Date: **Ex No:** 5.2	**Title of the Lab** Implementation of A* Algorithm	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement the A* Algorithm.

Description of the Concept or Problem given:

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

Manual Solution:

1. Define a list, OPEN, consisting solely of a single node, the start node, s.
2. IF the list is empty, return failure.
3. Remove from the list the node n with the best score (the node where f is the minimum), and move it to a list, CLOSED.
4. Expand node n.
5. IF any successor to n is the goal node, return success and the solution (by tracing the path from the goal node to s).
6. FOR each successor node: 1.apply the evaluation function, f, to the node. 2. IF the node has not been in either list, add it to OPEN.
7. Looping structure by sending the algorithm back to the second step.

Program Implementation [Coding]:

```
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')
```

Screenshots of the Outputs:

```
Path found: ['A', 'E', 'D', 'G']
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]