

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[1]:
```

```
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
```

```
    #distance 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')
```