TASK 3

Implementation of **A * Algorithm** to find the optimal path using Python by following constraints.

3(A): A* Algorithm

Aim : To implement of A * Algorithm to find the optimal path using Jupiter notebook.

Algorithm:

Step 1: start

Step 2: Place the starting node into open and find its f(n) [start node] value.

Step 3: Remove the node from OPEN, having the smallest f(n) value, if it is x goal node, then stop and return to success.

Step 4: Else remove the node from OPEN, and find all its successors.

Step 5: Find the f(n) value of all the successors, Place them into OPEN and place the removed node into close **Step 6:** Go to step 2.

Step 7: Exit.

Program :

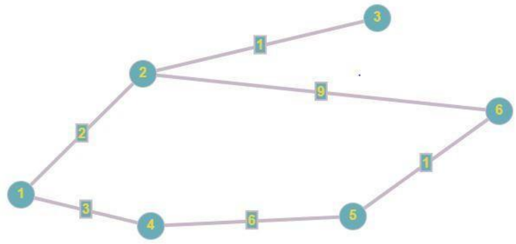
```
# Define graph with neighbors and edge weights
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
    'J': []  # Explicitly add 'J' for completeness
}
# Heuristic function (estimated distance to goal)
def heuristic(n):
    h_dist = {
        'A': 11,
```

```
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return h_dist.get(n, float('inf'))
# Get neighbors of a node
def get_neighbors(v):
    return Graph_nodes.get(v, [])
# A* Algorithm
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}  # Distance from start node
    parents = {}  # To reconstruct path
    g[start_node] = 0
    parents[start_node] = start_node  # Start node is
its own parent
    while len(open_set) > 0:
        n = None
        # Find node with lowest f = g + h
        for v in open_set:
            if n is None or g[v] + heuristic(v) < g[n] +
heuristic(n):
                n = v
        if n == stop_node or n is None or
Graph_nodes.get(n) is None:
            break
        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)
        open_set.remove(n)
        closed_set.add(n)
    # Reconstruct path
    if n == stop_node:
        path = []
        while parents[n] != n:
            path.append(n)
            n = parents[n]
        path.append(start_node)
        path.reverse()
        print('Path found:', path)
        return path
    print('Path does not exist!')
    return None
# Driver code
print("Following is the A* Algorithm:")
aStarAlgo('A', 'J')
```

OUTPUT:

```
==================================================================================== RESTART: C:/Users/mahes/VTU26520.py =
Following is the A* Algorithm:
Path found: ['A', 'F', 'G', 'I', 'J']
>>>
```



Result:

Thus the Implementation of A * Algorithm to find the optimal path using Python Was successfully executed and output was verified.

3(B) – Simplified A* Algorithm.

Aim: To implement the simplified A*Algorithm using Jupiter notebook.

Algorithm:

Step 1 : start.

Step 2: place the starting node into open and find its f(n) value

Step 3: Remove the node from OPEN , having the smallest f(n) value, if it is a goal node , then stop and return to success.

Step 4: else remove the node from OPEN, and find all its successors

Step 5: Find the f(n) value of all the successors, Place them into OPEN and place the removed node into close **Step 6:** Go to step 2.

Step 7: Exit.

Program:

```
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}          # store distance from start node
    parents = {}    # map each node to its parent in
the path
    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0:
```

```
        n = None
        # find node with lowest f() = g + heuristic
        for v in open_set:
            if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n is None:
            print("Path does not exist!")
            return None
        # stop condition
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print("Path found:", path)
            return path
        # process all neighbors
        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)
        open_set.remove(n)
        closed_set.add(n)
    print("Path does not exist!")
```

```
    return None
def get_neighbors(v):
    return Graph_nodes.get(v, [])
def heuristic(n):
    h_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0
    }
    return h_dist.get(n, float('inf'))
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('A', 2), ('C', 1), ('G', 9)],
    'C': [('B', 1)],
    'D': [('E', 6), ('G', 1)],
    'E': [('A', 3), ('D', 6)],
    'G': [('B', 9), ('D', 1)]
}
print("Following is the A* Algorithm:")
aStarAlgo('A', 'G')
```

Output:

```
============================================================================ RESTART: C:/Users/mahes/VTU26520.py
Following is the A* Algorithm:
Path found: ['A', 'E', 'D', 'G']
>>>
```

Result:

Thus the implementation of the simplified A*Algorithm using Jupiter notebook was successfully executed and output was verified.