|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 6 |

**Название:**

Муравьиный алгоритм и метод полного перебора для решения задачи коммивояжёра

**Дисциплина:** Анализ Алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Л.Л. Волкова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

Оглавление

[Введение 3](#_Toc58411992)

[1. Аналитическая часть 4](#_Toc58411993)

[1.1. Поиск полным перебором 4](#_Toc58411994)

[1.2. Двоичный поиск в упорядоченном словаре 4](#_Toc58411995)

[1.3. Сегментный поиск с частным анализом 4](#_Toc58411996)

[2. Конструкторская часть 5](#_Toc58411997)

[2.1. Схемы алгоритмов 5](#_Toc58411998)

[2.2. Частный анализ данных 7](#_Toc58411999)

[3. Технологическая часть 8](#_Toc58412000)

[3.1. Реализация алгоритмов 8](#_Toc58412001)

[3.2. Тестирование алгоритмов 9](#_Toc58412002)

[4. Экспериментальная часть 10](#_Toc58412003)

[4.1. Сравнение алгоритмов по времени работы 10](#_Toc58412004)

[Заключение 11](#_Toc58412005)

[Список литературы 12](#_Toc58412006)

# Введение

**Цель** лабораторной работы: изучить и применить на практике алгоритмы для решения задачи коммивояжера. В данной лабораторной работе рассматривается муравьиный алгоритм, алгоритм полного перебора.

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

В ходе выполнения лабораторной работы требуется решить следующие **задачи**.

1. Реализовать муравьиный алгоритм.
2. Реализовать алгоритм полного перебора.
3. Сравнить алгоритмы по затраченным ресурсам.
4. Сравнить результаты муравьиного алгоритма от параметров.

# Аналитическая часть

Рассмотрим понятия, с которыми мы столкнемся при выполнении лабораторной работы.

# Алгоритм полного перебором

Полный перебор (или метод «грубой силы», англ. brute force) — метод решения математических задач. Относится к классу методов поиска решения исчерпыванием всевозможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

# Муравьиный алгоритм

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений. Они могут использоваться как для статических, так и для динамических комбинаторных оптимизационных задач. Сходимость гарантирована, то есть в любом случае мы получим оптимальное решение, однако скорость сходимости неизвестна.

# Конструкторская часть

На вход алгоритмы получают матрицы расстояний. Муравьиный алгоритм имеет несколько настроек такие, как количество муравьев, элитный муравьев, итераций, испарение феромонов, альфа и бета.

# Схемы алгоритмов

На рис. 1-2 приведена схема муравьиного алгоритма.

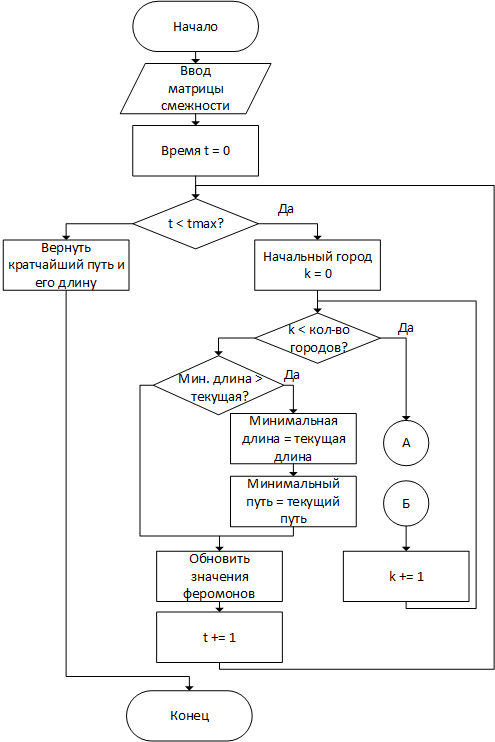


Рисунок 1. Муравьиный алгоритм

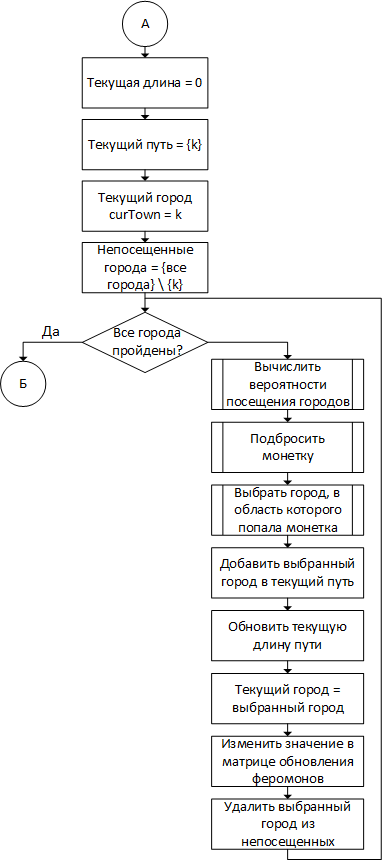


Рисунок 2. Муравьиный алгоритм

На рис. 3–4 приведена схема полного перебора

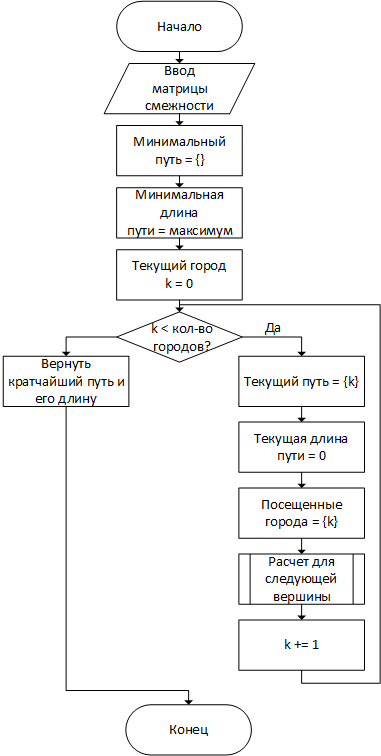


Рисунок 3. *Алгоритм полного перебора*

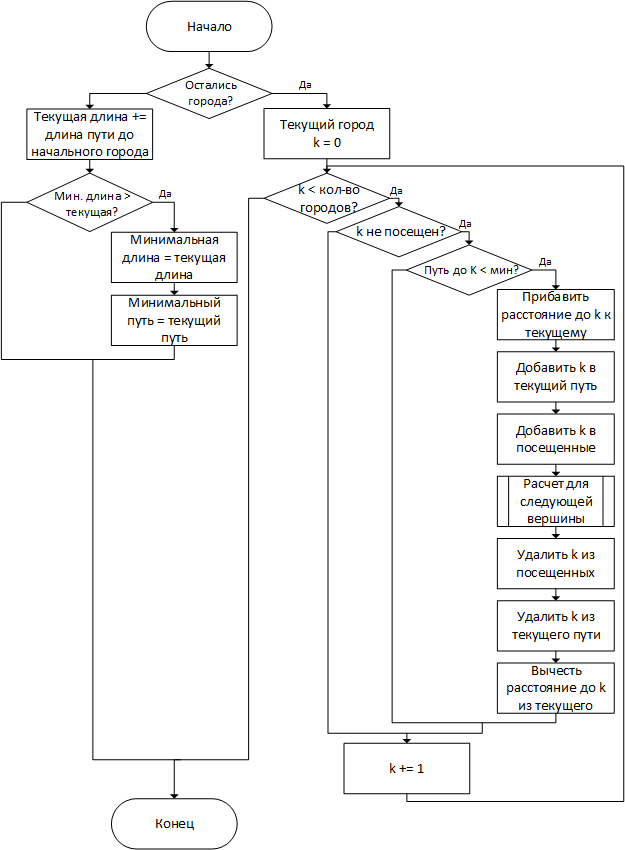


Рисунок 4. *Алгоритм полного перебора*

# Технологическая часть

В качестве языка программирования был выбран python, т.к. данный язык программирования имеет большое количество полезных библиотек для различных необходимостей, а также язык предоставляет средства для быстрого прототипирования и динамической семантики. Для замера процессорного времени была использована функция process\_time(), стандартной библиотеки python – time [2].

# Реализация алгоритмов

В листингах 1-3 представлена реализация алгоритмов сортировки массивов. В листинге 4 представлена функция для замера времени выполнения заданной функции на заданном количестве итераций на матрицах указанного размера.

Листинг 1. Алгоритм полного перебора

**def** brute(dist):  
 n = len(dist)  
  
 **for** i **in** range(n):  
 **for** j **in** range(len(dist[i])):  
 **if** dist[i][j] == 0:  
 dist[i][j] = np.inf  
  
 iteration = math.factorial(n)  
 res = []  
  
 **for** i **in** range(iteration):  
 tmp = []  
 elem = [i **for** i **in** range(n)]  
  
 **for** j **in** range(n):  
 a = math.factorial(n - j - 1)  
 index = ((i // a) % (n - j))  
 tmp.append(elem[index])  
 elem.pop(index)  
 res.append(tmp)  
  
 shortcut = (**"placeholder"**, np.inf)  
 **for** path **in** res:  
 tmp = dist[path[0]][path[-1]]  
 **for** i **in** range(1, len(path)):  
 tmp += dist[path[i - 1]][path[i]]  
 **if** tmp < shortcut[1]:  
 shortcut = (path, tmp)  
 **return** shortcut

Листинг 2. Муравьиный алгоритм

**class** Ant:  
 **def** \_\_init\_\_(self, dist, ant, elite, iteration, decay, alpha=1., beta=1.):  
 self.dist = dist  
 self.ant = ant  
 self.elite = elite  
 self.decay = decay  
 self.alpha = alpha  
 self.beta = beta  
 self.iteration = iteration  
  
 self.pheromone = np.ones(dist.shape) / len(dist)  
 self.cities = range(len(dist))  
  
 **def** run(self):  
 shortcut = (**"placeholder"**, np.inf)  
 **for** i **in** range(self.iteration):  
 all\_paths = self.gen\_paths()  
 self.update\_pheromone(all\_paths, self.elite)  
 tmp = min(all\_paths, key=**lambda** x: x[1])  
 **if** tmp[1] < shortcut[1]:  
 shortcut = tmp  
 self.pheromone = self.pheromone \* self.decay  
 **return** shortcut  
  
 **def** update\_pheromone(self, all\_paths, elite):  
 sorted\_paths = sorted(all\_paths, key=**lambda** x: x[1])  
 **for** path, len **in** sorted\_paths[:elite]:  
 **for** move **in** path:  
 self.pheromone[move] += 1.0 / self.dist[move]  
  
 **def** probe\_move(self, pheromone, dist, visited):  
 tmp = np.copy(pheromone)  
 **for** i **in** range(len(tmp)):  
 **if** i **in** visited:  
 tmp[i] = 0  
 chance = tmp \*\* self.alpha \* ((1.0 / dist) \*\* self.beta)  
 chance = chance / chance.sum()  
 move = np.random.choice(self.cities, 1, p=chance)[0]  
 **return** move  
  
 **def** gen\_path(self, start):  
 path = []  
 visited = set()  
 visited.add(start)  
 prev = start  
 **for** i **in** range(len(self.dist) - 1):  
 move = self.probe\_move(self.pheromone[prev], self.dist[prev], visited)  
 path.append((prev, move))  
 prev = move  
 visited.add(move)  
 path.append((prev, start))  
 **return** path  
  
 **def** path\_len(self, path):  
 len = 0  
 **for** city **in** path:  
 len += self.dist[city]  
 **return** len  
  
 **def** gen\_paths(self):  
 all\_paths = []  
 *# Размещаем муравьев в города* **for** i **in** range(self.ant):  
 path = self.gen\_path(0)  
 all\_paths.append((path, self.path\_len(path)))  
 **return** all\_paths

# Экспериментальная часть

В данном разделе приведены примеры работы программы, проведена параметризация метода решения задачи коммивояжера на основе муравьиного алгоритма, а также проведен сравнительный анализ двух алгоритмов.

# Примеры работ

На рисунках 5-6 приведены примеры работы программы.

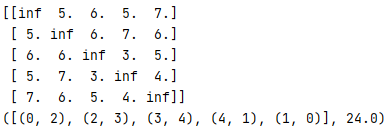


Рисунок 5. *Муравьиный алгоритм*

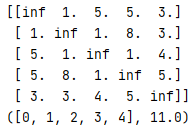
**

Рисунок 6. *Алгоритм полного перебора*

# Параметризация метода

Для проведения экспериментов была использована матрица смежности 10x10. В каждом эксперименте фиксировались значения 𝛼, *decay* и iteration. В течение экспериментов значения 𝛼, *decay* менялись от 0 до 1 с шагом 0.25, *iteration* от 10 до 300 с шагом 10. Количество повторов каждого эксперимента равнялось 100, результатом проведения эксперимента считалась усредненная разница между длинной маршрута, рассчитанного алгоритмом полного перебора и муравьиным алгоритмом с текущими параметрами.

В таблице 1 представлены 10 лучших результатов по наименьшему отклонению от минимального расстояния.

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр *decay* | Параметр  𝛼 | Параметр  *iteration* | Отклонение от минимального пути |
| 0.25 | 0.75 | 290 | 0.05 |
| 0.25 | 0.75 | 250 | 0.09 |
| 0.25 | 0.75 | 280 | 0.09 |
| 0.25 | 0.75 | 300 | 0.1 |
| 0.5 | 0.75 | 270 | 0.1 |
| 0.5 | 0.75 | 300 | 0.1 |
| 0.25 | 0.75 | 240 | 0.11 |
| 0.25 | 0.75 | 270 | 0.11 |
| 0.5 | 0.75 | 290 | 0.11 |
| 0.25 | 0.75 | 220 | 0.12 |

Как видно из представленной таблицы, наилучшим значением настроечного параметра 𝛼 является значение, равное 0.75, а коэффициента испарения *decay* - 0.25. При *iteration* = 290 достигается наименьшее различие с минимальным путем.

# Сравнение алгоритмов

Для сравнения муравьиного алгоритма с алгоритмом полного перебора используются оптимальные параметры муравьиного алгоритма: 𝛼 = 0.75, *decay* = 0.25, *iteration* = 290. Количество городов меняется от 3 до 15. Для произведения замеров времени выполнения реализации алгоритмов будет использована формула: 𝑡 = 𝑇/𝑁, где t — среднее время выполнения алгоритма, N — количество замеров, T — время выполнения N замеров. Неоднократное измерение времени необходимо для получения более точного результата. Количество замеров взято равным 100. На рисунке 7 представлен график зависимости времени работы каждого из реализованных алгоритмов от количества городов.



Рисунок 7. *График зависимости времени работы алгоритмов от количества городов.*

**Вывод:** в результате проведенных экспериментов были выявлены оптимальные параметры для муравьиного алгоритма: 𝛼 = 0.75, *decay* = 0.25, *iteration* = 290. Однако стоит учитывать, что чем больше значение *iteration*, тем больше вероятность того, что будет найден идеальный маршрут, но при этом будет возрастать время выполнения программы. Также был проведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора. Алгоритм полного перебора рационально использовать для матриц небольшого размера и в случае, если необходимо получить наименьшее расстояние. В остальных случаях муравьиный алгоритм является более эффективным по времени.

# Заключение

Цель работы достигнута. Были изучены и реализованы основные алгоритмы для решения задачи коммивояжера: муравьиный и полного перебора. Был проведен их сравнительный анализ, в ходе которого были получены зависимости времени выполнения алгоритмов от размеров входной матрицы расстояний, кроме того, была проведена параметризация муравьиного метода на фиксированном наборе данных.

В результате экспериментов было получено, что муравьиный алгоритм работает быстрее алгоритма полного перебора и линейно зависит от размеров входной матрицы, также были выявлены оптимальные параметры для муравьиного алгоритма, обеспечивающие минимальное отклонение длины вычисленного маршрута от длины минимального.

# Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 c.

2. Официальный сайт Python, документация [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html>, свободный (дата обращения: 16.09.20).