```
Class Graph:
  def init _(self, adjac_lis):
  self. adjac_lis = adjac-lis

 def get_neighbours (self, v):
 return self.adjac_lis[v]

 def h (self, n):
 H = {
        'S' : 14,
        'A' : 7,
        'B' : 12,
        'C' : 13,
        'D' : 5,
        'E' : 6,
        'G' : 7,
        'F' : 5,
        'H' : 2,
   }
   return H[n]
def a_star_algorithm (self, start, stop):
  open_lst = set([start])
  closed_lst = set ([])
  poo = {}
   poo [start] = 0
   ~~par~~{}
   par [start] = start

 while len(open_lst) > 0:
  n=None
  for v in open_lst:
    if n==None or poo[v]+self.h(v) < poo[n]+self.h(n):
      n=v:
 if n==None
   print('path does not exist')
   return None

 If n==stop:
    reconst_path=[]

  while par[n]!=n:
  reconst_path.append(n)
    n=par[n]

 reconst-path.append(start)
 reconst-path.reverse()
  print('path found:{}'.format(rconst-path))
  return reconst-path

for(m,weight) in self.get_neighbour(n):
   if m not in open-lst and m not in closed_lst:
   open_lst.add(m)
   par[m]=n
   poo[m]=poo[n]+weight

else:
   if poo[m]>poo[n]+weight
```

```
poo[m] = poo[n] + weight
par[m] = n

if m in closed_lst
    closed_lst.remove(m)
    open_lst.add(m)

open_lst.remove(n)
closed_lst.add(n)

print('path does not exist!')
return None

adjac_lis = {
        'S': [('A', 1), ('B', 1), ('C', 1)],
        'A': [('D', 1), ('E', 1)],
        'C': [('E', 1) ('G', 1)],
        'D': [('H', 1)],
    }

Graph1 = Graph(adjac_lis)
Graph1.a-star_algorithm('S', 'H')
```