AI LAB EXP – 5b

A* ALGORITHM FOR REAL WORLD PROBLEMS

Tejas Ashok

RA1911030010090

AIM

To implement A* Algorithm using python.

ALGORITHM

- We create two lists – Open List and Closed List (just like Dijkstra Algorithm)
- Initialize the open list
- Initialize the closed list put the starting node on the open list (you can leave its f at zero)
- While the open list is not empty
    1. Find the node with the least f on the open list, call it "q"
    2. Pop q off the open list
    3. Generate q's 8 successors and set their parents to q
    4. For each successor
        i. If successor is the goal, stop search
        ii. Else, compute both g and h for successor
            - successor.g = q.g + distance between successor and q
            - successor.h = distance from goal to successor(This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics)
            - successor.f = successor.g + successor.h

        iii. If a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor
        iv. If a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list
        end (for loop)
        v. Push q on the closed list
        end (while loop)

CODE

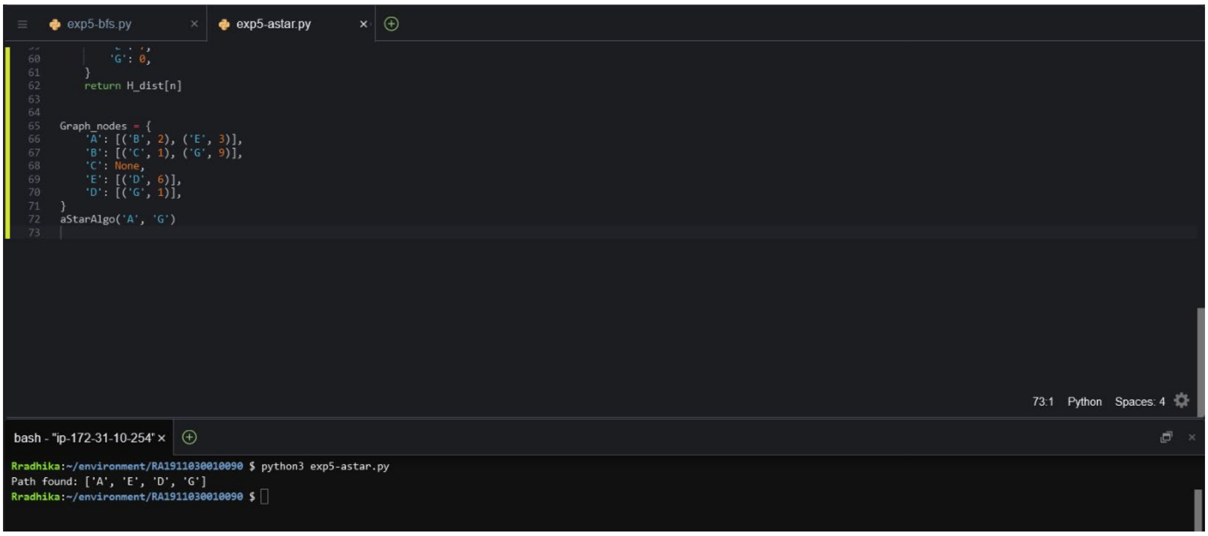
```
def aStarAlgo(start_node, stop_node):

    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



RESULT

A* search algorithm was successfully executed in python.