

# Advanced Algorithms Lab-Min term

Prashanth.S  
(19MID0020)

## Code Structure

```
1 > def a_star_algorithm(starting_node, stop_node):...
53
54
55 > def get_neighbors(v):...
60
61
62 > def heuristic(n):...
66
67 > if __name__ == '__main__':...
81
```

## Code Snippet

```
1 def a_star_algorithm(starting_node, stop_node):
2     open_set = set(starting_node)
3     closed_set = set()
4     g = {}
5     parents = {}
6     g[starting_node] = 0
7     parents[starting_node] = starting_node
8
9     while len(open_set) > 0:
10         n = None
11         for v in open_set:
12             if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
13                 n = v
14
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
28         if m in closed_set:
29             closed_set.remove(m)
30             open_set.add(m)
31
32     if n == None:
33         print('The Path does not exist!')
34         return None
35
36     if n == stop_node:
37         path = []
38
39         while parents[n] != n:
40             path.append(n)
41             n = parents[n]
42         path.append(starting_node)
43         path.reverse()
44
45         print('Final Path from vertex-0 to vertex-4 : {}'.format(path))
46         return path
47
48     open_set.remove(n)
49     closed_set.add(n)
50
51 print('Path does not exist!')
52 return None
```

```

53
54
55 def get_neighbors(v):
56     if v in graph_nodes:
57         return graph_nodes[v]
58     else:
59         return None
60
61
62 def heuristic(n):
63     heurisitc_distances = {'0': 0, '1': 2, '2': 15, '3': 8,
64         '4': 0, '5': 8, '6': 5, '7': 4, '8': 1, }
65     return heurisitc_distances[n]
66
67 if __name__ == '__main__':
68     graph_nodes = {
69         '0': [('1', 4), ('7', 8)],
70         '1': [('0', 4), ('7', 11), ('2', 8)],
71         '2': [('1', 8), ('8', 2), ('5', 4), ('3', 7)],
72         '3': [('2', 7), ('5', 14), ('4', 9)],
73         '4': [('3', 9), ('5', 10)],
74         '5': [('4', 10), ('3', 14), ('2', 4), ('6', 2)],
75         '6': [('5', 2), ('8', 6), ('7', 1)],
76         '7': [('0', 8), ('1', 11), ('8', 7), ('6', 1)],
77         '8': [['2', 2], ['6', 6], ['7', 7]]
78     }
79
80     a_star_algorithm('0', '4')

```

## Output

Final Path from vertex-0 to vertex-4 : ['0', '7', '6', '5', '4']

## Code

```
def a_star_algorithm(starting_node, stop_node):
    open_set = set(starting_node)
    closed_set = set()
    g = {}
    parents = {}
    g[starting_node] = 0
    parents[starting_node] = starting_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('The Path does not exist!')
            return None

        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(starting_node)
```

```

        path.append(n)
        n = parents[n]
        path.append(starting_node)
        path.reverse()

        print('Final Path from vertex-0 to vertex-
4 : {}'.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):
    heurisitc_distances = {'0': 0, '1': 2, '2': 15, '3': 8,
        '4': 0, '5': 8, '6': 5, '7': 4, '8': 1, }
    return heurisitc_distances[n]

if __name__ == '__main__':
    graph_nodes = {
        '0': [('1', 4), ('7', 8)],
        '1': [('0', 4), ('7', 11), ('2', 8)],
        '2': [('1', 8), ('8', 2), ('5', 4), ('3', 7)],
        '3': [('2', 7), ('5', 14), ('4', 9)],
        '4': [('3', 9), ('5', 10)],
        '5': [('4', 10), ('3', 14), ('2', 4), ('6', 2)],
        '6': [('5', 2), ('8', 6), ('7', 1)],
        '7': [('0', 8), ('1', 11), ('8', 7), ('6', 1)],
        '8': [('2', 2), ('6', 6), ('7', 7)]
    }

    a_star_algorithm('0', '4')

```