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 Методы решения задачи коммивояжера

Студент Симонович Р.Д.

Группа ИУ7-55Б

Преподаватель Волкова Л.Л.

Москва – 2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача коммивояжера состоит в поиске кратчайшего замкнутого маршрута, проходящего через все города ровно один раз [1].

Цель лабораторной работы – исследование методов решения задачи коммивояжера: метод полного перебора и метод на основе муравьиного алгоритма.

1) описание алгоритма полного перебора и муравьиного алгоритма;
2) реализация обоих алгоритмов;
3) анализ и сравнение реализованных версий алгоритмов по времени выполнения;
4) описание результатов в отчете.

Вариант: неорграф, без элитных муравьёв, вокруг света за 80 дней (на рёбрах затраты по времени; есть предел – 80 дней в человеческом измерении, не совпадающем с муравьиным), гамильтонов цикл.

1 Аналитическая часть

В данном разделе приведены описания методов решения задачи коммивояжера.

1.1 Метод полного перебора

Метод полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор кратчайшего из них. Преимущество алгоритма – гарантия нахождения кратчайшего пути; недостаток – большая трудоемкость — O(n!) [2].

1.2 Метод муравьиного алгоритма

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

Муравьи действуют, руководствуясь органами чувств. Каждый муравей оставляет на своем пути феромоны, чтобы другие могли ориентироваться. При большом количестве муравьев наибольшее количество феромона остается на наиболее посещаемом пути, посещаемость же может быть связана с длинами ребер.

Муравей имеет следующие характеристики:

1) зрение — способность определить длину ребра;
2) память — способность запомнить пройденный маршрут;
3) обоняние — способность чуять феромон.

Целевая функцию (1.1), характеризующая 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.

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.

Формула обновления феромона ночью ((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

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

где L_k – длина пути k-ого муравья, Q – константа порядка длины путей, N — количество муравьев.

Необходимо, чтобы вероятность (1.2) перехода в заданную точку не была равна нулю. Следовательно, необходимо ввести минимальное возможное количество феромона (τ_{min}) и в случае, если $\tau_{ij}(t+1)$ принимает значение, меньшее, то увеличить значение феромона 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).

2 Конструкторская часть

В данном разделе приведены схемы алгоритмов решения задачи коммивояжера: полным перебором и на основе муравьиного алгоритма. Также приведена оценка трудоёмкости рассмотренных алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора. На рисунке 2.2 представлена схема муравьиного алгоритма.

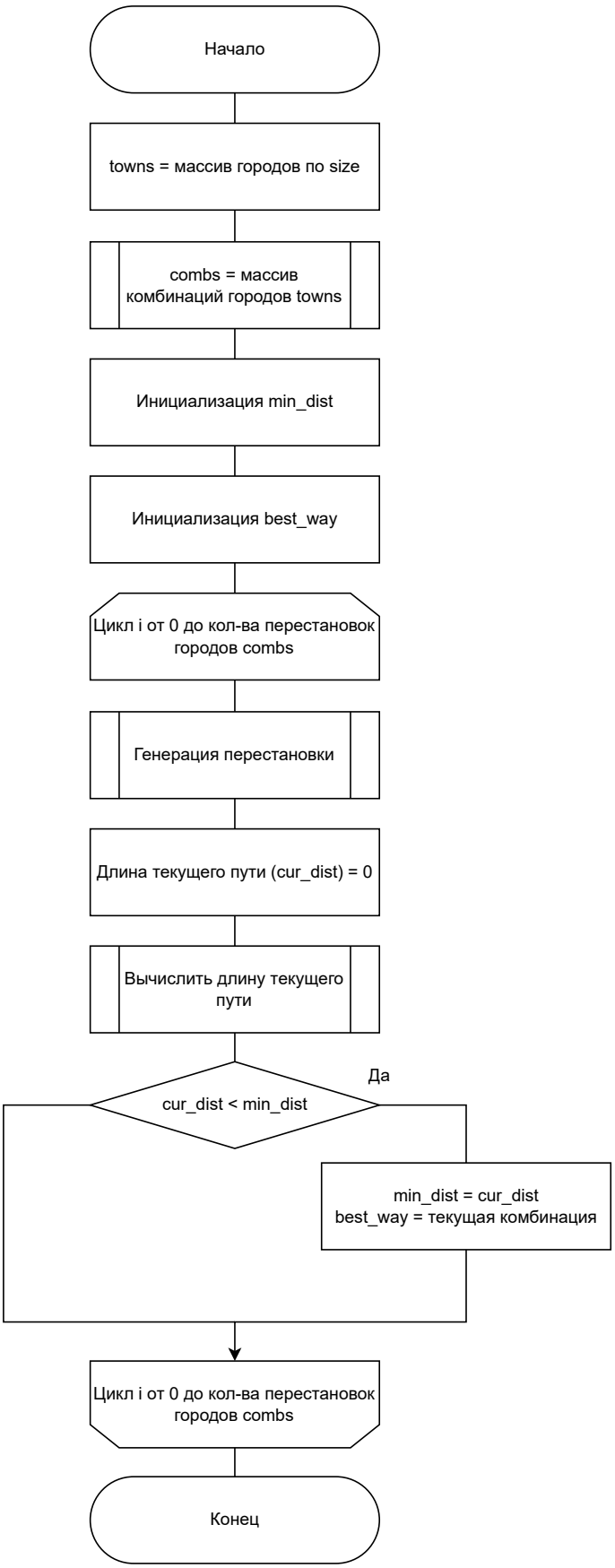

Рисунок 2.1 — Схема алгоритма полного перебора

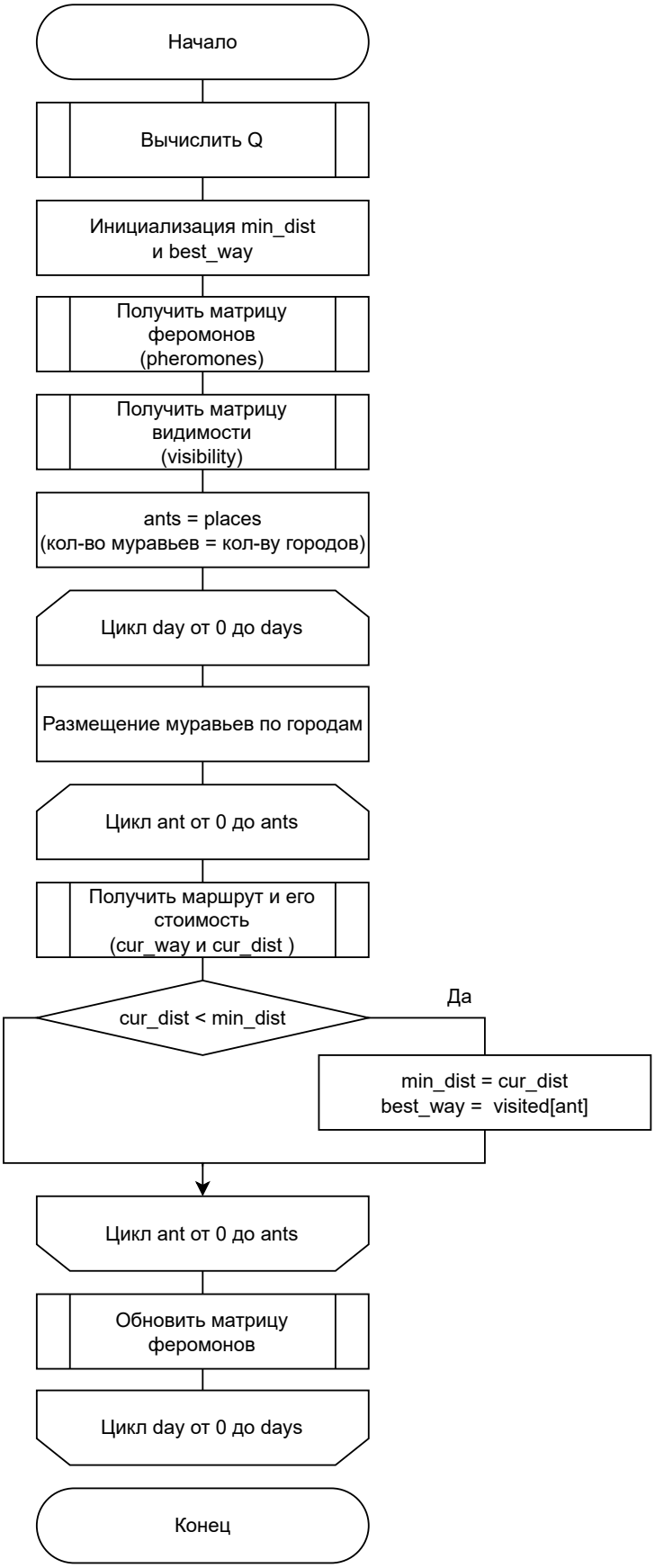

Рисунок 2.2 — Схема алгоритма полного перебора

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению (ПО), а также средства реализации и сами реализации алгоритмов умножения матриц.

3.1 Требования к программному обеспечению

Программа должна предоставлять пользователю возможность ввода произвольной строки любого размера. В качестве результата работы программа должна вывести индексы начала вхождения искомой подстроки в строке или уведомить пользователя о том, что вхождений нет.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования Python. В данной лабораторной работе требуется замерить процессорное время работы выполняемой программы. Инструменты для этого присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time* [**python-lang-time**].

3.3 Сведения о файлах программы

Программа состоит из следующих файлов:

1) main.py – файл, содержащий меню;
2) algorithms.py – файл, содержащий реализации алгоритма;
3) test.py – файл с системой тестирования реализаций алгоритмов.

3.4 Реализация алгоритмов

В листингах **??** и **??** приведены реализации алгоритма полного перебора и муравьиного алгоритма соответственно.

Листинг 3.1 — Алгоритм полного перебора

```
1 def full_comb(matrix, size):
2     places = np.arange(size)
3     placesCombinations = list()
4
5     for combination in it.permutations(places):
6         combArr = list(combination)
7         placesCombinations.append(combArr)
```

```
8
9       minDist = float("inf")
10
11      for i in range(len(placesCombinations)):
12          placesCombinations[i].append(placesCombinations[i][0])
13          curDist = 0
14          for j in range(size):
15              startCity = placesCombinations[i][j]
16              endCity = placesCombinations[i][j + 1]
17              curDist += matrix[startCity][endCity]
18
19          if (curDist < minDist):
20              minDist = curDist
21              bestWay = placesCombinations[i]
22
23      return minDist, bestWay
```

Листинг 3.2 — Муравьиный алгоритм

```
1  def ant_alg(matrix, places, alpha, beta, k_evaporation, days):
2      q = calcQ(matrix, places)
3      bestWay = []
4      minDist = float("inf")
5      pheromones = calcPheromones(places)
6      visibility = calcVisibility(matrix, places)
7      ants = places
8      for day in range(days):
9          route = np.arange(places)
10         visited = calcVisitedPlaces(route, ants)
11         for ant in range(ants):
12             while (len(visited[ant]) != ants):
13                 pk = findWays(pheromones, visibility, visited,
                       places, ant, alpha, beta)
14                 chosenPlace = chooseNextPlaceByPosibility(pk)
15                 visited[ant].append(chosenPlace - 1)
16
17             visited[ant].append(visited[ant][0])
18
19             curLength = calcLength(matrix, visited[ant])
20
21             if (curLength < minDist):
22                 minDist = curLength
```

```
23                 bestWay = visited[ant]
24
25         pheromones = updatePheromones(matrix, places, visited,
               pheromones, q, k_evaporation)
26
27     return minDist, bestWay
```

4 Исследовательская часть

В данном разделе будет приведен анализ реализаций ...

4.1 Технические характеристики

Технические характеристики устройства, на котором осуществлялся анализ реализаций алгоритмов:

— операционная система – Windows 10;

— оперативная память – 16 Гб;

— процессор – AMD Ryzen 7 4700U with Radeon Graphics;

— количество физических ядер – 8;

— количество логических ядер – 8.

Так как анализ проводился на ноутбуке, то для корректного замера времени ноутбук был подключен в сеть электропитания. Во время провидения анализа была обеспечена стабильная загруженность системы.

4.2 Демонстрация работы программы

На рисунке **??** продемонстрирована работа программы при ручном вводе строки.

4.3 Анализ времени выполнения реализаций алгоритмов

Замеры времени работы реализованных алгоритмов для каждого эксперимента проводились 10 раз. В таблице **??** представлен результат зависимости времени работы реализованных алгоритмов от оазмера строки. В таблице **??** представлен результат зависимости времени работы реализованных алгоритмов от количества потоков.

Таблица 4.1 – Результаты замеров времени реализованных алгоритмов в микросекундах

Количество потоков	Последовательный алгоритм
1	4056
2	2557
4	1836

На рисунке **??** представлена зависимость времени работы реализован-

ных алгоритмов от размера строки.

4.4 Вывод

Параллельная реализация алгоритма Бойера – Мура на 8 потоках быстрее последовательной при поиске подстроки в строке размером 500 КБ в 3 раза. При этом, максимальное увеличение скорости параллельной версии алгоритма было достигнута на количестве потоков, равным количеству логических ядер системы, т.е. 8.

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были выполнены следующие задачи:

1) описаны последовательная и параллельная версии алгоритма;
2) реализованы обе версии алгоритма;
3) проведены анализ и сравнение реализованных версий алгоритма по времени выполнения;
4) полученные результаты описаны в отчете.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Штовба С. Д.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003.

2. *Ульянов М. В.* Ресурсно-эффективные компьютерные алгоритмы. — ФИЗМАТЛИТ, 2008. — С. 304.