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. 9383 _____ Соседков К.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритмы поиска пути в ориентированном графе (жадный алгоритм и A*).

Задание.

Жадный алгоритм.

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 первой строке через пробел указываются начальная и конечная вершины Далее в каждой строке указываются ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Алгоритм A*.

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.

Задание (Вариант 7).

Перед выполнением A* выполнять предобработку графа: для каждой вершины отсортировать список смежных вершин по приоритету.

Выполнение работы.

При выполнении работы были реализованы два алгоритма поиска пути в ориентированном графе. Их описание представлено ниже.

Жадный алгоритм.

Входные данные: граф, начальная вершина(start), конечная вершина(end).

Выходные данные: строка, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

Данный алгоритм обходит граф, начиная с вершины start, по ребрам с минимальным весом, пока не встретится вершина end. Если какое либо ребро с минимальным весом заводит в тупик, алгоритм возвращается на шаг назад и берет следующее минимальное ребро.

Алгоритм A*.

Входные данные: граф, начальная вершина(start), конечная вершина(end).

Выходные данные: строка, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной.

В алгоритме A*, в отличии от жадного алгоритма, переход на следующую вершину зависит от функции f(n) = g(n)+h(n), где n – смежная вершина, g(n) – стоимость пути от начальной вершины до вершины n, h(n) – эвристическая стоимость от вершины n до вершины end. Переход будет выполнен на вершину с минимальным значением f(n).

Перед выполнением алгоритма A* выполняется предобработка графа. Для каждой вершины список смежных вершин отсортировывается по приоритету.

Описание основных функций и переменных.

greedy(graph, start, end) — реализация жадного алгоритма.

a_star(graph, start, end) — реализация алгоритма A*.

read_graph() - чтение графа.

heuristic(a, b) — эвристическая функция.

graph — словарь(ключ — вершина, значение — смежные вершины).

queue — очередь для хранения вершин.

pqueue — очередь с приоритетом для хранения вершин(используется в алгоритме A*).

graph_map - словарь(ключ — вершина, значение — вершина из которой был совершен переход, для начальной вершины значение равно None).

Результаты тестирования представлены в таблице 1.

Таблица 1. Результаты тестирования

Ввод	Вывод
a f a c 1.0 a b 1.0 c d 2.0 b e 2.0 d f 3.0 e f 3.0	abef
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ade

a f a c 1.0 a b 1.0 c d 2.0 b e 2.0 d f 3.0 e f 3.0	abef

Выводы.

При выполнении работы были изучены и реализованы алгоритмы поиска пути в ориентированном графе(жадный алгоритм и алгоритм A*).

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла: lab2_astar.py

```
import heapq

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

def a_star(graph, start, end):
	path_cost = {start:0}
	pqueue = []
	heapq.heappush(pqueue, (0, start))
	graph_map = {start: None}

	while len(pqueue):
		current = heapq.heappop(pqueue)[1]

		if current == end:
			break

		for node in graph[current]:
			cost = path_cost[current] + graph[current][node]
			if node not in path_cost or cost < path_cost[node]:
				path_cost[node] = cost
				priority = cost + heuristic(end, node)
				heapq.heappush(pqueue, (priority, node))
				graph_map[node] = current

	answer = end
	while graph_map[answer[0]]:
		answer = graph_map[answer[0]] + answer
	return answer



def read_graph():
```

```
    graph = {}
    while True:
        try:
            input_string = input()
        except EOFError:
            break
        if not input_string:
            break
        start_vertex, end_vertex, weight = input_string.split(' ')

        if start_vertex in graph:
            graph[start_vertex][end_vertex] = float(weight)
        else:
            graph[start_vertex] = {end_vertex: float(weight)}

        if end_vertex not in graph:
            graph[end_vertex] = {}

    for i in graph:
        graph[i] = dict(sorted(graph[i].items(), key=lambda x: x[1], reverse=False))
    return graph



if __name__ == '__main__':
    start, end = input().split(' ')
    graph = read_graph()
    print(a_star(graph, start, end))
```

Название файла: lab2_greedy.py

```
import heapq

def greedy(graph, start, end):
    queue = [start]
    visited = set()

    while len(queue):
```

```
            current = queue.pop()
            if current[-1] == end:
                return current

            for node in graph[current[-1]]:
                if current[-1]+node not in visited:
                    queue.append(current)
                    queue.append(current+node)
                    visited.add(current[-1]+node)
                    break


def read_graph():
    graph = {}
    while True:
        try:
            input_string = input()
        except EOFError:
            break
        if not input_string:
            break
        start_vertex, end_vertex, weight = input_string.split(' ')

        if start_vertex in graph:
            graph[start_vertex][end_vertex] = float(weight)
        else:
            graph[start_vertex] = {end_vertex: float(weight)}

        if end_vertex not in graph:
            graph[end_vertex] = {}

    for i in graph:
        graph[i] = dict(sorted(graph[i].items(), key=lambda x: x[1], reverse=False))
    return graph


if __name__ == '__main__':
    start, end = input().split(' ')
```

```
graph = read_graph()
print(greedy(graph, start, end))
```