def astaralgo(start\_node,stop\_node):

open\_set = set(start\_node)

closed\_set= set()

g={}

parents = {}

g[start\_node] = 0

parents[start\_node] = start\_node

while len(open\_set) > 0:

n=None

for v in open\_set:

if n==None or g[v] + heuristic(v) < g[n] + heuristic(n):

n=v

if n==stop\_node or Graph\_nodes[n]==None:

pass

else:

for(m,weight) in get\_neighbors(n):

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m]=n

g[m] = g[n] + weight

else:

if g[m] > g[n] + weight:

g[m] = g[n] + weight

parents[m]=n

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n==None:

print("path does not exist!")

return None

if n==stop\_node:

path=[]

while parents[n]!=n:

path.append(n)

n=parents[n]

path.append(start\_node)

path.reverse()

print("Path found: {}".format(path))

return path

open\_set.remove(n)

closed\_set.add(n)

print("Path does not exist!")

return None

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

def heuristic(n):

H\_dist = {

'A' : 11,

'B' : 6,

'C' : 99,

'D' : 1,

'E' : 7,

'G' : 0,

}

return H\_dist[n]

Graph\_nodes = {

'A' : [('B',2),('E',3)],

'B' : [('C',1),('G',9)],

'C' : None,

'E' : [('D',6)],

'D' : [('G',1)],

}

astaralgo('A','G')

