A\* ALGORITHM :

**def aStarAlgo(start\_node, stop\_node):**

**open\_set = set(start\_node)**

**closed\_set = set()**

**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': 11,**

**'B': 6,**

**'C': 99,**

**'D': 1,**

**'E': 7,**

**'G': 0,**

**}**

**return H\_dist[n]**

**#Describe your graph here**

**Graph\_nodes = {**

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

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

**'C': None,**

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

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

**}**

**aStarAlgo('A', 'G')**