#!/usr/bin/env python

# coding: utf-8

# In[1]:

<https://youtu.be/ob4faIum4kQ>

<https://github.com/learlinian/UW-Madison-CS540-Introduction-to-AI/blob/master/3.%20A%20Start%20Search.py>

def aStarAlgo(start\_node, stop\_node): #A, #J

open\_set = set(start\_node) #set is used to store multiple dataTypes

closed\_set = set() #did not pass anything

g = {} #store distance from starting node

parents = {}# parents contains an adjacency map of all nodes

#ditance of starting node from itself is zero

g[start\_node] = 0

#start\_node is root node i.e it has no parent nodes

#so start\_node is set to its own parent node

parents[start\_node] = start\_node

while len(open\_set) > 0:

n = None

#node with lowest f() is found

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):

#nodes 'm' not in first and last set are added to first

#n is set its parent

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

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

#for each node m,compare its distance from start i.e g(m) to the

#from start through n node

else:

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

#update g(m)

g[m] = g[n] + weight

#change parent of m to n

parents[m] = n

#if m in closed set,remove and add to open

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop\_node

# then we begin reconstructin the path from it to the start\_node

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

# remove n from the open\_list, and add it to closed\_list

# because all of his neighbors were inspected

open\_set.remove(n)

closed\_set.add(n)

print('Path does not exist!')

return None

#define fuction to return neighbor and its distance

#from the passed node

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

#for simplicity we’ll consider heuristic distances given

#and this function returns heuristic distance for all nodes

def heuristic(n):

H\_dist = {

'A': 10,

'B': 8,

'C': 5,

'D': 7,

'E': 3,

'F': 6,

'G': 5,

'H': 3,

'I': 1,

'J': 0

}

return H\_dist[n]

#Describe your graph here

Graph\_nodes = {

'A': [('B', 6), ('F', 3)],

'B': [('C', 3), ('D', 2)],

'C': [('D', 1), ('E', 5)],

'D': [('C', 1), ('E', 8)],

'E': [('I', 5), ('J', 5)],

'F': [('G', 1),('H', 7)] ,

'G': [('I', 3)],

'H': [('I', 2)],

'I': [('E', 5), ('J', 3)],

}

aStarAlgo('A', 'J')