

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: Алгоритмы на графах

Студент гр. 8382

Чирков С.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Написать программу для поиска наименьшего пути в графе между двумя заданными вершинами, используя жадный алгоритм и 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

Входные данные:

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

Выходные данные:

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

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
```

Соответствующие выходные данные

```
abcde
```

Задание 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.

Входные данные:

Начальная и конечная вершины, ребра с их весами.

Выходные данные:

Минимальный путь из начальной вершины в конечную, написанный слитно.

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
```

Соответствующие выходные данные

```
ade
```

Вариант дополнительного задания.

Вар. 8. Перед выполнением A* выполнять предобработку графа: для каждой вершины отсортировать список смежных вершин по приоритету.

Описание алгоритма

Жадный алгоритм:

Сначала словарь с данными графа сортируется так, чтобы вершина с минимальным весом была первой в нём. Далее на каждом шаге выбирается соседняя не посещённая вершина с минимальным весом ребра, обращаясь к первому элементу словаря. Если вершина была посещена, то она удаляется из словаря. Алгоритм записывает свой текущий пройденный путь в строку ответа.

Процесс повторяется до тех пор, пока не будет обработана конечная вершина или вершины закончатся. Такой алгоритм не гарантирует нахождение оптимального решения при его существовании.

Алгоритм A*:

Перед началом алгоритма A* выполняется сортировка графа по сумме начального веса ребра и эвристической функции, в данном случае ею является близость символов, обозначающих вершины графа, в таблице ASCII. На каждом шаге алгоритма ищем наименьший приоритет до соседа через текущую вершину и обновляем значение пройденного пути до соседней вершины. Отличие алгоритма A* от алгоритма Дейкстры заключается в том, что переход к следующей вершине выполняется по приоритету, который равен сумме длины пройденного пути к этой вершине и значения эвристической функции этой вершины. Эвристическая функция должна быть монотонна и допустима, иначе алгоритм A* будет асимптотически хуже алгоритма Дейкстры. Правильно подобранная эвристическая функция в ряде прикладных задач позволяет значительно ускорить поиск пути, уменьшив фронт вершин поиска. Алгоритм A* гарантирует нахождение оптимального решения, если оно существует, при условии, что эвристическая функция допустима, монотонна и определена в тех же единицах измерений, что и веса ребер. A* отличается от жадного алгоритма тем, что учитывает уже построенный путь и некоторую топологическую оценку нахождения целевой вершины.

Искомый минимальный приоритет $f(x)=g(x)+h(x)$, где $g(x)$ – длина пути до x , $h(x)$ – эвристическая функция.

Функция $h(x)$ должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояние к целевой вершине.

$$|h(x)-h^*(x)| \leq O(\log(h^*(x)))$$

Где $h^*(x)$ – оптимальная эвристика

Особенности реализации алгоритма.

Для реализации жадного алгоритма ребра записываются в словарь и в цикле, пока не программа не придет в конечную вершину ищем наименьшее ребро (путём сортировки и обращения к первому элементу словаря), исключая из списка уже пройденные.

Для реализации алгоритма A^* используется не только словарь, но и списки открытых и закрытых вершин. Открытый и закрытый список вершин выражен через список списков, описывающих начальную и конечную вершины, стоимость пути без и с учетом эвристики. Программа идет, постепенно заполняя вектор закрытых вершин, пока не дойдет до последней точки, опираясь на функцию $f(v)=g(v)+h(v)$ выбирает следующую вершину, по 1наименьшему значению f . Путь строится в конце с помощью закрытого списка.

Описание функций и структур данных

graph = { } – словарь, с помощью которого хранится граф.

q = [], u = [] – списки открытых и закрытых вершин соответственно.

В программе используются встроенные функции языка.

Тестирование.

Для A^* :

a g			
a b 3.0			
a c 1.0	b e		
b d 2.0	a b 1.0		
b e 3.0	a c 2.0		
d e 4.0	b d 7.0		
e a 3.0	b e 8.0	a e	
e f 2.0	a g 2.0	a b 3.0	a d
a g 8.0	b g 6.0	b c 1.0	a b 1.0
f g 1.0	c e 4.0	c d 1.0	b c 1.0
c m 1.0	d e 4.0	a d 5.0	c a 1.0
m n 1.0	g e 1.0	d e 1.0	a d 8.0

Ответ: ag Ответ: bge Ответ: ade Ответ: ad

Для Жадного алгоритма:

```

a g
a b 3.0
a c 1.0
b d 2.0
b e 3.0
d e 4.0
e a 3.0
e f 2.0
a g 8.0
f g 1.0
c m 1.0
m n 1.0
      b e
      a b 1.0
      a c 2.0
      b d 7.0
      b e 8.0
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0
      a g 2.0
      a d
      b g 6.0
      a b 1.0
      c e 4.0
      b c 1.0
      d e 4.0
      c a 1.0
      a d 8.0

abdefg  abcde  bge  abcad

```

Тестирование с промежуточными данными

```

введите начальную и конечную вершину
a e
введите ребра и их вес (пустая строка для завершения ввода)
a c 1.0
a b 1.0
b d 1.0
b e 1.0
a g 1.0
b g 1.0
c e 1.0
d e 1.0
g f 1.0
g e 1.0

граф до сортировки рёбер
{'a': [['c', 1.0], ['b', 1.0], ['g', 1.0]], 'b': [['d', 1.0], ['e', 1.0], ['g', 1.0]], 'c': [['e', 1.0]], 'd': [['e', 1.0]], 'e': [], 'g': [['f', 1.0], ['e', 1.0]]}
граф после сортировки рёбер
{'a': [['b', 1.0], ['c', 1.0], ['g', 1.0]], 'b': [['d', 1.0], ['e', 1.0], ['g', 1.0]], 'c': [['e', 1.0]], 'd': [['e', 1.0]], 'e': [], 'g': [['e', 1.0], ['f', 1.0]]}

состояние открытого списка : [['a', 'g', 1.0, -1.0], ['a', 'c', 1.0, 3.0], ['a', 'b', 1.0, 4.0]]
в закрытый список добавлена новая вершина g с приоритетом -1.0 : [['a', 'g', 1.0, -1.0]]

в открытый список добавлены пути из вершины g

состояние открытого списка : [['g', 'f', 2.0, 1.0], ['g', 'e', 2.0, 2.0], ['a', 'c', 1.0, 3.0], ['a', 'b', 1.0, 4.0]]
в закрытый список добавлена новая вершина f с приоритетом 1.0 : [['a', 'g', 1.0, -1.0], ['g', 'f', 2.0, 1.0]]

состояние открытого списка : [['g', 'e', 2.0, 2.0], ['a', 'c', 1.0, 3.0], ['a', 'b', 1.0, 4.0]]
в закрытый список добавлена новая вершина e с приоритетом 2.0 : [['a', 'g', 1.0, -1.0], ['g', 'f', 2.0, 1.0], ['g', 'e', 2.0, 2.0]]

Отсортированный массив закрытых вершин, формат: [начальная вершина, конечная вершина, стоимость пути, приоритет]
[['a', 'g', 1.0, -1.0], ['g', 'f', 2.0, 1.0], ['g', 'e', 2.0, 2.0]]
Ответ: age

```

Сложность алгоритма

Для жадного алгоритма имеем по времени выполнения: сортировка ребер $|E|\log|E|$, просмотр каждой вершины и каждого ребра $|V|+|E|$. Итого $O(|V| + |E|\log|E|)$. По памяти $O(|V|+|E|)$, $|V|$ - количество вершин, $|E|$ - количество рёбер.

Для алгоритма A^* оценка по времени зависит от эвристической функции. При хорошей эвристической функции алгоритм будет всегда двигаться в верном направлении - искомая сложность $O(|V|+|E|)$, а в худшем случае будут рассмотрены все пути, а значит сложность будет $O(|V|^{|E|})$. По памяти $O(|V|+|E|)$.

Выводы.

В результате выполнения работы была разработана программа для нахождения минимального пути во взвешенном графе с помощью алгоритма A^* и жадного алгоритма. Была проанализирована асимптотика данных алгоритмов, а также их корректность. Жадные алгоритмы очень быстрые, но не всегда могут обеспечить глобальное лучшее решение. Но обычно они проще и легче кодируются, чем их аналоги. Жадный поиск исследует перспективные направления, но может не найти кратчайший путь. Алгоритм Дейкстры хорош в поиске кратчайшего пути, но он тратит время на исследование всех направлений, даже бесперспективных. Алгоритм A^* использует и подлинное расстояние от начала, и оцененное расстояние до цели. Алгоритм A^* , по оценке, работает быстрее жадного алгоритма и более эффективен в использовании.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

Жадный алгоритм

```
graph={}          #граф - словарь
z = input().split(' ')
x = list(map(lambda x: ord(x)-96, z))
for i in range(x[1]):
    graph[chr(i+97)]=[] #инициализация начальных вершин

while 1:          #добавление ребер в словарь
    x = input().split(' ')
    if (x==""):
        break
    if (graph.get(x[0],1)==1):
        graph[x[0]]=[]
        graph[x[0]].append([x[1],float(x[2])])
    else:
        graph[x[0]].append([x[1],float(x[2])])

for value in graph.values(): #сортировка словаря по приоритету
    value.sort(key=lambda x: (x[1],x[0]))

ans=""
while z[0]!=z[1] and z[0]:
    if (graph.get(z[0],1)==1):
        graph[z[0]]=[]
    if (graph[z[0]]): #выбор вершины с самым дешевым путём
        ans+=z[0]
        z[0]=graph[z[0]][0][0]
        graph[ans[-1:]].pop(0)
    else:
        z[0]=ans[-1:]
        ans=ans[:-1]
    if z[0]==z[1]:
        ans+=z[1]
        print(ans)
    else:
        print('нет пути')
```

А*

```
graph={}          #граф - словарь
print("введите начальную и конечную вершину")
z = input().split(' ')
print("введите ребра и их вес (пустая строка для завершения ввода)")
x = list(map(lambda x: ord(x)-96, z))
for i in range(x[1]):
    graph[chr(i+97)]=[] #инициализация начальных вершин
while 1:          #добавление ребер в словарь
    x = input().split(' ')
    if (x==""):
        break
    if (graph.get(x[0],1)==1):
        graph[x[0]]=[]
```



```

graph[x[0]].append([x[1],float(x[2])])
else:
graph[x[0]].append([x[1],float(x[2])])
print("граф до сортировки рёбер")
print(graph)
for key, value in graph.items(): #сортировка словаря по приоритету
value.sort(key=lambda x: (x[1]-ord(keyg)+ord(z[1]),x[0]))
print("граф после сортировки рёбер")
print(graph)
print()
q=[] #массив открытых вершин
for item in graph[z[0]]:
q.append([z[0], item[0], item[1], item[1]-ord(item[0])+ord(z[1])])
u=[] #массив закрытых вершин
q.sort(key=lambda x: (x[3],x[2],x[1])) #сортировка массива по приоритету

while len(q)!=0: #алгоритм А*
if graph.get(q[0][1],1)==1:
graph[q[0][1]]=[]
u.append(q[0]) #добавление пройденной вершины
print("состояние открытого списка :",q)
print("в закрытый список добавлена новая вершина", q[0][1],"с минимальным приоритетом",
q[0][3],":",u)
print()
if(q[0][1]==z[1]):
break
for item in graph[q[0][1]]: #обновление открытых вершин
q.append([q[0][1], item[0], q[0][2]+item[1],q[0][2]+item[1]-ord(item[0])+ord(z[1])])
if graph[q[0][1]]:
print("в открытый список добавлены пути из вершины", q[0][1])
print()
q.pop(0) #удаление пройденной вершины
q.sort(key=lambda x: (x[3],x[2],x[1],abs(ord(x[0])-ord(x[1])))) #сортировка, чтобы снова на первом месте
стоял приоритетный шаг

u.sort(key=lambda x: (x[2],x[0]))
print("отсортированный массив закрытых вершин, формат: [начальная вершина, конечная вершина,
стоимость пути, приоритет]")
print(u)
print()
answer=""
now=z[1]
while now!=z[0]: #формирование ответа по массиву закрытых вершин
for x in u:
if x[1]==now:
answer+=now
now=x[0]
break
answer+=now
answer=answer[::-1]
print("ответ:", answer)

```