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 :

```
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}  # store distance from starting node
    parents = {}  # parents contain an adjacency map of all nodes

    # distance of starting node from itself is zero
    g[start_node] = 0
    # start_node is the root node, so it has no parent nodes
    # so start_node is set to its own parent node
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None
        # node with the lowest f() is found
        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[n] is None:
        break
    else:
        for m, weight in get_neighbors(n):
            # nodes 'm' not in open_set and closed_set are added to open_set
            # n is set as its parent
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            # for each node m, compare its distance from start i.e g(m)
            # to the from start through n node
            else:
                if g[m] > g[n] + weight:
                    # update g(m)
                    g[m] = g[n] + weight
                    # change parent of m to n
                    parents[m] = n
                    # if m is in closed_set, remove and add to open_set
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)

    # remove n from the open_set and add it to closed_set
    # because all of its neighbors were inspected
    open_set.remove(n)
    closed_set.add(n)

if n is None:
    print('Path does not exist!')
    return None
```

```
    # if the current node is the stop_node,
    # then we begin reconstructing the path from it to the start_node
    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


# define function to return neighbors and their distances from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None


# for simplicity, we'll consider heuristic distances given
# and this function returns heuristic distance for all nodes
def heuristic(n):
    h_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
```

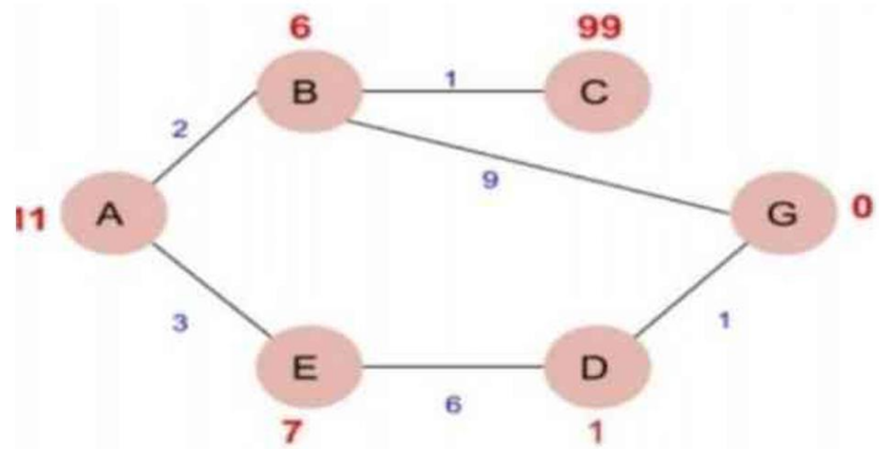
```
        'H': 3,
        'I': 1,
        'J': 0
    }
    return h_dist[n]


# Describe your graph here
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)],
}

print("Following is the A* Algorithm:")
aStarAlgo('A', 'J')
```

Output:

```
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 starting node
    parents = {}  # parents contain an adjacency map of all nodes

    # distance of starting node from itself is zero
    g[start_node] = 0
    # start_node is the root node, so it has no parent nodes
    # so start_node is set to its own parent node
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None
        # node with the lowest f() is found
        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 n not in Graph_nodes:
```

```
            break
        else:
            for m, weight in get_neighbors(n):
                # nodes 'm' not in open_set and closed_set are added to open_set
                # n is set as its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                # for each node m, compare its distance from start i.e g(m)
                # to the from start through n node
                else:
                    if g[m] > g[n] + weight:
                        # update g(m)
                        g[m] = g[n] + weight
                        # change parent of m to n
                        parents[m] = n
                        # if m is in closed_set, remove and add to open_set
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        # remove n from the open_set and add it to closed_set
        # because all of its neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)

    if n is None:
        print('Path does not exist!')
        return None

    # if the current node is the stop_node,
    # then we begin reconstructing the path from it to the start_node
    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

# define function to return neighbors and their distances from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

# for simplicity, we'll consider heuristic distances given
# and this function returns heuristic distance for all nodes
def heuristic(n):
    h_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0
    }
    return h_dist[n]

# Describe your graph here
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:

```
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.