Name:- Kaveri Sandip Gaikwad

Roll no: C03016

ASSIGNMENT NO:- 02

```
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list


    def get_neighbors(self, v):
        return self.adjacency_list[v]


    # heuristic function with distances from the current node to the goal node
    def h(self, n):
        H = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0
        }

        return H[n]


    def a_star_algorithm(self, start_node, stop_node):
        # open_list is a list of nodes which have been visited, but who's neighbors
        # haven't all been inspected, starts off with the start node
```

```
# closed_list is a list of nodes which have been visited
# and who's neighbors have been inspected
open_list = set([start_node])
closed_list = set([])

# g contains current distances from start_node to all other nodes
# the default value (if it's not found in the map) is +infinity
g = {}

g[start_node] = 0

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

while len(open_list) > 0:
    n = None

    # find a node with the lowest value of f() - evaluation function
    for v in open_list:
        if n == None or g[v] + self.h(v) < g[n] + self.h(n):
            n = v;

    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:
    reconst_path = []

    while parents[n] != n:
        reconst_path.append(n)
        n = parents[n]

    reconst_path.append(start_node)

    reconst_path.reverse()

    print('Path found: {}'.format(reconst_path))
    return reconst_path

# for all neighbors of the current node do
for (m, weight) in self.get_neighbors(n):
    # if the current node isn't in both open_list and closed_list
    # add it to open_list and note n as it's parent
    if m not in open_list and m not in closed_list:
        open_list.add(m)
        parents[m] = n
        g[m] = g[n] + weight

    # otherwise, check if it's quicker to first visit n, then m
    # and if it is, update parent data and g data
```

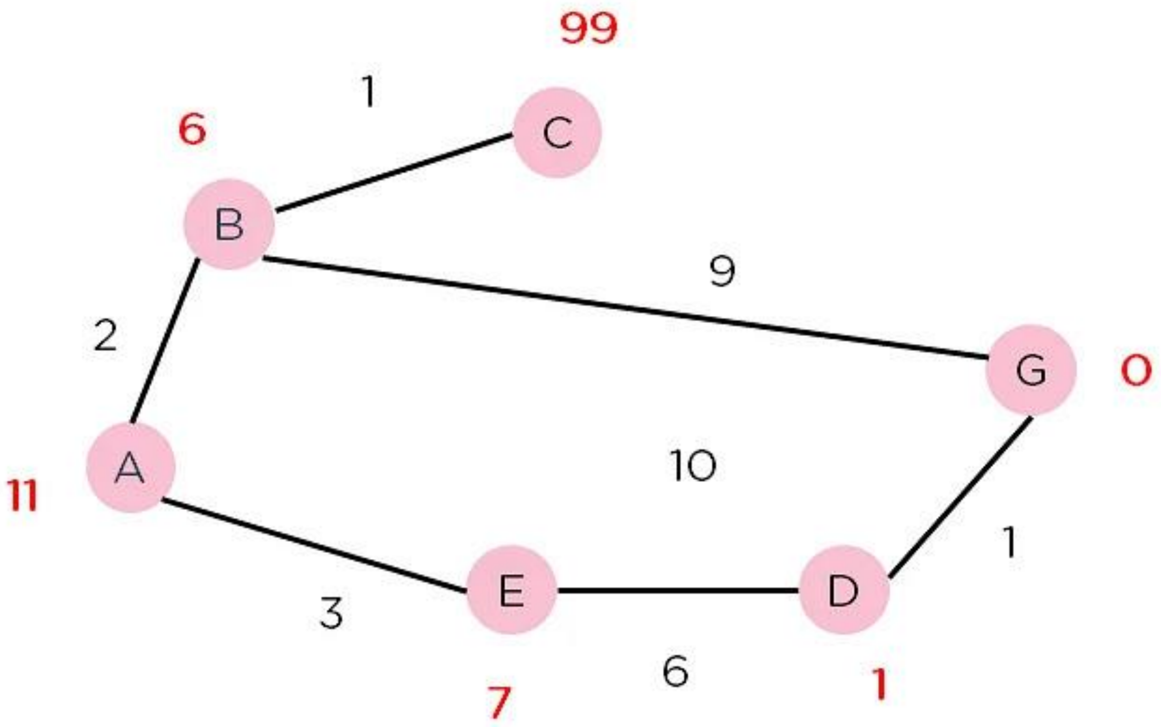
```
                # and if the node was in the closed_list, move it to open_list
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)

            # remove n from the open_list, and add it to closed_list
            # because all of his neighbors were inspected
            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')
        return None

adjac_lis = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'D': [('G', 1)],
    'E': [('D', 6)]
}
graph = Graph(adjac_lis)
graph.a_star_algorithm('A', 'G')
```



OUTPUT :

Path found= [A,E,D,G]