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 Мамин Р.А.

Преподаватель Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучение алгоритмов на графах. Изучение жадных алгоритмов, их сравнение с эвристическими алгоритмами, а также решение задачи поиска кратчайшего пути между 2 вершинами графа.

Задание.

1. Разработайте программу, которая решает задачу построения пути вориентированном 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 построениякратчайшего 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 первой задачи был разработан класс *Edge*, объектом которого является ребро графа со следующим списком методов и полей.

- Метод *__init__ (self, edge)* - конструктор класса, принимающий на вход массив из трёх елементов *edge* и заполняющий ими целочисленные поля *start, end, dist*, являющиеся начальным узлом ребра, конечным и весом ребра соответственно.

Класс *List*, объектом которого является двунаправленный список рёбер графа, отсортированный в порядке возрастания весов, со следующим списком методов и полей.

- Метод *insert(self, edge)* – принимает аргументом ребро *edge* и вставляет его в список, сортируя его после. Возвращает *None*.

Класс *Data,* объектом которого является структура введённой строки данных.

- Метод *__init__(self)* – заполняет начальными (пустыми строками) данными поля *start, end, edge, edges, path,* являющиеся начальным узлом, конечным узлом ребра(в виде строки), ребром, списком рёбер и путём соответственно.

Функция *inputKeyboard(),* принимающая на вход начальное ребро, конечное, путь, двунаправленный список и массив посещённых при переборе рёбер графа. В функции реализуется ввод данных с клавиатуры пользователем. Возвращаемое значение – объект класса *Data*.

Функция *writePath(start, end, path, list, visited),* принимающая на вход начальное ребро, конечное, путь, двунаправленный список и массив посещённых при переборе рёбер графа. В функции реализуется поиск пути с помощью жадного алгоритма, выбирающего ребро с наименьшим весом. Возвращаемое значение – *None*.

Ответы к тестированию программы представлены в таблице 1 Приложения Б.

Для решения второй задачи было реализованы следующие функции:

Функция *h(a, b),* принимающая на вход вершины *a* и *b и* и возвращающая эвристическое приближение, равное разнице кодов символов вершин.

Функция *get_min(open, closed),* принимающая на вход массив доступных вершин *open* и массив просмотренных *closed и* и возвращающая доступную вершину с наименьшим расстоянием, перебирая и сравнивая все доступные.

Функция *ASTAR(start, end, graph),* принимающая на вход начальную вершину *start* и конечную *end,* словарь вершин *graph* и возвращающая искомый ответ. Перед выполнением создаются описанные выше структуры и словарь map, в который будут заносить подходящие вершины(из него в конце формируется путь). Цикл while работает, пока существуют доступные для обработки элементы, содержащиеся в open. В начале каждой итерации находится элемент

из этого словаря с минимальным значением, соответствующим сумме расстояния от начала графа до данный вершины и ее эвристической функции. Если вершина 8 уже попадалась, она находится в close и повторно не обрабатывается. Для текущей вершины обрабатываются все ее дочерние вершины, они добавляются к map и их приоритет может быть перезаписан, если найдется элемент с меньшими значениями G_rate (расстояние от начала) и F_rate (эвристическая функция). Дочерние элементы добавляются к возможным на обработку. При нахождении конца графа, формируется путь.

Ответы к тестированию программы представлены в таблице 2 Приложения Б.

Исходный код программы представлен в Приложении А.

Выводы.

Были изучены основные алгоритмы на графах, такие как A* и жадный алгоритм. При сравнении двух алгоритмов было получено, что жадный алгоритм, выбирая локально лучший результат не всегда вычисляет глобально лучшее решения. Также был изучен эвристический подход к решению задач. С помощью алгоритма A* был найден кратчайший путь между 2 вершинами в ориентированном графе. На платформе *Stepik* были успешно пройдены проверки и обе программы оказались верными.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:
Название файла: greed.py

```
class Edge:
    next = None
    prev = None

    def __init__(self, edge):
        self.start = edge[0]
        self.end = edge[1]
        self.dist = float(edge[2])


class List:
    head = None
    cur = None

    def insert(self, edge):
        if self.head is None:
            self.head = edge
        else:
            self.cur = self.head
            while self.cur is not None:
                if edge.dist < self.cur.dist:
                    if self.cur == self.head:
                        self.cur.prev = edge
                        edge.next = self.cur
                        self.head = edge
                        break
                    else:
                        prev = self.cur.prev
                        prev.next = edge
                        edge.prev = prev
                        edge.next = self.cur
                        self.cur.prev = edge
                else:
                    if self.cur.next is not None:
                        if edge.dist > self.cur.next.dist:
                            self.cur = self.cur.next
                            continue
                        else:
                            edge.next = self.cur.next
                            edge.prev = self.cur
                            self.cur.next = edge
                            break
```

```
                    else:
                        self.cur.next = edge
                        edge.prev = self.cur
                        break
                self.cur = self.cur.next


class Data:
    start, end = '', ''
    edge = ''
    edges = List()
    path = []


def inputKeyboard():
    data = Data()
    i = 0

    while True:
        if i == 1:
            try:
                string = input()
            except EOFError:
                break
            if not string:
                break
            data.edge = Edge(string.split(" "))
            data.edges.insert(data.edge)
        if i == 0:
            data.start, data.end = input().split()
            i += 1
            continue
    return data


def writePath(start, end, path, list, visited):
    list.cur = list.head
    while list.cur is not None:
        if list.cur.start == start:
            if list.cur in visited:
                list.cur = list.cur.next
                continue
            if list.cur.end == end:
                path.append(list.cur.end)
                path.append(list.cur.start)
                return
            temp = list.cur
            visited.append(list.cur)
            writePath(list.cur.end, end, path, list, visited)
            list.cur = temp
```

```
            if list.cur is None:
                break
            if len(path) and path[0] == end:
                path.append(list.cur.start)
                return
        list.cur = list.cur.next


data = inputKeyboard()

visited = []

writePath(data.start, data.end, data.path, data.edges, visited)
data.path.reverse()
print(''.join(data.path))
```

Название файла: ASTAR.py

```
def h(a, b):
    return float(abs(ord(b) - ord(a)))


def get_min(open, closed):
    min = 10000
    for i in open:
        if i not in closed:
            if min > open[i]:
                min = open[i]
    for i in open:
        if i not in closed:
            if open[i] == min:
                return i


def ASTAR(start, end, graph):
    # множество уже пройденных вершин
    closed = []
    G_rate = {start: 0}
    # множество частных решений, в нем же хранятся F_rate
    open = {start: h(start, end)}
    map = {start: None}
    while len(open):
        cur = get_min(open, closed)
        if cur in closed:
            continue
        if cur == end:
            res = cur
            while map[cur]:
```

```
                res += map[cur]
                cur = map[cur]
            res = res[::-1]
            return res
        closed.append(cur)
        # добавляем смежные вершины
        for child in graph[cur]:
            temp_g = G_rate[cur] + graph[cur][child]
            if child not in open or temp_g < G_rate[child]:
                map[child] = cur
                G_rate[child] = temp_g
                temp_f = temp_g + h(child, end)
                if child not in open:
                    open[child] = temp_f


if __name__ == '__main__':
    graph = {}
    start, end = input().split()
    while True:
        try:
            string = input().split()
        except EOFError:
            break
        if not string:
            break
        if string[0] not in graph.keys():
            graph[string[0]] = {}
        graph[string[0]][string[1]] = float(string[2])
        if string[1] not in graph.keys():
            graph[string[1]] = {}

    print(ASTAR(start, end, graph))
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММ

Таблица 1 – Ход программы test_greed.py.

№ п/п	Данные	Вывод	Результат
1	a e a b 7 b e 2	abe	Программа работает верно
2	a b a b 1.0 a c 1.0	ab	Программа работает верно
3	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	abcdf	Программа работает верно
4	a d a b 1.0 b c 1.0 c a 1.0 a d 8.0	abcad	Программа работает верно
5	a b a b 1.0	ab	Программа работает верно

Таблица 2 – Ход программы ASTAR.py.

№ п/п	Данные	Вывод	Результат
1	a e a b 7 b e 2	abe	Программа работает верно
2	a b a b 1.0 a c 1.0	ab	Программа работает верно
3	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	Программа работает верно
4	a d a b 1.0 b c 1.0 c a 1.0 a d 8.0	ad	Программа работает верно
5	a b a b 1.0	ab	Программа работает верно