WEEK2 MACHINE INTELLIGENCE

ARYANSH BHARGAVAN

PES1UG20CS084

CODE

```
"""
You can create any other helper funtions.
Do not modify the given functions
"""
# Aryansh Bhargavan
# PES1UG20CS084
# Helper Function for DFS

def DFS_Traversal_helper(cost, start_point, t1, visited, goals):
    visited[start_point] = True
    t1.append(start_point)
    if (start_point not in goals):
        temp = cost[start_point]
        for i in range(len(temp)):
            if ((visited[i] == False) and (temp[i] > 0)):
                r = DFS_Traversal_helper(cost, i, t1, visited, goals)
                if r == -1:
                    return -1
                t1.pop()
    else:
        return -1
#-----------------------------------------------------------------------#

def A_star_Traversal(cost, heuristic, start_point, goals):

    n = len(cost)
    visited = [0 for i in range(n)]
    frontier_priority_queue = queue.PriorityQueue()
    frontier_priority_queue.put((heuristic[start_point], ([start_point],
start_point, 0)))

    while(frontier_priority_queue.qsize() != 0):

        total_estimated_cost, nodes_tuple = frontier_priority_queue.get()
        A_star_path_till_node = nodes_tuple[0]
        node = nodes_tuple[1]
        node_cost = nodes_tuple[2]

        if visited[node] == 0:
            visited[node] = 1

            if node in goals:
```

```
                return A_star_path_till_node

            for neighbour_node in range(1, n):
                if cost[node][neighbour_node] > 0 and visited[neighbour_node] ==
0:

                    total_cost_till_node = node_cost + cost[node]
[neighbour_node]
                    estimated_total_cost = total_cost_till_node +
heuristic[neighbour_node]

                    A_star_path_till_neighbour_node =
copy.deepcopy(A_star_path_till_node)
                    A_star_path_till_neighbour_node.append(neighbour_node)
                    frontier_priority_queue.put((estimated_total_cost,
(A_star_path_till_neighbour_node, neighbour_node, total_cost_till_node)))

    return list()

def DFS_Traversal(cost, start_point, goals):
    """
    Perform DFS Traversal and find the optimal path
        cost: cost matrix (list of floats/int)
        start_point: Staring node (int)
        goals: Goal states (list of ints)
    Returns:
        path: path to goal state obtained from DFS(list of ints)
    """
    path = []
    visited = [False]*(len(cost[0]))
    DFS_Traversal_helper(cost, start_point, path, visited, goals)
    return path
```

OUTPUT

```
PS D:\SEM5\MI\week2> .\SampleTest.py --SRN PES1UG20CS084
Test Case 1 for A* Traversal PASSED
Test Case 2 for DFS Traversal PASSED
PS D:\SEM5\MI\week2>
```