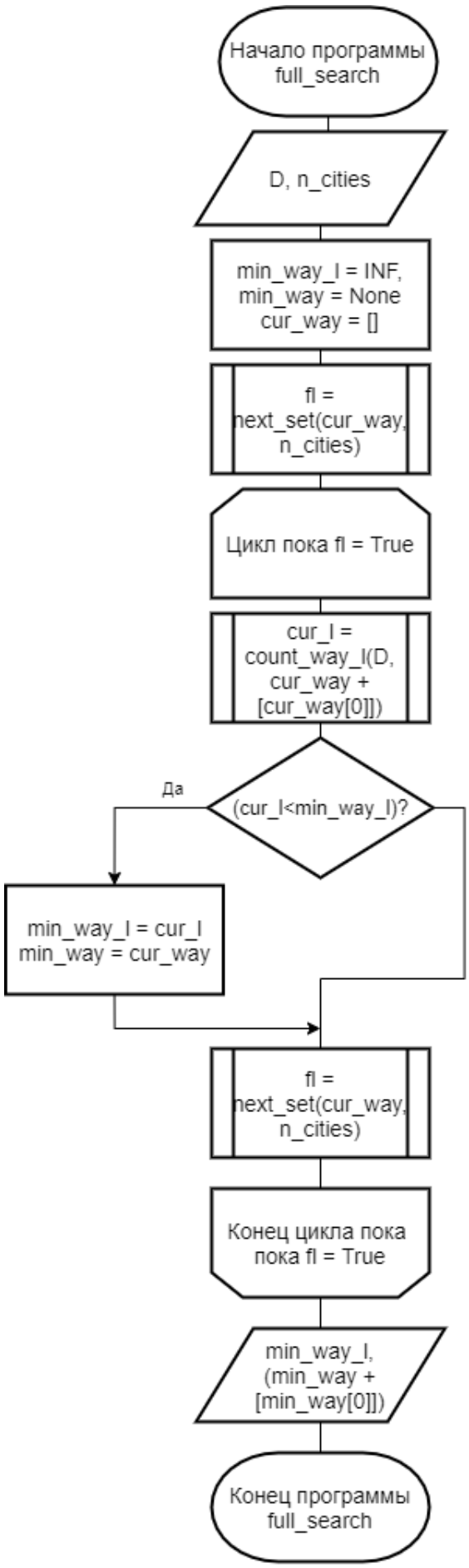

Рисунок 2.3 – Схема алгоритма полного перебора

2.2 Схема муравьиного алгоритма

На рисунке 2.4 приведена схема муравьиного алгоритма для решения задачи коммивояжера.

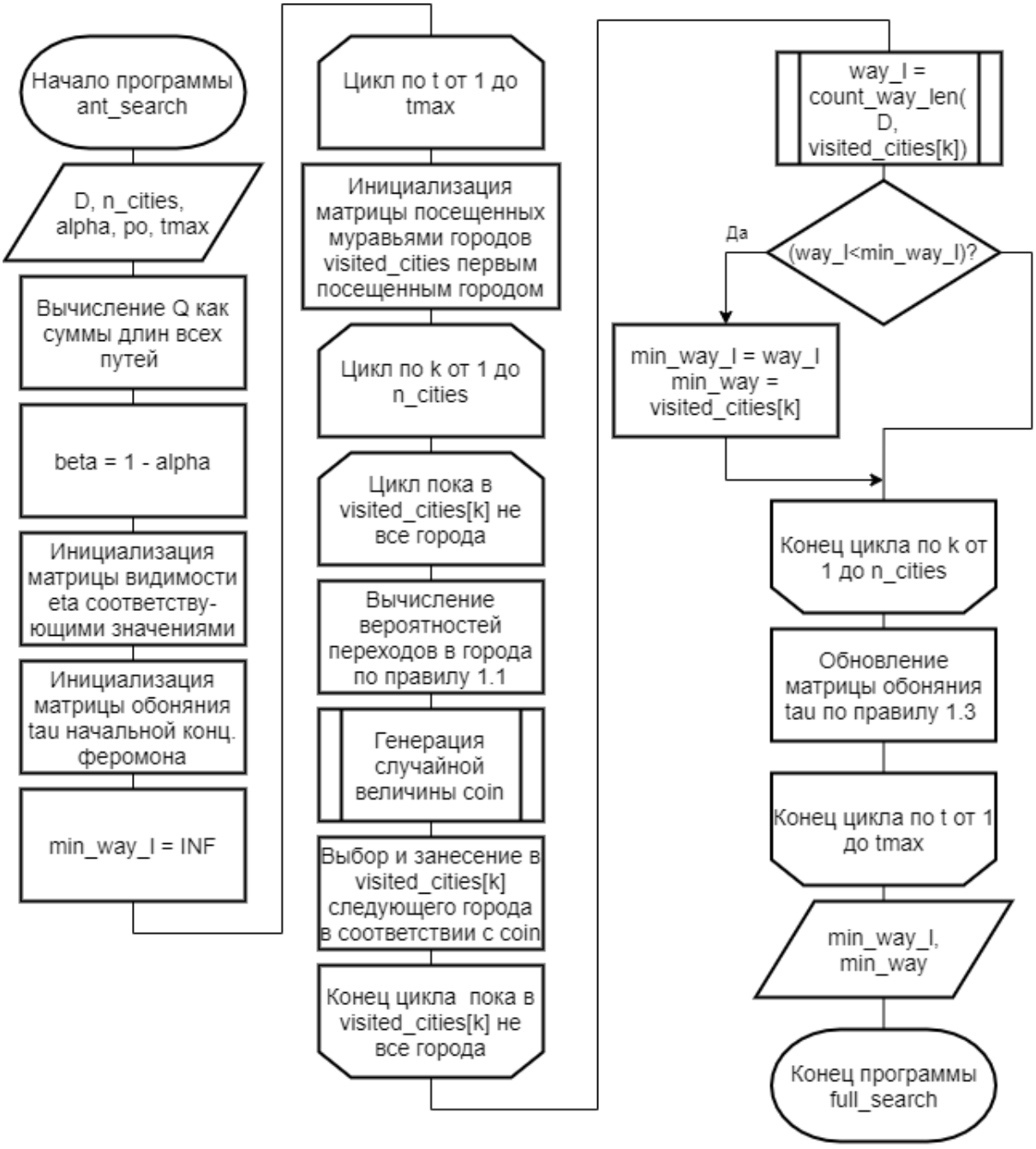

Рисунок 2.4 – Схема муравьиного алгоритма

2.3 Вывод из конструкторской части

Были приведены схемы разрабатываемых алгоритмов.

3 Технологическая часть

В данном разделе производится выбор средств реализации, приведятся требования к ПО, листинги реализованных алгоритмов решения задачи коммивояжера, а также результаты их тестированиия.

3.1 Требования к ПО

На вход программе подаются количество городов и симметричная матрица смежности, а также параметры α (коэффициент жадности), ро (коэффициент испарения) и tmax (время жизни колонии) для муравьиного алгоритма. На выходе должны быть получены решения задачи коммивояжера, найденые с помощью каждого реализованного алгоритма: алгоритма полного перебора и муравьиного алгоритма.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Python [4]. Он позволяет быстро реализовывать различные алгоритмы без выделения большого времени на проектирование сруктуры программы и выбор типов данных.

Кроме того, в Python есть библиотека time, которая предоставляет функцию process_time для замера процессорного времени [5].

В качестве среды разработки выбран PyCharm. Он является кроссплатформенным, а также предоставляет удобный и функциональнаый отладчик и средства для рефакторинга кода, что позволяет быстро находить и исправлять ошибки [6].

3.3 Листинги кода

В листинге 3.1 представлена реализация алгоритма полного перебора для решения задачи коммивояжера.

Листинг 3.1 – Алгоритм полного перебора

```
def full_search(D, n_cities):
    min_way_length = INF
    min_way = None
    cur_way = []

    while next_set(cur_way, n_cities):
        cur_way_lenth = count_way_lenth(D, cur_way + [cur_way[0], ])
        if cur_way_lenth < min_way_length:
            min_way_length = cur_way_lenth
            min_way = cur_way

    return min_way_length, min_way + [min_way[0]]
```

В листинге 3.2 представлена реализация муравьиного алгоритма для решения задачи коммивояжера.

Листинг 3.2 – Муравьиный алгоритм

```
def ant_search(D, n_cities, alpha=ALPHA, po=PO, tmax=TMAX):
  Q = 0
  for i in range(n_cities):
    for j in range(i):
      if D[i][j] < INF:
        Q += D[i][j]
  beta = 1 - alpha

  eta = [[0 for i in range(n_cities)] for j in range(n_cities)]
  tau = [[0 for i in range(n_cities)] for j in range(n_cities)]
  for i in range(n_cities):
    for j in range(i):
      eta[i][j] = 1 / D[i][j]
      eta[j][i] = 1 / D[j][i]
      tau[i][j] = 2 * EPS
      tau[j][i] = 2 * EPS

  min_way_length = INF


  for t in range(tmax):
    visited_cities = [[i] for i in range(n_cities)]

    for k in range(n_cities):

      while len(visited_cities[k]) != n_cities:
        P_ch = [0 for i in range(n_cities)]
        for j in range(n_cities):
          if j not in visited_cities[k]:
            i = visited_cities[k][-1]
            P_ch[j] = (tau[i][j] ** alpha) * (eta[i][j] ** beta)

        P_zn = sum(P_ch)
        for j in range(n_cities):
          P_ch[j] /= P_zn

        coin = random()
        summ, j = 0, 0
```

```
            while summ < coin:
                summ += P_ch[j]
                j += 1
            visited_cities[k].append(j - 1)

        visited_cities[k].append(visited_cities[k][0])
        way_length = count_way_lenth(D, visited_cities[k])


        if way_length < min_way_length:
            min_way_length = way_length
            min_way = visited_cities[k]


    for i in range(n_cities):
        for j in range(i):
            delta_tau = 0

            for k in range(n_cities):
                way_length = count_way_lenth(D, visited_cities[k])
                for m in range(1, len(visited_cities[k])):
                    if (visited_cities[k][m], visited_cities[k][m - 1])
                        in ((i, j), (j, i)):
                        delta_tau += Q / way_length
                        break

        tau[i][j] = tau[i][j] * (1 - po) + delta_tau
        if tau[i][j] < EPS:
            tau[i][j] = EPS

    return min_way_length, min_way
```

В листинге 3.3 представлены реализации вспомогательных функций.

Листинг 3.3 – Вспомогательные функции

```
1
2  def next_set(a, n):
3      if not a:
4          for i in range(n):
5              a.append(i)
6          return True
7
8      j = n - 2
9      while j != -1 and a[j] > a[j + 1]:
10         j -= 1
11     if j == -1:
12         return False
13
14     k = n - 1
15     while a[j] > a[k]:
16         k -= 1
17     a[j], a[k] = a[k], a[j]
18
19     l = j + 1
20     r = n - 1
21     while l < r:
22         a[l], a[r] = a[r], a[l]
23         l += 1
24         r -= 1
25     return True
26
27 def count_way_lenth(D, visited_cities):
28     lk = 0
29     for l in range(1, len(visited_cities)):
30         i = visited_cities[l - 1]
31         j = visited_cities[l]
32         lk += D[i][j]
33
34     return lk
```

3.4 Тестирование

На рисунке 3.1 приведен пример работы программы.

```
Введите количество городов: 5
Введите матрицу смежности:
  0 1 3 4 5
  1   0 2 3 4
  3   2   0 4 5
  4   3   4   0 1
  5   4   5   1   0
Введите параметр alpha: 0.9
Введите параметр po: 0.2
Введите параметр tmax: 50
Матрица смежности:
  0   1   3   4   5
  1   0   2   3   4
  3   2   0   4   5
  4   3   4   0   1
  5   4   5   1   0
Алгоритм полного перебора: ответ=13.0, путь=[4, 3, 2, 1, 0, 4]
Муравьиный алгоритм: ответ=13.0, путь=[2, 1, 0, 4, 3, 2]
```

Рисунок 3.1 – Пример работы программы

В таблице 3.1 приведены функциональные тесты для алгоритмов решения задачи коммивояжера:алгоритма полного перебора и муравьиного алгоритма. Тест считается успешно пройденным, если длина найденного пути совпадает с ожидаемой, а в найденном пути созраняется ожидаемая последовательность городов (то есть город, с которого начинается поссследовательность, не является принципиальным). Все тесты пройдены успешно каждым алгоритмом.

Таблица 3.1 – Тестирование функций

Матрица смежности	Ожидаемый наименьший путь
$\begin{pmatrix} 0 & 3 & 4 & 7 \\ 3 & 0 & 3 & 7 \\ 4 & 3 & 0 & 7 \\ 7 & 7 & 7 & 0 \end{pmatrix}$	20, [0, 1, 2, 3, 0]
$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$	6, [0, 1, 2, 3, 4, 5, 0]
$\begin{pmatrix} 0 & 1 & INF & 4 \\ 1 & 0 & 2 & INF \\ INF & 2 & 0 & 3 \\ 4 & INF & 3 & 0 \end{pmatrix}$	10, [0, 1, 2, 3, 0]

Вывод

Был производен выбор средств реализации, приведены требования к ПО, реализованы и протестированы алгоритмы умножения матриц.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10;
- оперативная память: 16 Гб;
- процессор: Intel® Core™ i5-8259U;
- количество ядер: 4;
- количество логических процессоров: 8.

Во время тестирования ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

4.2 Сравнение времени выполнения реализаций алгоритмов

Сравнивалось время работы (обычное, по таймеру) последовательной стандартизации данных и стандартизации с использованием параллельного конвейера. Эти реализации сравнивались по времени обработки заявок на стандартизацию массива вещественных чисел из 10000 элементов в зависимости от количества заявок: 1, 5, 25 и от 50 до 250 с шагом 50.

Так как некоторые задачи выполняются достаточно быстро, а замеры времени имеют некоторую погрешность, они для каждой реализации и каждого количества заявок выполнялись 10 раз, а затем вычислялось среднее время работы.

На рисунке 4.1 приведены результаты сравнения времени выполнения реализаций алгоритмов.

Рисунок 4.1 – Сравнение времени работы реализаций в зависимости от количества заявок

Как и ожидалось, параллельная реализация выполняется за меньшее количество времени в сравнении с линейной за счет того, что в ней одновременно на разных лентах (потоках) обрабатываются несколько заявок. Причем с ростом числа заявок разрыв между реализациями увеличивается.

4.3 Анализ статистики параллельного конвейера

На рисунке 4.2 приведен результат сбора статистики при обработке 1000 заявок на обработку массива из 100000 элементов.

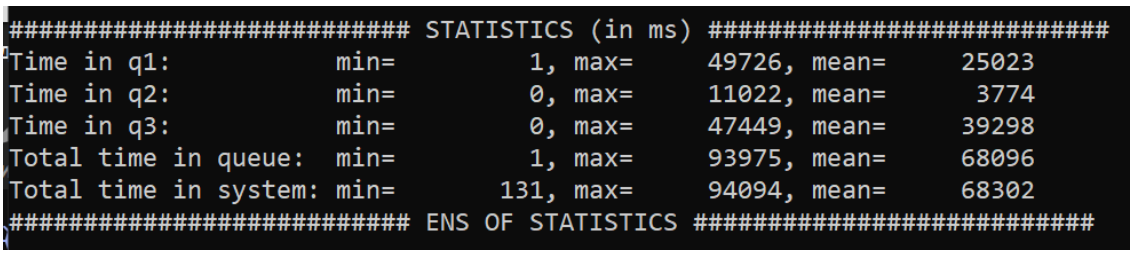

Рисунок 4.2 – Результат сбора статистики

Как видно из рисунка, очереди с наибольшим максимальным временем нахождения в них заявки – первая и третья. Для первой очереди такой результат объясняетсяя тем, что она заполняется генератором заранее, и последняя заявка находится в ней до того момента, пока все предшествующие ей не будут обработаны первой лентой. Это подтверждает и среднее время, проведенное заявкой в первой очереди, которое приблизительно равно половине от максимального.

Для третьей же очереди наибольшее максимальное (и, в частности, среднее) время нахождения в ней заявки связано со сложностью работы соответсвующей ей ленты. Она преобразовывает исходный массив, записывая полученные в результате вычитаний и делений новые значения в результирующий массив, на что тратится большое количество операций и, соовтетсвенно, время.

Минимальное время нахождения заявки в каждой очереди соответсвует отметкам первой задачи: в каждую ленту она попадает сразу же, не ожидая окончания обработки в этом потоке предыдущей задачи.

Вычитая из минимального времени, проведенного заявкой в системе, минимальное суммарное время, проведенное заявкой в очередях, можно вычислить время обработки этой заявки, равное 130 мс.

При этом можно заметить, что минимальное, максмальное и среднее время, проведенное заявкой в системе слабо отличается от тех же замеров для времени, проведенного заявкой в очередях. Это, а также анализ времени, проведенного заявками в третьей очереди, еще раз подтверждает, что при организации параллельного конвейера необходимо разбивать задачу на этапы, схожие по трудоемкости, иначе большую часть времени заявки будут простаивать в очередях.

4.4 Вывод из исследовательской части

Таким образом, параллельная организацияя обработки данных с использованием конвейера работает быстрее, чем линейная обработка. При этом для достижения наилучших показателей необходимо корректно разделять задачу на этапы: так, чтобы время их выполнения было приблизительно равным, иначе большую часть времени заявки будут простаивать в очереди наиболее трудоемкой ленты.

Заключение

В результате выполнения лабораторной работы была достигнута поставленная цель: был реализован муравьиный алгоритм для решения задачи коммивояжера и приобретены навыки параметризации алгоритмов.

В рамках выполнения работы были выаолнены следующие задачи:

1) реализован алгоритм полного перебора для решения задачи коммивояжера;

2) изучен и реализован муравьиный алгоритм для решения задачи коммивояжера;

3) проведена параметризаця муравьиного алгоритма на трех классах данных и подобраны оптимальные параметры;

4) проведен сравнительный анализ трудоемкостей реализаций.

Литература

[1] Ульянов М. В. РЕСУРСНО-ЭФФЕКТИВНЫЕ КОМПЬЮТЕРНЫЕ АЛГОРИТМЫ. РАЗРАБОТКА И АНАЛИЗ // НАУКА ФИЗМАТЛИТ. 2007. С. 201–205.

[2] Задача коммивояжера [Электронный ресурс]. Режим доступа: http://www.math.nsc.ru/LBRT/k5/OR-MMF/TSPr.pdf (дата обращения: 28.10.2021).

[3] Алгоритмы решения задачи коммивояжера [Электронный ресурс]. Режим доступа: https://scienceforum.ru/2021/article/2018025171 (дата обращения: 28.10.2021).

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

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 05.09.2021).

[6] Python и Pycharm [Электронный ресурс]. Режим доступа: https://py-charm.blogspot.com/2017/09/pycharm.html (дата обращения: 05.09.2021).