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 _____ 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.

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:

На каждом шаге выбирается последняя посещенная вершина, выбирается соседняя не посещённая вершина с минимальным весом ребра. 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*:

На каждом шаге проверяем, меньше ли расстояние до соседа через текущую вершину, и обновляем наименьший путь до соседней вершины. Текущая вершина на каждой итерации должна иметь минимальную дистанцию от начальной вершины, получаемая сортировкой массива закрытых вершин. Отличие алгоритма A* от алгоритма Дейкстры заключается в том, что к приоритету вершины прибавляется значение некоторой эвристической функции от этой вершины до целевой. 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.

Эвристическая функция $f(x)=g(x)+h(x)$

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

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

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 наименьшему 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 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

Ответ: ag
```

```
b e
a b 1.0
a c 2.0
b d 7.0
b e 8.0
a g 2.0
b g 6.0
c e 4.0
d e 4.0
g e 1.0

Ответ: bge
```

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0

Ответ: ade
```

```
a d
a b 1.0
b c 1.0
c a 1.0
a d 8.0

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

abdefg
```

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0

abcde
```

```
b e
a b 1.0
a c 2.0
b d 7.0
b e 8.0
a g 2.0
b g 6.0
c e 4.0
d e 4.0
g e 1.0

bge
```

```
a d
a b 1.0
b c 1.0
c a 1.0
a d 8.0

abcad
```

Тестирование с промежуточными данными

```
введите начальную и конечную вершину
a g
введите ребра и их вес (пустая строка для завершения ввода)
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

В закрытый список добавлена новая вершина c
В закрытый список добавлена новая вершина m
В закрытый список добавлена новая вершина n
В закрытый список добавлена новая вершина b
В закрытый список добавлена новая вершина d
В закрытый список добавлена новая вершина e
В закрытый список добавлена новая вершина g
Отсортированный массив закрытых вершин, формат: [начальная вершина, конечная вер
шина, стоимость пути, стоимость с учетом эвристики]
[['a', 'c', 1.0, 5.0], ['c', 'm', 2.0, -4.0], ['a', 'b', 3.0, 8.0], ['m', 'n', 3
.0, -4.0], ['b', 'd', 5.0, 8.0], ['b', 'e', 6.0, 8.0], ['a', 'g', 8.0, 8.0]]
Ответ: ag
```

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|).

Для алгоритма A* оценка по памяти O(|V|+|E|). Оценка по времени зависит от эвристической функции и используемой структуры для хранения очереди, списка посещенных вершин и т.п. Получается O(|E|log|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. Затраты памяти в написанных программах были одинаковы.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

```
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])])
for value in graph.values():  #сортировка словаря по приоритету
  value.sort(key=lambda x: (x[1],x[0]))
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 A*
  if graph.get(q[0][1],1)==1:
        graph[q[0][1]]=[]
  u.append(q[0])          #добавление пройденной вершины
  print("В закрытый список добавлена новая вершина", q[0][1])
  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])])
  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, стоимость с учетом эвристики]")
print(u)
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)
```