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»

«Муравьиный алгоритм»

Студент Маслова Марина Дмитриевна

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Лилия Леонидовна

2021 г.

Содержание

Введение

В последние два десятилетия при оптимизации сложных систем исследователи все чаще применяют природные механизмы поиска наилучших решений. Одним из таких механизмов являются муравьиные алгоритмы – новый перспективный метод оптимизации, базирующийся на моделировании поведения колонии муравьев [1]. Первой задачей, к которой был применен муравьиный алгоритм, является проблема коммивояжера (или Travelling Salesman Problep – TSP) [2]. На основе этой задачи будет проводиться исследование муравьиного алгоритма в данной лабораторной работе.

Целью данной работы является проведение сравнительного анализа метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- описать алгоритм полного перебора для решения задачи коммивояжера;
- описать муравьиный алгоритм для решения задачи коммивояжера;
- описать функциональные требования к программе;
- разработать описанные алгоритмы;
- реализовать алгоритм полного перебора и муравьиный алгоритм для решения задачи коммивояжера;
- провести тестирование реализованных алгоритмов;
- провести сравнительный анализ алгоритмов по времени работы реализаций;
- провести параметризацию муравьиног алгоритма на двух классах данных;
- сделать выводы по полученным результатам.

1 Аналитическая часть

В данном разделе представлено теоретическое описание задачи коммивояжера, а алгоритм полного перебора и муравьиный алгоритм для её решения.

1.1 Задача коммивояжера

Задача коммивояжера (или *задача о путешествующем торговце*) состоит в поиске кратчайшего замкнутого маршрута, проходящего через все города ровно 1 раз [1]. Если говорить формально, то необходимо найти гамильтонов цикл минимального веса во взвешенном полном графе.

1.2 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Данный алгоритм имеет высокую сложность $O(n!)$, что большие затраты по времени даже при небольших значениях числа вершин в графе.

1.3 Муравьиный алгоритм

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев [3]. Колония представляет собой систему с очень простыми правилами автономного поведения особей.

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает, во время прохождения, каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. При этом феромон испаряется для исключения случая, когда все муравьи движутся одним и тем же субоптимальным маршрутом. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Для каждого муравья переход из города i в город j зависит от трех составляющих:

– Память муравья (tabu list) — это список посещенных муравьем городов, заходить в которые еще раз нельзя. Используя этот список, муравей гарантированно не попадет в один и тот же город дважды.

– Видимость — величина, обратная расстоянию: $\eta_{ij} = 1/D_{ij}$, где D_{ij} — расстояние между городами i и j. Эта величина выражает эвристическое желание муравья поседить город i из города j, чем ближе город, тем больше желание его посетить.

– Виртуальный след феромона τ_{ij} на ребре (i, j) представляет подтвержденное муравьиным опытом желание посетить город j из города i.

Вероятность перехода k-ого муравья из города i в город j на t-й итерации определяется формулой 1.1:

$$\begin{cases} P_{ij,k}(t) = \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum\limits_{l \in J_{i,k}} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} & \text{если } j \in J_{i,k}, \\ P_{ij,k}(t) = 0 & \text{если } j \notin J_{i,k} \end{cases}, \tag{1.1}$$

где α, β – настраиваемые параметры, $J_{i,k}$ - список городов, которые надо посетить k-ому муравью, находящемуся в i-ом городе, τ - концентрация феромона, а при $\alpha = 0$ алгоритм вырождается в жадный.

После завершения движения всеми муравьями происходит обновление феромона. Если $p \in [0, 1]$ – коэффициент испарения феромона, то новое значения феромона на ребре (i, j) рассчитывается по формуле 1.2:

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

где

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

L_k — длина пути k-ого муравья, Q — некоторая константа порядка длины путей, N — количество муравьев [1].

1.4 Вывод

В данном разделе была описана задача коммивояжера и алгоритмы её решения: полный перебор и муравьиный алгортим. Из представленных описаний можно предъявить ряд требований к разрабатываемому программному обеспечению:

- на вход должна подаваться матрица смежности графа, представляющего города и стоимость переезда между ними;
- для муравьиного алгоритма также должны подаваться на вход коэффициенты α, p и количество итераций;
- на выходе должны выдаваться искомый путь и его длина;
- при неверных входных данных должна выдаваться ошибка.

2 Конструкторская часть

В данном разделе разрабатываются алгоритмы решения задачи коммивояжера: полный перебор и муравьиный алгоритм, также описывается структура программы и способы её тестироваия.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора для решения задачи коммивояжера.

На рисунке 2.2 представлена схема муравьиного алгоритма для решения задачи коммивояжера.

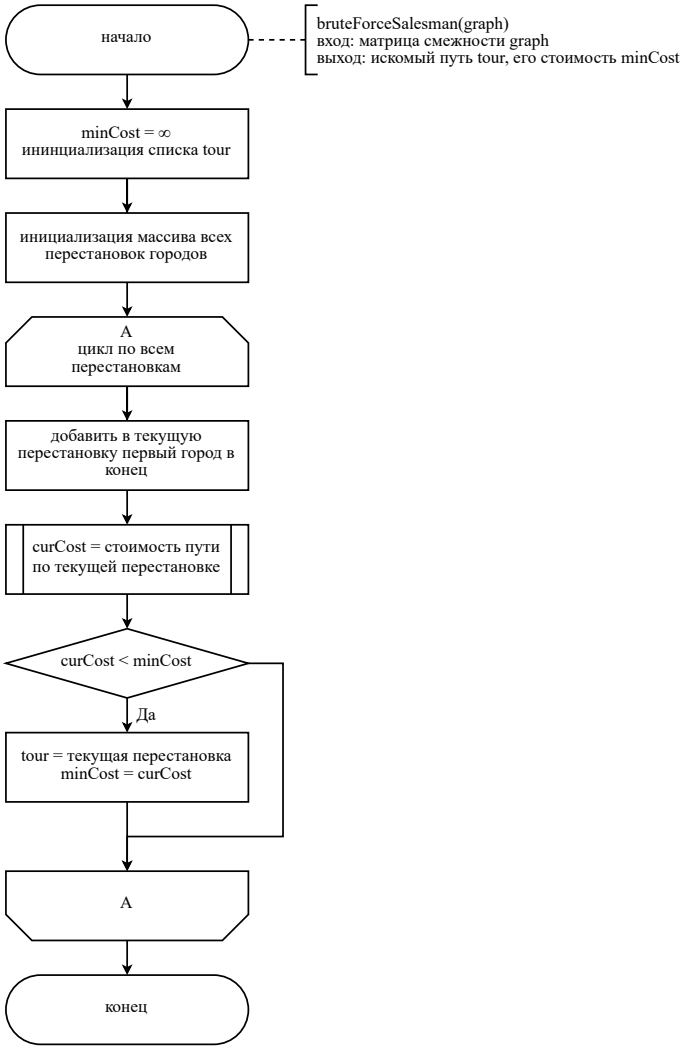

Рисунок 2.1 – Схема алгоритма полного перебора

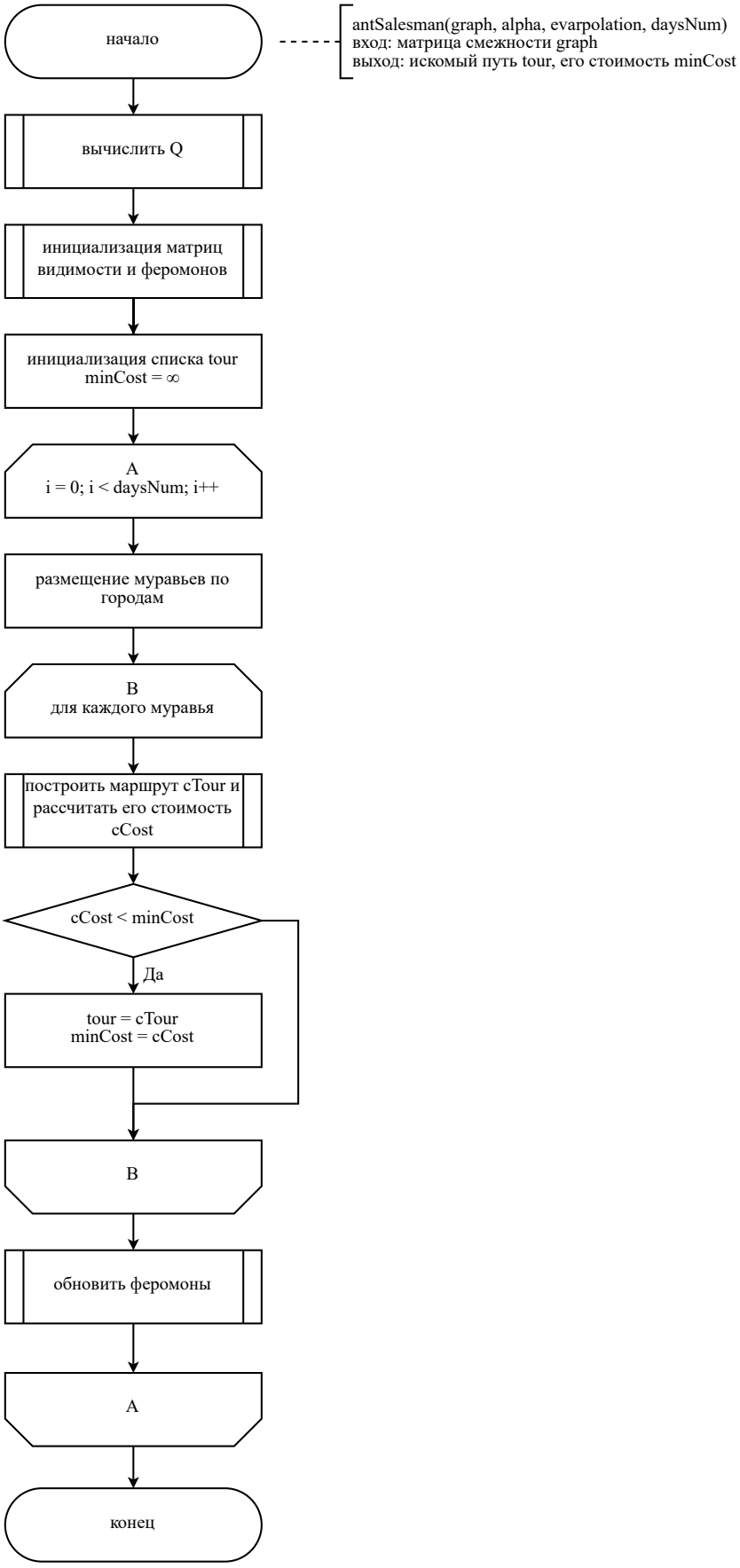

Рисунок 2.2 – Схема муравьиного алгоритма

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.3 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- неверно выбран файл с матрицей смежности;
- неверный ввод коэффициента α;
- неверный ввод коэффициента p;
- неверный ввод количество интераций;
- корректные данные.

2.4 Вывод

В данном разделе были разработаны алгоритмы решения задачи коммивояжера, была описана структура разрабатываемого ПО. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- ввод имени файла с матрицей смежности графа;
- ввод коэффициентов для муравьиного алгоритма;
- вывод искомого пути и минимальной стоимости;
- подсчет времени работы алгоритмов на различных значениях количества вершин;
- параметризация муравьиного алоритма с оценкой точности при заданных параметрах.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран интерпретируемый язык программирования высокого уровня Python[4], так как он позволяет реализовывать сложные задачи за кратчайшие сроки за счет простоты синтаксиса и наличия большого количества подключаемых библиотек.

В качестве среды разработки выбран текстовый редактор Vim[5] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции process_time_ns из библиотеки time[6].

3.3 Листинги кода

В данном подразделе представлены листинги кода алгоритмов:

– реализация алгоритма полного перебора (листинги 3.1-3.2);

– реализация муравьиного алгоритма (листинги 3.3-3.9).

Листинг 3.1 – Релизация алгоритма полного пербора

```
def salesman(graph):
    if len(graph) > 10:
        return ()
    tour = []
    minCost = float("inf")

    for perm in permutations(list(range(len(graph)))):
        curTour = list(perm)
        curTour.append(perm[0])

        curCost = getTourCost(curTour, graph)

        if curCost < minCost:
            tour = curTour
            minCost = curCost

    return tour, minCost
```

Листинг 3.2 – Функция поиска стоимости пути

```
def getTourCost(tour, graph):
    cost = 0

    for i in range(len(tour) - 1):
        cost += graph[tour[i]][tour[i + 1]]

    return cost
```

Листинг 3.3 – Релизация муравьиного алгоритма

```
def salesman(graph, alpha, evarpolation, daysNum):
    num = len(graph)

    if num <= 1:
        return [0] if num else [], 0

    Q = findCostOrder(graph)
    vis, phero = getEdgeCoefs(graph)
    tour = []
```

Листинг 3.3 (продолжение)

```
    minCost = float("inf")

    for i in range(daysNum):
        ants = [{'tabu' : [j], 'cost' : 0} for j in range(num)]

        for j, ant in enumerate(ants):
            ants[j] = getAntTour(ant, graph, vis, phero, alpha)

            if ants[j]['cost'] < minCost:
                minCost = ants[j]['cost']
                tour = ants[j]['tabu']

        updatePheromones(phero, ants, Q, evarpolation)

    return tour, minCost
```

Листинг 3.4 – Функция поиска параметра Q

```
def findCostOrder(graph):
    size = len(graph)
    return sum(graph[i][j] if i != j else 0
               for j in range(size)
               for i in range(size)) / size
```

Листинг 3.5 – Функция инициализации видимости и феромонов

```
def getEdgeCoefs(graph):
    size = len(graph)

    eta = [[0 for j in range(size)] for i in range(size)]
    for i in range(size):
        for j in range(size):
            if i != j and graph[i][j]:
                eta[i][j] = 1 / graph[i][j]

    tau = [[START_PHEROMONE for j in range(size)] for i in range(size)]

    return eta, tau
```

Листинг 3.6 – Функция получения тура и его стоимости для одного муравья

```
def getAntTour(ant, graph, vis, phero, alpha):
    size = len(graph)

    while len(ant['tabu']) != size:
        nextCity = chooseCity(ant['tabu'], size, vis, phero, alpha)
```

Листинг 3.6 (продолжение)

```
6
7           ant['cost'] += graph[ant['tabu'][-1]][nextCity]
8           ant['tabu'].append(nextCity)
9
10      ant['cost'] += graph[ant['tabu'][-1]][ant['tabu'][0]]
11      ant['tabu'].append(ant['tabu'][0])
12
13      return ant
```

Листинг 3.7 – Функция выбора муравьем следующего города

```
1  def chooseCity(tabuList, size, vis, phero, alpha):
2      probabilities = getProbabilities(size, tabuList, vis, phero, alpha)
3
4      choise = random()
5      probSum = probabilities[0]
6      nextCity = 0
7
8      while choise > probSum and nextCity < size - 1:
9          nextCity += 1
10         probSum += probabilities[nextCity]
11
12     return nextCity
```

Листинг 3.8 – Получение вероятностей перехода муравья в каждый город

```
1  def getProbNumerator(cFrom, cTo, vis, phero, alpha):
2      return pow(phero[cFrom][cTo], alpha) * pow(vis[cFrom][cTo], 1 - alpha)
3
4
5  def getProbabilities(citiesNum, tabu, vis, phero, alpha):
6      probabilities = [0] * citiesNum
7
8      for city in range(citiesNum):
9          if city not in tabu:
10             curCity = tabu[-1]
11
12             probabilities[city] = getProbNumerator(curCity, city, vis, phero,
13                                                    alpha)
14
15     denominator = sum(probabilities)
16
17     for i in range(citiesNum):
18         probabilities[i] /= denominator
19
20     return probabilities
```

Листинг 3.9 – Функция обновления феромона

```
def updatePheromones(phero, ants, Q, evarpolation):
    size = len(phero)

    addPhero = [[0 for i in range(size)] for j in range(size)]

    for ant in ants:
        delta = Q / ant['cost']
        for i in range(size - 1):
            addPhero[ant['tabu'][i]][ant['tabu'][i + 1]] += delta

    for i in range(size):
        for j in range(size):
            phero[i][j] *= (1 - evarpolation)
            phero[i][j] += addPhero[i][j]
            phero[i][j] = 0.1 if phero[i][j] < MIN_PHEROMONE else phero[i][j]
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты программы.

Таблица 3.1 – Функциональные тесты

Количество заявок	**Ожидаемый результат**
пустая строка	Сообщение об ошибке
./data/g5.txt j	Сообщение об ошибке
./data/g5.txt 0.5 j	Сообщение об ошибке
./data/g5.txt 0.5 0.5 j	Сообщение об ошибке
./data/g5.txt 0.5 0.5 100	0 -> 2 -> 1 -> 4 -> 3 -> 0 19.0

3.5 Вывод

В данном разделе были реализованы алгоритмы решения задачи коммивояжера: полный перебор и муравьиный алгоритм. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– Операционная система: Manjaro [7] Linux x86_64.

– Память: 8 GiB.

– Процессор: Intel® Core™ i5-8265U, 4 физических ядра, 8 логических ядра[8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

На рисунке **??** представлены результаты работы последовательной обработки, на рисунке **??** – конвейерной.

4.3 Результаты тестирования

Программа была протестирована на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки времени работы последовательной и конвейерной реализации алгоритмов шифрования был проведен эксперимент, в котором определялось влияние количества заявок и количества слов в сообщении на время работы каждого из алгоритмов. Тестирование проводилось на количестве заявок от 5 до 10 с шагом 5, от 25 до 100 с шагом 25 и от 100 до 1000 с

шагом 250, а количество слов принимало значения $5, 10, 25, 50, 75, 100$. Время работы на каждом из значений было получено с помощью *бенчмарков*[**?**], являющимися встроенными средствами языка Go. В них количество повторов измерений времени изменяется динамически до тех пор, пока не будет получен стабильный результат.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

В таблице **??** представлены результаты измерения времени работы последовательной и конвейерной реализаций в зависимости от числа заявок. На рисунке **??** представлен соответствующий график.

В таблице **??** представлены результаты измерения времени работы последовательной и конвейерной реализаций в зависимости от числа слов в сообщениях при фиксированном числе заявок, равном 50. На рисунке **??** представлен соответствующий график.

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- при количестве заявок до 10 последовательная и конвейерная реализация генерации зашифрованных сообщений отрабатывают за одинаковое время, а при количестве заявок до 5 последовательная реализация работает 1.3 раза быстрее, что объясняется затратами на передачу данных между лентами с помощью очередей/каналов в конвейерной реализации;
- при большем количестве заявок от 25 конвейерная реализация работает в 1.7 раза быстрее последовательной;
- при фиксированном количестве заявок при различных количествах слов в сообщениях конвейерная реализация работает в 1.7 раза быстрее последовательной.

Таким образом, для обработки количества заявок большего 10 для достижения оптимальной скорости вычислений необходимо использовать конвейерную реализацию. Если работа просходит с количеством заявок до 10 достаточно последовательной реализации, то есть нет необходимости реализовывать более сложный конвейерный алгоритм.

Заключение

В ходе исследования был проведен сравнительный аналализ последовательной и конвейерной реализации алгорима поэтапного шифрования сообщений. В результате исследования было выяснено, что при количестве заявок более 10 конвейерная реализация дает выигрыш в 1.7 раза по сравнению с последовательной, что говорит о преимуществе параллельной реализации этапов конвейера при решении поставленной задачи.

В ходе выполения лабораторной работы:

- были описаны и разработаны алгоритмы этапов конвейерной обработки данных;
- были описаны и разработаны линейная и конвейерная обрабоки данных;
- был реализован каждый из описанных алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов;
- были получены зависимости времени работы линейной и конвейерной реализаций от числа и размера заявок.

Таким образом, все поставленные задачи были выполнены, а цель достигнута.

Список литературы

[1] Штовба С. Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — № 4. — С. 70–75. — URL: https://www.researchgate.net/profile/Serhiy-Shtovba/publication/279535061_Stovba_SD_Muravinye_algoritmy_Exponenta_Pro_Matematika_v_prilozeniah_-_2003_-_No4_-_S_70-75/links/55b7d36808aed621de04d99f/Stovba-SD-Muravinye-algoritmy-Exponenta-Pro-Matematika-v-prilozeniah-2003-No4-S-70-75.pdf (дата обращения: 12.12.2021).

[2] Зайченко Ю., Мурга Н. Исследование муравьиных алгоритмов оптимизации в задачи коммивояжера // International Journal "Information Models and Analyses". — 2013. — Т. 2, № 4. — С. 370–384. — URL: http://www.foibg.com/ijima/vol02/ijima02-04-p08.pdf (дата обращения: 12.12.2021).

[3] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. — ФИЗМАТЛИТ, 2008. — с. 304.

[4] Welcome to Python. — URL: https://www.python.org (дата обращения: 12.10.2021).

[5] welcome home : vim online. — URL: https://www.vim.org/ (дата обращения: 12.10.2021).

[6] time — Time access and conversions. — URL: https://docs.python.org/3/library/time.html#time.process_time_ns (дата обращения: 04.10.2021).

[7] Manjaro — enjoy the simplicity. — URL: https://manjaro.org/ (дата обращения: 17.10.2021).

[8] Процессор Intel® Core™ i5-8265U. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html (дата обращения: 17.10.2021).