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. 1304 _____ Маркуш А.Е.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Разработайте программу, которая решает задачу построения пути в ориентированном графе при помощи жадного алгоритма и метода методом A*.

Задание.

1. 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.

2. 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.

Выполнение работы.

Для решения задачи с помощью жадного алгоритма была написана представленная в файле Lab2_greedy_stepik в приложении А. Далее представлено описание функций и классов реализованных в данной программе:

1. Класс *Path,* который содержит информацию о пути между двумя вершинами графа, а именно: стартовую вершину, конечную вершину, максимальный размер ребра и массив, в котором содержится набор вершин составляющих путь между на началом и концом. В нём кроме метода *__init__()* есть метод *__str__()*, который возвращает строковое представление объекта класса как путь между вершинами.

2. Функция *inputs()*. Данная функция осуществляет считывание входных данных и возвращает объект класса *Path* и словарь, с вершинами графа.

Считывание заканчивается, когда перехватывается ошибка EOFError или ValueError в блоке *try except*.

3. Функция *check_deadlock(point, graph, passed, path)*. Данная функция проверяет на то не зашли ли мы в вершину, являющуюся листом графа, но которая не является концом нужного нам пути. На вход принимает вершину графа, словарь, описывающий граф, список пройденных рёбер и объект класса *Path*. Возвращает вершину и bool-переменную, в которой содержится информация о том является ли данная вершина листом. С помощью трёх циклов *for* и трёх условных инструкций *if* осуществляется проверка является ли данная вершина листом, и если является, то удаляется ребро, по которому в эту вершину пришли, и мы возвращаемся в вершину, откуда пришли в лист, а также удаляем лист из нашего пути.

4. Функция *choose_the_shortest_path(point, path, passed, graph)*. Данная функция ищет ближайшую вершину к нынешней. На вход получает вершину, объект класса Path, список пройденных рёбер, словарь, описывающий граф. С помощью цикла, функция ищет вершину, в которую можно пройди из нынешней кратчайшим путём. Возвращает новую вершину и длину пути до неё.

5. Функция *find_way()*. Функция, решающая задачу поиска кратчайшего пути в орграфе с помощью жадного алгоритма. Возвращает объект класса *Path*. Данная функция вызывает функцию *inputs()* для считывания и в цикле *while* остальные описанные ранее функции для решения поставленной задачи.

6. Функция *main()*. Данная функция выводит ответ, возвращённый функцией *find_way()*.

Для решения задачи с помощью жадного алгоритма была написана представленная в файле Lab2_A_star_stepik.py в приложении А. Далее представлено описание функций и классов реализованных в данной программе:

1. Класс *Path* такой же как и в задаче с жадным алгоритмом, за исключением того, что в нём больше нет поля с максимальным размером ребра, а также была немного измен метод *__str__()*, для корректного вывода ответа.

2. Функция *inputs()* осталась такой же, как и в программе с жадным алгоритмом.

3. Функция *form_answer(path, current_node, came_from)*. Данная функция формирует ответ к заданию. На вход подаётся объет класса *Path*, последнюю пройденную вершину и словарь, описывающий из какой вершины мы пришли в данную. Возвращает объект класса Path. Ответ формируется из словаря, в котором записаны все пройденные нами рёбра. Начиная с конечной вершины нашего пути мы переходим по вершинам, откуда пришли, пока не дойдём до начальной вершины.

4. Функция *find_way()*. Данная функция решает задачу использую алгоритм A* для поиска кратчайшего пути в орграфе. Возвращает объект класса *Path*. Алгоритм A* реализован с помощью очереди с приоритетом и цикла *while*, который работает, пока очередь не опустеет(на случай если до конечной вершины нет пути) либо пока не придёт в нужный нам конечную вершину. Так были созданы два словаря, один из которых содержит пути, по которым мы передвигались между вершинами, а второй содержит текущую стоимость пути для каждой из вершин. В цикле из очереди извлекается минимальная по стоимости вершина и далее в очередь попадают все вершины, в которые можно пройти из нынешней вместе с их стоимостью. Так же информация о пути и стоимости заносится в два ранее созданных словаря.

5. Функция *main()* работает так же, как и в задаче с жадным алгоритмом.

Выводы.

Были разработаны программы, которые решают задачи построения пути в ориентированном графе при помощи жадного алгоритма и метода методом A*.

Жадный алгоритм был реализован итерационно с помощью цикла *while*, в котором выбиралась ближайшая непосещённая нами вершина.

Алгоритм A* так де реализован итерационно с помощью цикла *while* и очереди с приоритетом. Эвристической функцией для данной реализации являлось разница кодов символов нынешнего и конечного узла в ASCII таблице.

Алгоритм A* зачастую находит более оптимальный путь благодаря использованию эвристической функции.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Lab2_greedy_stepik.py

```
 # Данный класс содержит информацию о пути, который надо найти, а именно: стартовую вершину, конечную,
# максимальный размер ребра и массив, в котором содержится набор вершин составляющих путь между на началом и концом
class Path:

    # Инициализация класса
    def __init__(self, points):
        self.start_point = points[0]
        self.finish_point = points[1]
        self.way = [self.start_point]
        self.max_edge_len = None

     # Возвращает строковое представление объекта класса как путь между вершинами
    def __str__ (self):
        str = ''
        for element in self.way:
            str += element
        return str

# Данная функция осуществляет считывание входных данных и возвращает объект класса Path и словарь, с вершинами графа
def inputs():
    graph = {}
    path = Path(input().split())
    max_edge_len = 0
    while True:
        try:
            line = input().split()
            if line == []:
                break
            if line[0] not in graph:
                graph[line[0]] = []
            graph[line[0]].append([line[1], float(line[2])])
            if float(line[2]) > max_edge_len:
                max_edge_len = float(line[2])
            path.max_edge_len = max_edge_len
        except (ValueError, EOFError):
            return path, graph

# Данная функция проверяет на то не зашли ли мы в вершину, являющуюся листом графа,
# но которая не является концом нужного нам пути
# На вход принимает вершину графа, словарь, описывающий граф, список пройденных рёбер и объект класса Path
# Возвращает вершину и bool-переменную, в которой содержится информация о том является ли данная вершина листом
def check_deadlock(point, graph, passed, path):
    deadlock = False
```

```
    if point not in graph or graph[point] == []:
        path.way.pop()
        for passed_edge in passed:
            if passed_edge[0] == point:
                for edge in graph.items():
                    for node_name in edge[1]:
                        if passed_edge[0] == node_name[0]:
                            edge[1].remove(node_name)
                break
        point = path.way[-1]
        deadlock = True
    return point, deadlock

# Функция ищет ближайшую вершину к нынешней
# На вход получает вершину, объект класса Path, список пройдённых рёбер,
словарь, описывающий граф
# Возвращает новую вершину и длину пути до неё
def choose_the_shortest_path(point, path, passed, graph):
    min_edge_length = path.max_edge_len + 1
    new_point = point
    for i in range(len(graph[point])):
        if graph[point][i][1] < min_edge_length and (graph[point][i][0] not
in elemnt for elemnt in passed):
            min_edge_length = graph[point][i][1]
            new_point = graph[point][i][0]
    point = new_point
    return point, min_edge_length

# Функция, решающая задачу поиска кратчайшего пути в орграфе с помощью
жадного алгоритма
# Возвращает объект класса Path
def find_way():
    path, graph = inputs()
    point = path.start_point
    passed = []
    while point != path.finish_point:
        if point == path.finish_point:
            path.way.append(point)
            break
        point, deadlock = check_deadlock(point, graph, passed, path)
        if deadlock: continue
           point, min_edge_lenght = choose_the_shortest_path(point, path,
passed, graph)
        passed.append([point, min_edge_lenght])
        path.way.append(point)
    return path

def main():
    print(find_way())

if __name__ == "__main__":
    main()
```

Название файла: Lab2_A_star_stepik.py

```
import queue
```

```
# Данный класс содержит информацию о пути, который надо найти, а именно стартовую вершину, конечную
# и массив, в котором содержится набор вершин составляющих путь мужду на началом и концом
class Path:

    # Инициализация класса
    def __init__(self, points):
        self.start_point = points[0]
        self.finish_point = points[1]
        self.way = [self.start_point]

      # Возвращает строковое представление объекта класса как путь между вершинами
    def __str__(self):
        path_str = ''
        for element in self.way[::-1]:
            path_str += element
        return path_str

# Данная функция осуществляет считываение с клавиатуры и возвращает объект класса Path и словарь, с вершинами графа
def inputs():
    graph = {}
    path = Path(input().split())
    max_length = 0
    while True:
        try:
            line = input().split()
            if line == []:
                break
            if line[0] not in graph:
                graph[line[0]] = []
            graph[line[0]].append([line[1], float(line[2])])
            if float(line[2]) > max_length:
                max_length = float(line[2])
        except (ValueError, EOFError):
            return path, graph

# Данная функция формирует ответ к заданию
# На вход подаётся объет класса Path, последнюю пройденную вершину и словарь,
# описывающий из какой вершины мы пришли в данную
# Возвращает объект класса Path
def form_answer(path, current_node, came_from):
    path.way = [current_node]
    node = path.finish_point
    while node != path.start_point:
        node = came_from[current_node]
        path.way.append(node)
        current_node = node
    return path

# Данная функция решает задачу использую алгоритм A* для поиска кратчайшего пути в орграфе
# Возвращает объект класса Path
def find_way():
    path, graph = inputs()
```

```
    nodes_queue = queue.PriorityQueue(maxsize = 0)
    nodes_queue.put((0, path.start_point))
    came_from = {}
    cost_so_far = {}
    cost_so_far[path.start_point] = 0
    current_node = None
    while not nodes_queue.empty():
        current_node = nodes_queue.get()[1]
        if current_node == path.finish_point: break
        if current_node in graph:
            for next in graph[current_node]:
                new_cost = cost_so_far[current_node] + 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 + abs(ord(next[0]) -
ord(path.finish_point))
                    nodes_queue.put((priority, next[0]))
                    came_from[next[0]] = current_node
    return form_answer(path, current_node, came_from)

def main():
    print(find_way())

if __name__ == "__main__":
    main()
```