

Name: Richard Nadar

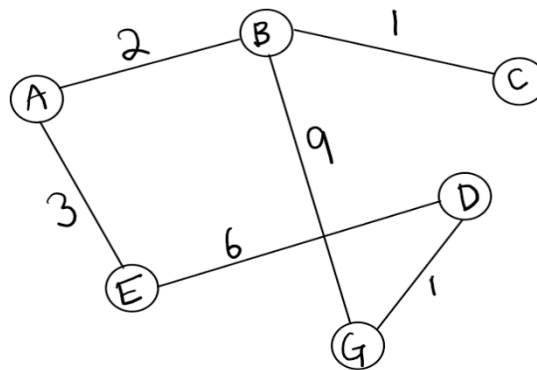
Reg No: RA1911030010109

AI LAB EXP-5

Implementation of A* Algorithm

AIM: To implement A* Algorithm using python

Graph:



Algorithm:

- *open set* is list of nodes which have been visited but neighbors haven't all been inspected whereas *closed set* is list of nodes which have been visited but neighbors have been inspected.
- *g* contains current distances from start node to all other nodes.
- *parents* contains adjacency map of all nodes
- we find a node with the lowest value of $f()$ - evaluation function
- if the current node is the *stop_node* then we begin reconstructing the path from it to the *start_node*
- if the current node isn't in both *open set* and *closed set* add it to *open set* and note *n* as it's parent
- otherwise, check if it's quicker to first visit *n*, then *m* and if it is, update parent data and *g* data and if the node was in the *closed set*, move it to *open set*
- remove *n* from the *open set*, and add it to *closed set* because all of his neighbors were inspected

CODE:

```
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 = {

```

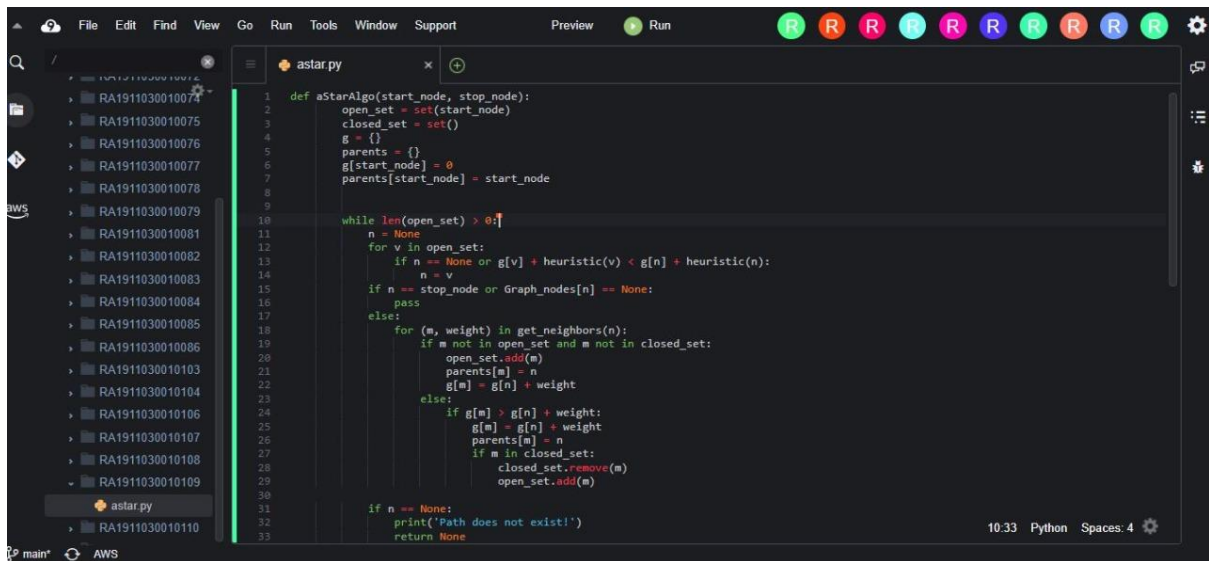
```
'A': 11,  
'B': 6,  
'C': 99,  
'D': 1,  
'E': 7,  
'G': 0,  
}
```

```
return H_dist[n]
```

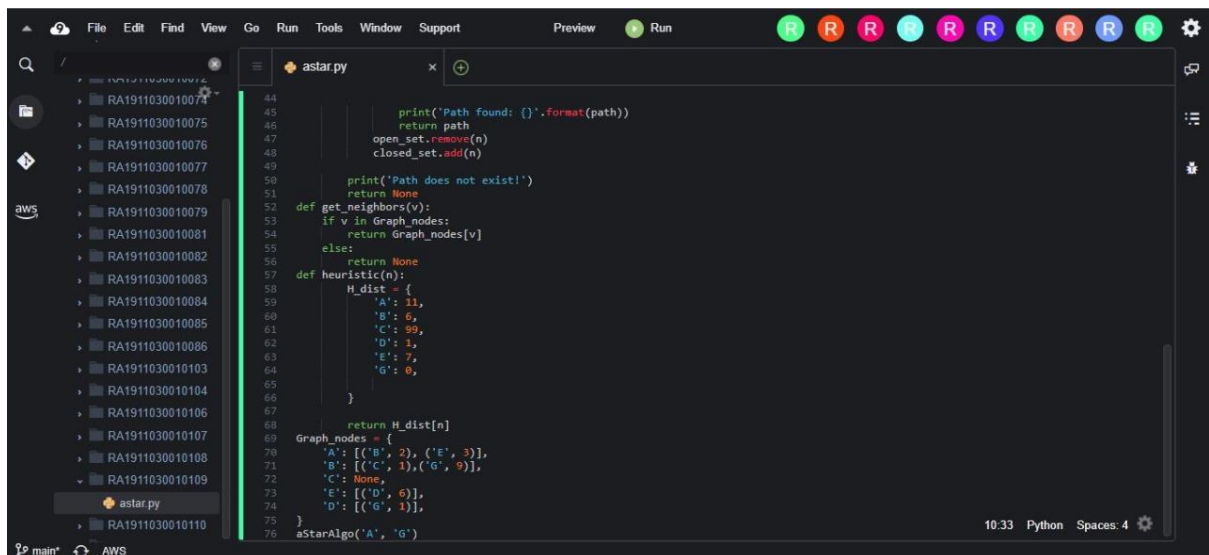
```
Graph_nodes = {  
    'A': [('B', 2), ('E', 3)],  
    'B': [('C', 1), ('G', 9)],  
    'C': None,  
    'E': [('D', 6)],  
    'D': [('G', 1)],  
}
```

```
aStarAlgo('A', 'G')
```

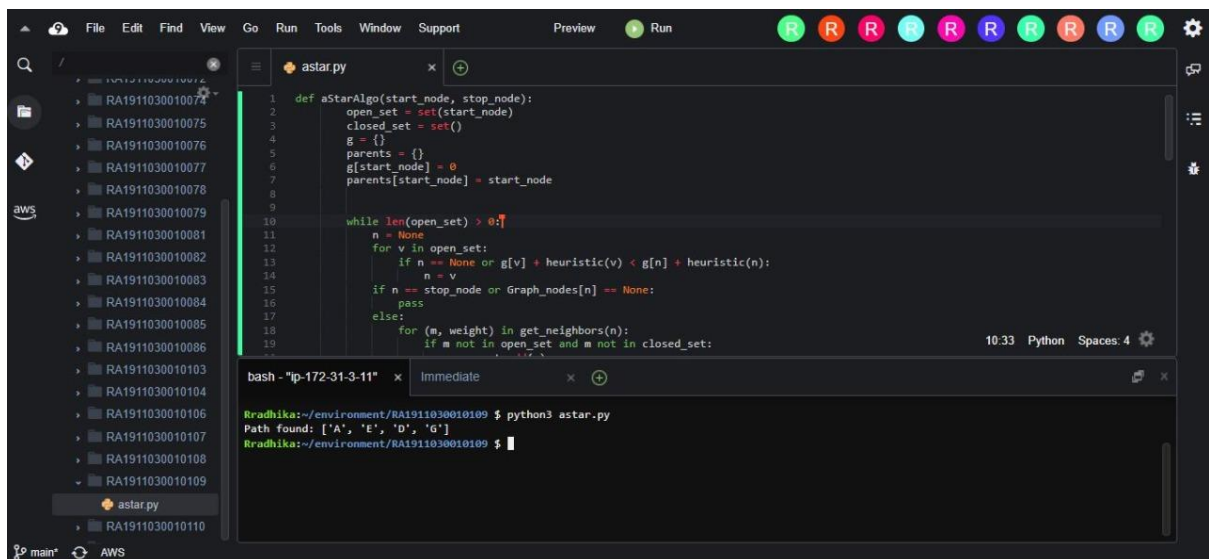
OUTPUT:



```
1 def aStarAlgo(start_node, stop_node):
2     open_set = set(start_node)
3     closed_set = set()
4     g = {}
5     parents = {}
6     g[start_node] = 0
7     parents[start_node] = start_node
8
9
10    while len(open_set) > 0:
11        n = None
12        for v in open_set:
13            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
14                n = v
15        if n == stop_node or Graph_nodes[n] == None:
16            pass
17        else:
18            for (m, weight) in get_neighbors(n):
19                if m not in open_set and m not in closed_set:
20                    open_set.add(m)
21                    parents[m] = n
22                    g[m] = g[n] + weight
23            else:
24                if g[m] > g[n] + weight:
25                    g[m] = g[n] + weight
26                    parents[m] = n
27                    if m in closed_set:
28                        closed_set.remove(m)
29                    open_set.add(m)
30
31    if n == None:
32        print('Path does not exist!')
33        return None
```



```
44     print('Path found: {}'.format(path))
45     return path
46     open_set.remove(n)
47     closed_set.add(n)
48
49     print('Path does not exist!')
50     return None
51
52     def get_neighbors(v):
53         if v in Graph_nodes:
54             return Graph_nodes[v]
55         else:
56             return None
57
58     def heuristic(n):
59         H_dist = {}
60         'A': 11,
61         'B': 6,
62         'C': 99,
63         'D': 1,
64         'E': 7,
65         'G': 0,
66     }
67
68     return H_dist[n]
69
70     Graph_nodes = {
71         'A': [('B', 2), ('E', 3)],
72         'B': [('C', 1), ('G', 9)],
73         'C': None,
74         'E': [('D', 6)],
75         'D': [('G', 1)],
76     }
77
78     aStarAlgo('A', 'G')
```



```
1 def aStarAlgo(start_node, stop_node):
2     open_set = set(start_node)
3     closed_set = set()
4     g = {}
5     parents = {}
6     g[start_node] = 0
7     parents[start_node] = start_node
8
9
10    while len(open_set) > 0:
11        n = None
12        for v in open_set:
13            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
14                n = v
15        if n == stop_node or Graph_nodes[n] == None:
16            pass
17        else:
18            for (m, weight) in get_neighbors(n):
19                if m not in open_set and m not in closed_set:
20                    open_set.add(m)
21                    parents[m] = n
22                    g[m] = g[n] + weight
23            else:
24                if g[m] > g[n] + weight:
25                    g[m] = g[n] + weight
26                    parents[m] = n
27                    if m in closed_set:
28                        closed_set.remove(m)
29                    open_set.add(m)
30
31    if n == None:
32        print('Path does not exist!')
33        return None
```

```
bash - "ip-172-31-3-11" x Immediate x +
Rradhika:~/environment/RA1911030010109 $ python3 astar.py
Path found: ['A', 'E', 'D', 'G']
Rradhika:~/environment/RA1911030010109 $
```

RESULT: Therefore A* algorithm has been implemented successfully.