МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Жадный алгоритм и A*

Студент гр. 9303 _____ Павлов Д.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритмы построения пути в ориентированном графе. Реализовать жадный алгоритм и алгоритм A*.

Задание.

Вариант 3. Написать функцию, проверяющую эвристику на допустимость и монотонность.

Разработайте программу, которая решает задачу построения пути в ориентированном графе при помощи жадного алгоритма. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом A*. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII. В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Выполнение работы.

Описание классов и функций:

1. Для жадного алгоритма

   `sorting(dictator)` – функция для сортировки словаря соединений узла по длине ребра.

   `Graph` – класс графа

Описание методов класса `Graph`:

`__init__(self, st, en)` – конструктор класса

`create_graph(self)` – строит граф, сохраняя все точки в словарь

`greedy(self, key)` – находит путь от начального узла графа до конечного, используя жадный алгоритм

2. Для A*:

`heuristic(a, b)` – эвристическая функция

`PriorityQueue` – класс очереди

`Graph` – класс графа

Описание методов класса `Graph`:

`__init__(self, st, en)` – конструктор класса.

`get_weight(self, v1, v2)` – находит вес ребра между узлами

`get_connections(self, node)` – находит все связи узла

`a_start_search(self, start, end)` – находит путь от одного узла графа до другого

`show_res(self)` – выводит ответ

`check_monotone(self, res)` – проверяет эвристику на монотонность

`check_admissibility(self, res)` – проверяет эвристику на допустимость

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Были изучены и реализованы следующие алгоритмы на графах: жадный алгоритм поиска пути в ориентированном графе и алгоритм A* поиска минимального пути в ориентированном графе.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: greedy.py

```
from sys import stdin


def sorting(dictator):
    keys = []
    list_d = list(dictator.items())
    list_d.sort(key=lambda k: k[1])
    for i in list_d:
        keys.append(i[0])
    return keys


class Graph:
    def __init__(self, st, en):
        self.start = st
        self.end = en
        self.graph = dict()
        self.path = [st]

    def greedy(self, key):
        seen = False
        tmp_keys = sorting(self.graph[key])
        for item in tmp_keys:
            if item in self.path:
                continue
            if item in self.graph.keys():
                self.path.append(item)
                seen = self.greedy(item)
            if seen or item == self.end:
```

```
                seen = True
                if self.end not in self.path:
                    self.path.append(self.end)
                break
        return seen

    def create_graph(self):
        self.graph = dict()
        for line in stdin:
            st, en, leng = line.split(" ")
            leng = leng.strip()
            if st in self.graph.keys():
                self.graph[st][en] = float(leng)
            else:
                self.graph[st] = dict({en: float(leng)})


if __name__ == "__main__":
    start, end = str(input()).split(" ")
    graph = Graph(start, end)
    graph.create_graph()
    graph.greedy(start)
    print(''.join(graph.path))
```

Название файла: a_star.py

```
import heapq
from sys import stdin
# from icecream import ic

ASCII = 97


class PriorityQueue:
    def __init__(self):
        self.elements = []
```

```
    def empty(self):
        return len(self.elements) == 0

    def put(self, item, priority):
        heapq.heappush(self.elements, (priority, item))

    def get(self):
        return heapq.heappop(self.elements)[1]


def heuristic(a, b):
    return abs(b - a)


class Graph:

    def __init__(self, st, en):
        self.start = st
        self.end = en
        self.graph = []

    def get_connections(self, node):
        return self.graph[node]

    def get_weight(self, v1, v2):
        con = self.get_connections(v1)
        # ic(con)
        for w in con:
            if w[0] == v2:
                return w[1]

    def check_monotone(self, res):
        # ic(res)
        evr_list = []
```

```
            for wt in self.graph:
                for j in wt:
                    evr_list.append(j)
            # ic(evr_list)
            tops = []
            for name in list(res):
                name = ord(name) - ASCII
                tops.append(name)
            wt = []
            # ic(self.get_connections(tops[0]))
            for i in range(len(tops)):
                if i != 0:
                    wt.append(self.get_weight(tops[i     -     1],
tops[i]))
                else:
                    wt.append(0)

            heur = []
            for j in range(len(tops)):
                if j != 0:
                    h1 = abs(tops[j] - self.end)
                    h2 = abs(tops[j - 1] - self.end)
                    heur.append(abs(h1 - h2))
                else:
                    heur.append(0)

            for k in range(len(tops)):
                if heur[k] > wt[k]:
                    return False

            if self.end != 0:
                return False

            return True
```

```
        def check_admissibility(self, res):
            evr_list = []
            # ic(self.graph)
            for wt in self.graph:
                for j in wt:
                    evr_list.append(j)
            # ic(evr_list)
            tops = []
            for name in list(res):
                name = ord(name) - ASCII
                tops.append(name)
            wt = []
            # ic(self.get_connections(tops[0]))
            for i in range(len(tops)):
                if i != 0:
                    wt.append(self.get_weight(tops[i     -     1],
tops[i]))
                else:
                    wt.append(0)

            heur = []
            for j in range(len(tops)):
                if j != 0:
                    h1 = abs(tops[j] - self.end)
                    heur.append(h1)
                else:
                    heur.append(0)

            path_w = []
            for k in range(len(tops)):
                if k != 0:
                    path_w.append(sum(wt[k+1:]))
                else:
                    path_w.append(sum(wt))
```

```
            for u in range(len(tops)):
                if heur[u] > path_w[u]:
                    return False
            return True

        def a_star_search(self, start, end):
            frontier = PriorityQueue()
            frontier.put(start, 0)
            came_from = {}
            cost_so_far = {}
            came_from[start] = None
            cost_so_far[start] = 0

            while not frontier.empty():
                current = frontier.get()

                if current == end:
                    break

                for next_ in self.graph[current]:
                    new_cost = cost_so_far[current] + next_[-1]
                    if next_[0] not in cost_so_far or new_cost <
cost_so_far[next_[0]]:
                        cost_so_far[next_[0]] = new_cost
                        priority  =  new_cost  +  heuristic(end,
next_[0])
                        frontier.put(next_[0], priority)
                        came_from[next_[0]] = current
            return came_from, cost_so_far

        def create_graph(self):
            self.graph = [[] for i in range(ord('a'), ord('z'))]
            for line in stdin:
                start, end, weight = line.split()
                begin = ord(start) - ASCII
```

```
            stop = ord(end) - ASCII
            self.graph[begin].append([stop, float(weight)])

    def show_res(self):
        x, y = self.a_star_search(self.start, self.end)
        ans = "" + chr(self.end + ASCII)
        cur = x[self.end]
        while cur is not None:
            ans += chr(cur + ASCII)
            cur = x[cur]
        return ans[::-1]


if __name__ == '__main__':
    start_, end_ = input().split()
    gr = Graph(start_, end_)
    gr.create_graph()
    gr.start = ord(start_) - ASCII
    gr.end = ord(end_) - ASCII
    ans = gr.show_res()
    print(ans)
    if not gr.check_monotone(ans):
        print("Эвристическая функция не монотонна")
    else:
        print("Эвристическая функция монотонна")

    if not gr.check_admissibility(ans):
        print("Эвристическая функция не допустима")
    else:
        print("Эвристическая функция допустима")
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица 1 - Примеры тестовых случаев для жадного алгоритма

№ п/п	Входные данные	Выходные данные	Комментарии
1.	a l a b 1 a f 3 b c 5 b g 3 f g 4 c d 6 d m 1 g e 4 e h 1 e n 1 n m 2 g i 5 i j 6 i k 1 j l 5 m j 3	abgenmjl	Программа работает корректно
2.	g j a b 1 a f 3 b c 5 b g 3 f g 4 c d 6 d m 1 g e 4 e h 1 e n 1 n m 2 g i 5	genmj	Программа работает корректно

	i j 6 i k 1 j l 5 m j 3		
3.	a j a b 1 b c 1 c d 1 d e 1 e j 1 a f 1 f g 1 g h 1 h i 1 i j 1	abcdej	Программа работает корректно

Таблица 2 - Примеры тестовых случаев для алгоритма A*

№ п/п	Входные данные	Выходные данные	Комментарии
1.	a l a b 1 a f 3 b c 5 b g 3 f g 4 c d 6 d m 1 g e 4 e h 1 e n 1 n m 2 g i 5 i j 6 i k 1	abgenmjl Эвристика не мотононна Эвристика допустима	Программа работает корректно

	j l 5 m j 3		
2.	g j a b 1 a f 3 b c 5 b g 3 f g 4 c d 6 d m 1 g e 4 e h 1 e n 1 n m 2 g i 5 i j 6 i k 1 j l 5 m j 3	genmj Эвристика не мотононна Эвристика допустима	Программа работает корректно
3.	a j a b 1 b c 1 c d 1 d e 1 e j 1 a f 1 f g 1 g h 1 h i 1 i j 1	afghij Эвристика не мотононна Эвристика допустима	Программа работает корректно