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**

О Т Ч Е Т

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

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.

1. Аналитическая часть

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

1.1. 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.

1.2. 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.

2. 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.

2.1. Схемы алгоритмов

На рис. 1-2 приведена схема муравьиного алгоритма.

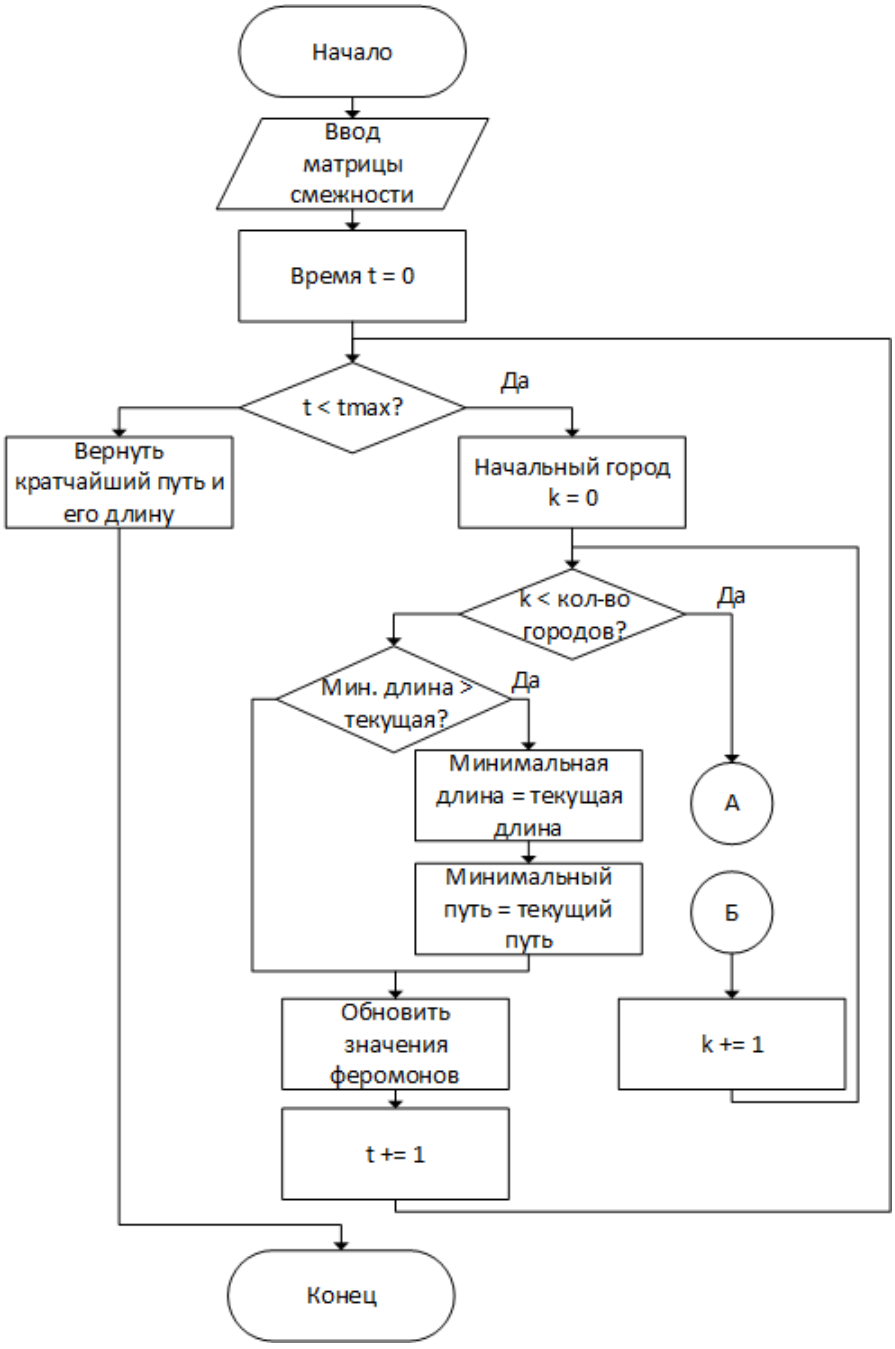

Рисунок 1. *Муравьиный алгоритм*

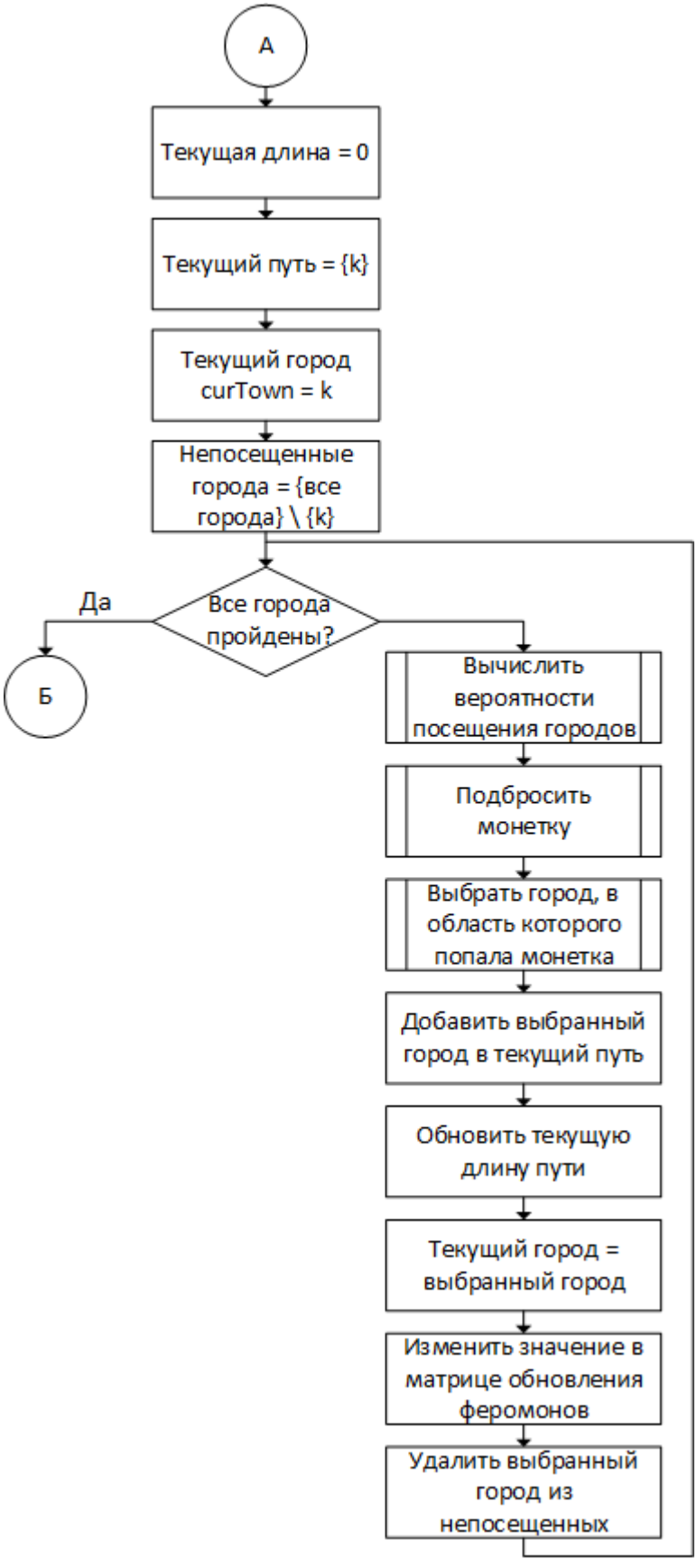

Рисунок 2. *Муравьиный алгоритм*

На рис. 3–4 приведена схема полного перебора

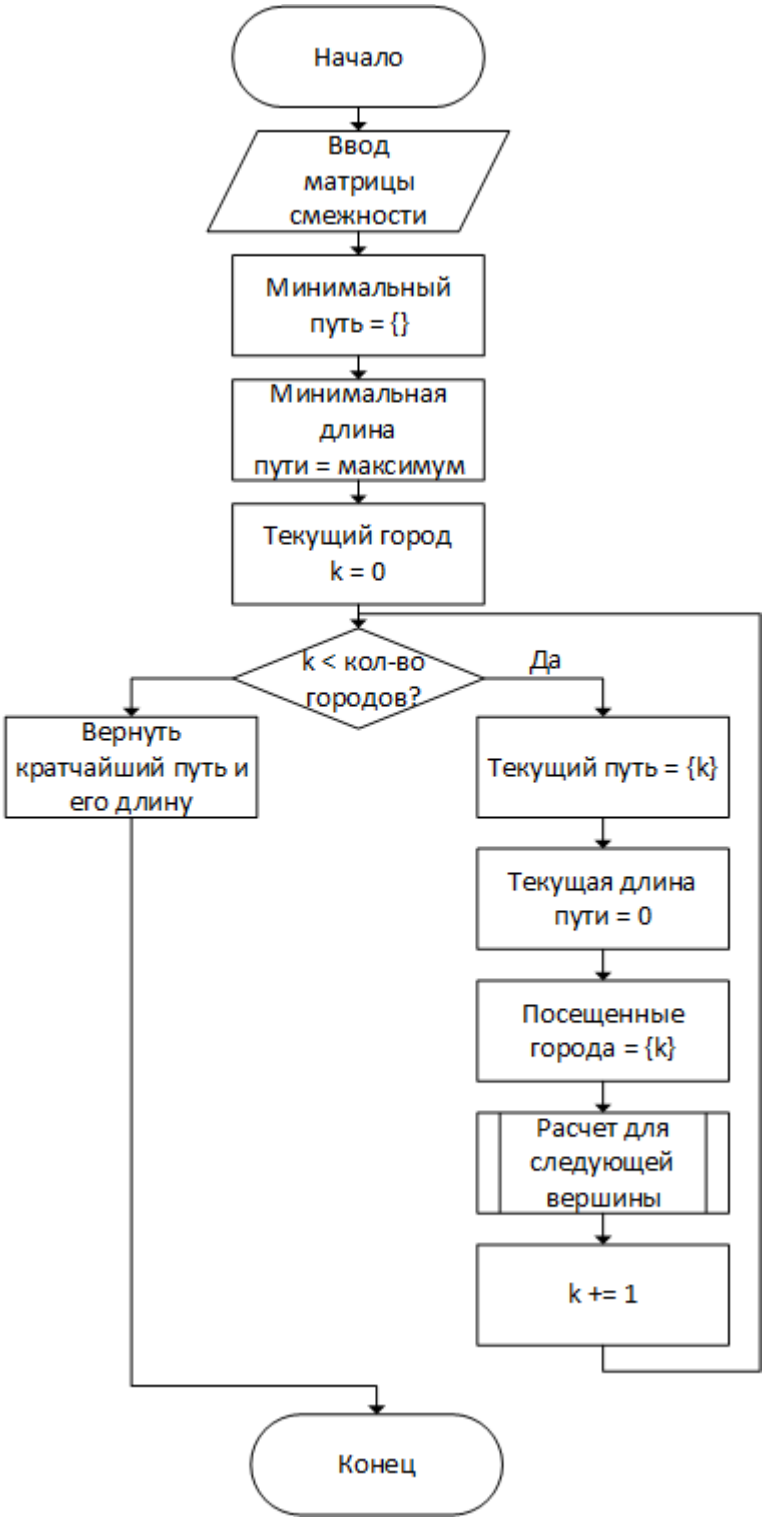

Рисунок 3. *Алгоритм полного перебора*

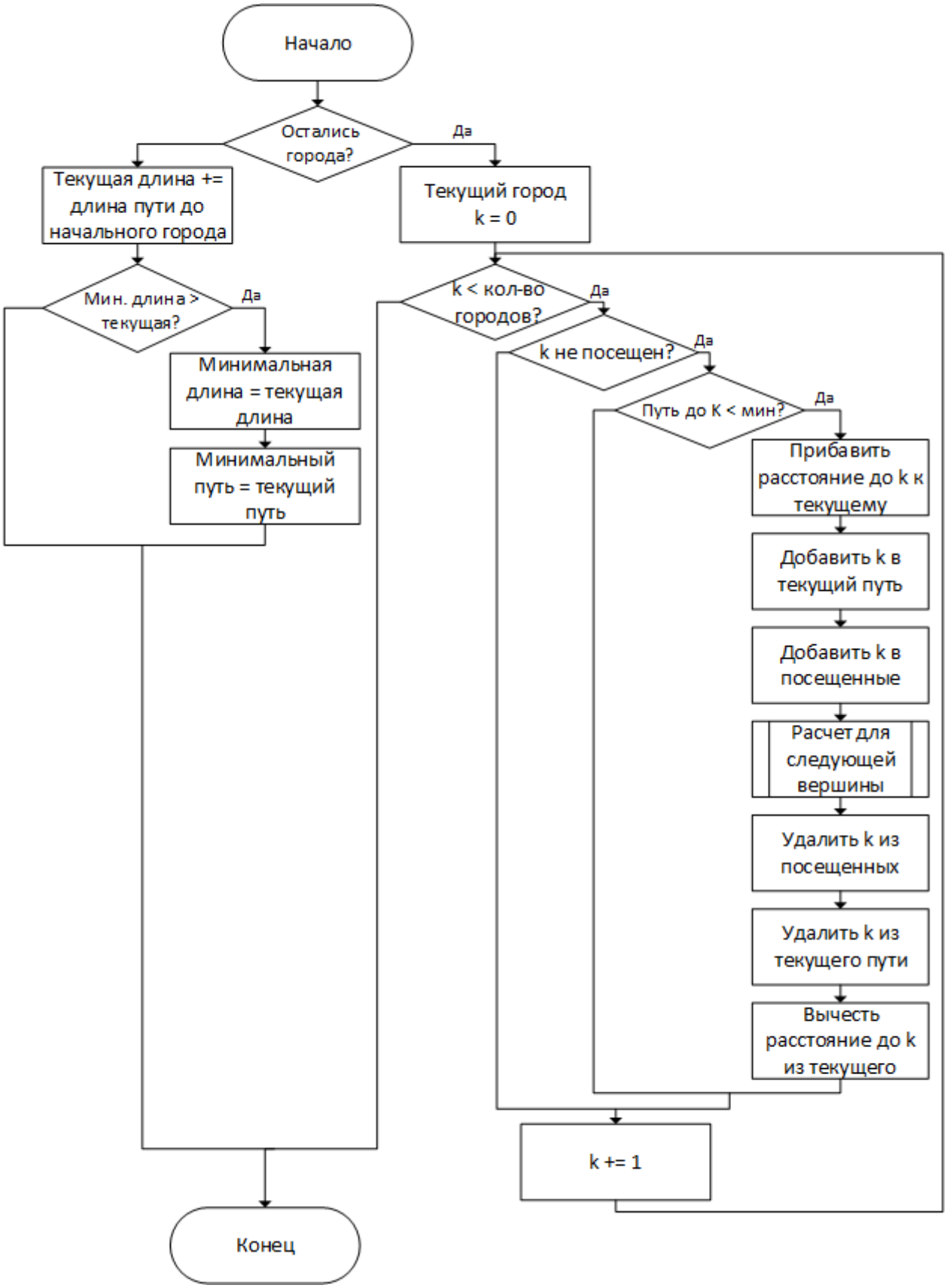

Рисунок 4. *Алгоритм полного перебора*

3. 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].

3.1. 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
```

4. 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.

4.1. Примеры работ

На рисунках 5-6 приведены примеры работы программы.

```
[[inf  5.  6.  5.  7.]
 [ 5. inf  6.  7.  6.]
 [ 6.  6. inf  3.  5.]
 [ 5.  7.  3. inf  4.]
 [ 7.  6.  5.  4. inf]]
([(0, 2), (2, 3), (3, 4), (4, 1), (1, 0)], 24.0)
```

Рисунок 5. *Муравьиный алгоритм*

```
[[inf  1.  5.  5.  3.]
 [ 1. inf  1.  8.  3.]
 [ 5.  1. inf  1.  4.]
 [ 5.  8.  1. inf  5.]
 [ 3.  3.  4.  5. inf]]
([0, 1, 2, 3, 4], 11.0)
```

Рисунок 6. *Алгоритм полного перебора*

4.2. 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.

4.3. 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 с.

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