МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Жадный алгоритм и A*.

Студент гр. 8382 _____ Терехов А.Е.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм работы жадного алгоритма и алгоритма A*.

Задание.

Вариант 1. В A* вершины именуются целыми числами (в т. ч. отрицательными).

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.

Пример входных данных

a e

a b 3.0

b c 1.0

c d 1.0

a d 5.0

d e 1.0

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. Для приведённых в примере входных данных ответом будет

ade

Описание структур.

С исходным кодом жадного алгоритма можно ознакомиться в приложении А.

Для хранения графа были использованы списки смежности, реализованные на базе словаря и класса Edge. Ключ словаря является началом дуги, а вся

остальная информация содержится в классе Edge имеющем два поля: конец дуги и стоимость передвижения. Все остальные используемые структуры такие как списки, числа, строки, символы предоставлены языком Python 3.7.

С исходным кодом алгоритма A* можно ознакомиться в приложениях Б и В.

Для алгоритма A* так же было выбрано хранение графа с использованием списков смежности только без лишних структур. Списки смежности представляют собой словарь, ключи которого это вершины, из которых существует хоть один путь. Значение – это тоже словарь, но в нем ключи – это концы дуг, а значение – стоимость. То есть это выглядит следующим образом:

{ начало_дуги : { конец_дуги : стоимость } }.

Описание алгоритма.

Жадный алгоритм очень просто устроен. В начале инициализируем путь и текущую вершину стартовой вершиной. Затем пока текущая рассматриваемая вершина не равна конечной выбираем самого дешевого соседа, добавляем его в путь, изменяем его вес на очень большой, чтобы избежать зацикливания, и переходим в соседа. Если же был встречен тупик или все соседи вершины были рассмотрены, то просто возвращаемся в предыдущую вершину и удаляем из пути тупик. Если путь станет пуст, то это скажет о том, что пути просто не существует.

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые могут быть ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь. Составляющая g(x) — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение f(x), после чего

этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению f(x) = g(x) + h(x). Алгоритм продолжает свою работу до тех пор, пока множество раскрытых вершин не опустеет. Из множества решений выбирается решение с наименьшей стоимостью.

Множество просмотренных вершин хранится в closedset, а требующие рассмотрения пути — в множестве openset. Приоритет пути вычисляется с помощью функции f(x) путем выбора первого минимального.

Сложность алгоритма.

Сложность жадного алгоритма в случае, если поиск минимальной дуги происходит по списку или массиву равна O(|E|+|V|). Для A* все сложнее, его сложность зависит от используемой эвристической функции. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию: |h(x)-h*(x)| < O(log(h*(x))). Где h*(x) – оптимальная эвристика, то есть точное расстояние от x до цели.

Тестирование.

Проверим жадный алгоритм на следующем тесте:

Таблица 1. Тестирование жадного алгоритма.

Input	Output
a l a b 1 a f 3 b g 3 b c 5 f g 4 c d 6 d m 1 g e 4 g i 5 e h 1 e n 1 n m 2 i k 1 i j 6 j l 5 m j 3	from a to b from b to g from g to e from e to h from h to e from e to n from n to m from m to j from j to l abgenmjl

Для A* были написаны два теста задающие одинаковые графы, но с разными названиями вершин, в одном это буквы, а в другом целые числа, в том числе и отрицательные.

Таблица 2. Тестирование алгоритма A* (начало).

Input	Output
a f a b 1 a c 1 b d 2 b f 6 c b 2 c e 4 d c 3 d e 1 d f 3 e f 4	a append to openset openset: ['a'] a pop from openset openset: [] a append to closedset closedset: ['a'] b append to openset openset: ['b'] To b from a fromset {'b': 'a'} c append to openset openset: ['b', 'c']

Таблица 2. Тестирование алгоритма A* (конец).

	To c from a fromset {'b': 'a', 'c': 'a'} c pop from openset openset: ['b'] c append to closedset closedset: ['a', 'c'] e append to openset openset: ['b', 'e'] To e from c fromset {'b': 'a', 'c': 'a', 'e': 'c'} b pop from openset openset: ['e'] b append to closedset closedset: ['a', 'c', 'b'] d append to openset openset: ['e', 'd'] To d from b fromset {'b': 'a', 'c': 'a', 'e': 'c', 'd': 'b'} f append to openset openset: ['e', 'd', 'f'] To f from b fromset {'b': 'a', 'c': 'a', 'e': 'c', 'd': 'b', 'f': 'b'} d pop from openset openset: ['e', 'f'] d append to closedset closedset: ['a', 'c', 'b', 'd'] To e from d fromset {'b': 'a', 'c': 'a', 'e': 'd', 'd': 'b', 'f': 'b'} To f from d fromset {'b': 'a', 'c': 'a', 'e': 'd', 'd': 'b', 'f': 'd'} e pop from openset openset: ['f'] e append to closedset closedset: ['a', 'c', 'b', 'd', 'e'] a b d f

Таблица 3. Тестирование Алгоритма A* с вершинами в целых числах.

Input	Output
-1 3 -1 1 1 -1 -2 1 1 2 2 1 3 6 -2 1 2 -2 -3 4 2 -2 3 2 -3 1 2 3 3 -3 3 4	-1 append to openset openset: [-1] -1 pop from openset openset: [] -1 append to closedset closedset: [-1] 1 append to openset openset: [1] To 1 from -1 fromset {1: -1} -2 append to openset openset: [1, -2] To -2 from -1 fromset {1: -1, -2: -1} 1 pop from openset openset: [-2] 1 append to closedset closedset: [-1, 1] 2 append to openset openset: [-2, 2] To 2 from 1 fromset {1: -1, -2: -1, 2: 1} 3 append to openset openset: [-2, 2, 3] To 3 from 1 fromset {1: -1, -2: -1, 2: 1, 3: 1} 2 pop from openset openset: [-2, 3] 2 append to closedset closedset: [-1, 1, 2] -3 append to openset openset: [-2, 3, -3] To -3 from 2 fromset {1: -1, -2: -1, 2: 1, 3: 1, -3: 2} To 3 from 2 fromset {1: -1, -2: -1, 2: 1, 3: 2, -3: 2} -2 pop from openset openset: [3, -3] -2 append to closedset closedset: [-1, 1, 2, -2] -1 1 2 3

Очевидно, что результаты немного отличаются. Это связано с эвристической функцией, она зависит от названий вершин, а не от расстояния, то есть, например, F – конечная вершина, и пусть она связана с A как соседи первого порядка, а с вершиной E как соседи пятого порядка, тогда эвристика от A будет равна 5, а от E – 1, что сложно назвать корректным. Но тем не менее ответы все равно сошлись.

Вывод.

В ходе работы были реализованы два алгоритма, находящие минимальный путь в графе: жадный и A*. Жадный алгоритм находит не самый короткий путь, так как в нем выбирается локально минимальная дуга. A* в свою очередь находит минимальный путь так как при его использовании хранится длина пути от стартовой до текущей. Характерной чертой алгоритма A* является эвристическая функция, которая оказывает влияние на приоритет выбора вершины для рассмотрения. При использовании корректной эвристической функции можно добиться уменьшения временной сложности, при некорректной не только можно увеличить время выполнения, но и получать некорректный результат.

ПРИЛОЖЕНИЕ А

Исходный код жадного алгоритма со степика.

```
from sys import stdin
from math import inf

class Edge:
    end = ''
    weight = 0
    def __init__(self, e, w):
        self.end = e
        self.weight = w

    def __str__(self):
        return str(self.end) + " " + str(self.weight)

    def __lt__(self, other):
        return (self.weight < other.weight)
    def __le__(self, other):
        return (self.weight <= other.weight)


if __name__ == "__main__":
    start, finish = input().split()
    edges = {}
    for line in stdin:
        u,v,c = line.replace('\n', '').split()
        if (u in edges.keys()):
            edges[u].append(Edge(v, float(c)))
        else:
            edges[u] = [Edge(v, float(c))]
    way = start
    cur = start
    while cur != finish:
        if cur in edges.keys():
            minimum = min(edges[cur])
            if minimum.weight != inf:
                minimum.weight = inf
                way += minimum.end
                print("from",cur,"to",minimum.end)
                cur = minimum.end
            else:
                try:
                    print("from", way[-1], "to", way[-2])
                    way = way[:-1]
                    cur = way[-1]
                except IndexError:
                    print("No way.")
                    exit()
        else:
            try:
                print("from", way[-1], "to", way[-2])
                way = way[:-1]
                cur = way[-1]
            except IndexError:
                print("No way.")
                exit()
    print(way)
```

ПРИЛОЖЕНИЕ Б

Исходный код A* со степика.

```
from sys import stdin
from math import inf
def h(u, finish):
    return abs(ord(finish) - ord(u))

def recPath(fset, start, finish):
    path = []
    curr = finish
    path.append(curr)
    while curr != start:
        curr = fromset[curr]
        path.append(curr)
    return "".join(reversed(path))

if __name__ == "__main__":
    start, finish = input().split()
    graph = {}
#   чтение графа
    for line in stdin:
        u,v,c = line.replace('\n', '').split()
        if u not in graph.keys():
            graph[u] = {v:float(c)}
        else:
            graph[u][v] = float(c)
#   Алгоритм A*
    closedset = []
    openset = []
    openset.append(start)
    fromset = {}
    g = {}
    g[start] = 0
    f = {}
    f[start] = g[start] + h(start, finish)
    while len(openset) > 0:
#       поиск минимального по f
        curr = openset[0]
        for u in openset:
            if f[curr] > f[u]:
                curr = u
        if curr == finish:
            break
        openset.pop(openset.index(curr))
        closedset.append(curr)
        if (curr in graph.keys()):
            for neib in graph[curr]:
                # print(graph[curr][neib])
                if neib in closedset:
                    continue
                tentScore = g[curr] + graph[curr][neib]
                if neib not in openset:
                    openset.append(neib)
                    tentIsBetter = True
                else:
                    tentIsBetter = True if tentScore < g[neib] else False
                if tentIsBetter:
                    fromset[neib] = curr
                    g[neib] = tentScore
                    f[neib] = g[neib] + h(neib, finish)
    print("".join(map(str, recPath(fromset, start, finish))))
```

ПРИЛОЖЕНИЕ В

Исходный код A* со спецификациями по варианту.

```
#Terekhov_8382_v#1
from sys import stdin
from math import inf

def h(u, finish):
    return abs(finish - u)

def recPath(fset, start, finish):
    path = []
    curr = finish
    path.insert(0, curr)
    while curr != start:
        curr = fromset[curr]
        path.insert(0, curr)
    return path

if __name__ == "__main__":
    start, finish = map(int, input().split())
    graph = {}
#   чтение графа
    for line in stdin:
        u,v,c = line.replace('\n', '').split()
        if int(u) not in graph.keys():
            graph[int(u)] = {int(v):float(c)}
        else:
            graph[int(u)][int(v)] = float(c)
#   Алгоритм A*
    closedset = []
    openset = []
    openset.append(start)
    print(start,"append to openset")
    print("openset:", openset)
    fromset = {}
    g = {}
    g[start] = 0
    f = {}
    f[start] = g[start] + h(start, finish)
    while len(openset) > 0:
#       поиск минимального по f
        curr = openset[0]
        for u in openset:
            if f[curr] > f[u]:
                curr = u
        if curr == finish:
            break
        openset.pop(openset.index(curr))
        print(curr, "pop from openset")
        print("openset:", openset)
        closedset.append(curr)
        print(curr, "append to closedset")
        print("closedset:", closedset)
        if (curr in graph.keys()):
            for neib in graph[curr]:
                # print(graph[curr][neib])
                if neib in closedset:
                    continue
                tentScore = g[curr] + graph[curr][neib]
                if neib not in openset:
                    openset.append(neib)
```

```
                print(neib, "append to openset")
                print("openset:", openset)
                tentIsBetter = True
            else:
                tentIsBetter = True if tentScore < g[neib] else False
            if tentIsBetter:
                fromset[neib] = curr
                print("To", neib, "from", curr)
                print("fromset",fromset)
                g[neib] = tentScore
                f[neib] = g[neib] + h(neib, finish)
print(" ".join(map(str, recPath(fromset, start, finish))))
```