NAME: SUFIYAN KHAN DOP:19/07/2023
CLASS: TE AI-DS
ROLL NO: A12

Experiment No. 3

Aim: To implement A star Search Algorithm in Python

Objective: - To find the shortest path between an initial and a final point

Theory: -

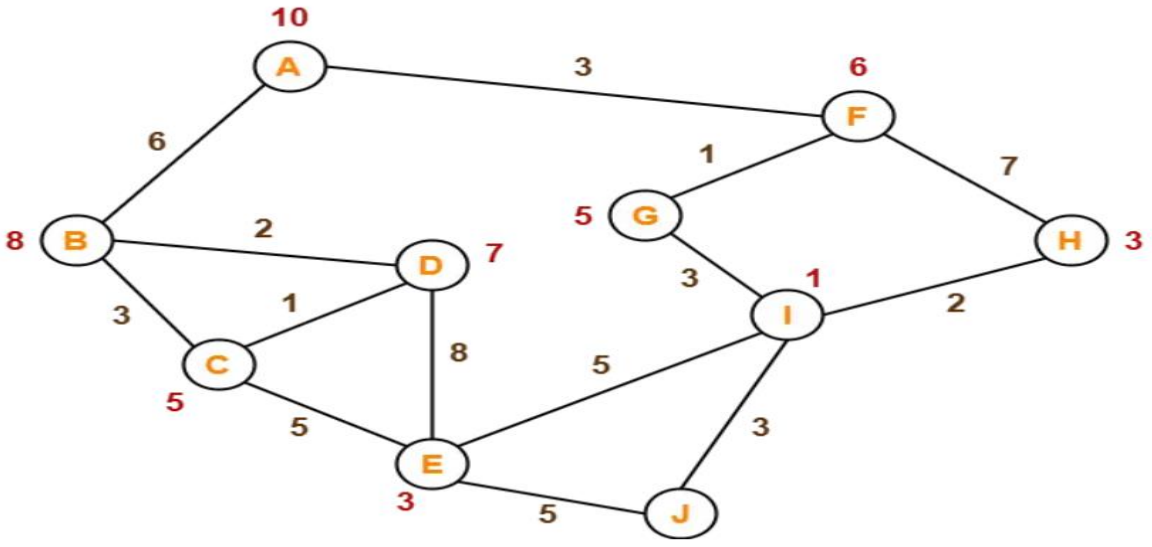
It is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It remains a widely popular algorithm for graph traversal.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance andtime

A major drawback of the algorithm is its space and time complexity. It takes a large amount of space to store all possible paths.



NAME: SUFIYAN KHAN DOP:19/07/2023
CLASS: TE AI-DS
ROLL NO: A12

Numbers written on edges represent the distance between nodes. Numbers written on nodes represent the heuristic value.

Step-01:
- You start at node A.
- Nodes B and F can be reached from node A.
- A* algorithm calculates the f-values for nodes B and F.
  - f(B) = g(B) + h(B) = 6 + 8 = 14
  - f(F) = g(F) + h(F) = 3 + 6 = 9
- Since f(F) < f(B), the algorithm decides to go to node F.
- Path so far: A → F

Step-02:
- From node F, you can reach nodes G and H.
- A* algorithm calculates the f-values for nodes G and H.
  - f(G) = g(G) + h(G) = (3 + 1) + 5 = 9
  - f(H) = g(H) + h(H) = (3 + 7) + 3 = 13
- Since f(G) < f(H), the algorithm decides to go to node G.
- Path so far: A → F → G

Step-03:
- From node G, you can reach node I.
- A* algorithm calculates the f-value for node I.
  - f(I) = g(I) + h(I) = (3 + 1 + 3) + 1 = 8
- It decides to go to node I.
- Path so far: A → F → G → I

Step-04:
- From node I, you can reach nodes E, H, and J.
- A* algorithm calculates the f-values for nodes E, H, and J.
  - f(E) = g(E) + h(E) = (3 + 1 + 3 + 5) + 3 = 15
  - f(H) = g(H) + h(H) = (3 + 1 + 3 + 2) + 3 = 12
  - f(J) = g(J) + h(J) = (3 + 1 + 3 + 3) + 0 = 10
- Since f(J) is the smallest among these options, the algorithm decides to go to node J.
- Path so far: A → F → G → I → J

Program: -

```
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
```

NAME: SUFIYAN KHAN
CLASS: TE AI-DS
ROLL NO: A12

DOP:19/07/2023

```
    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:
```

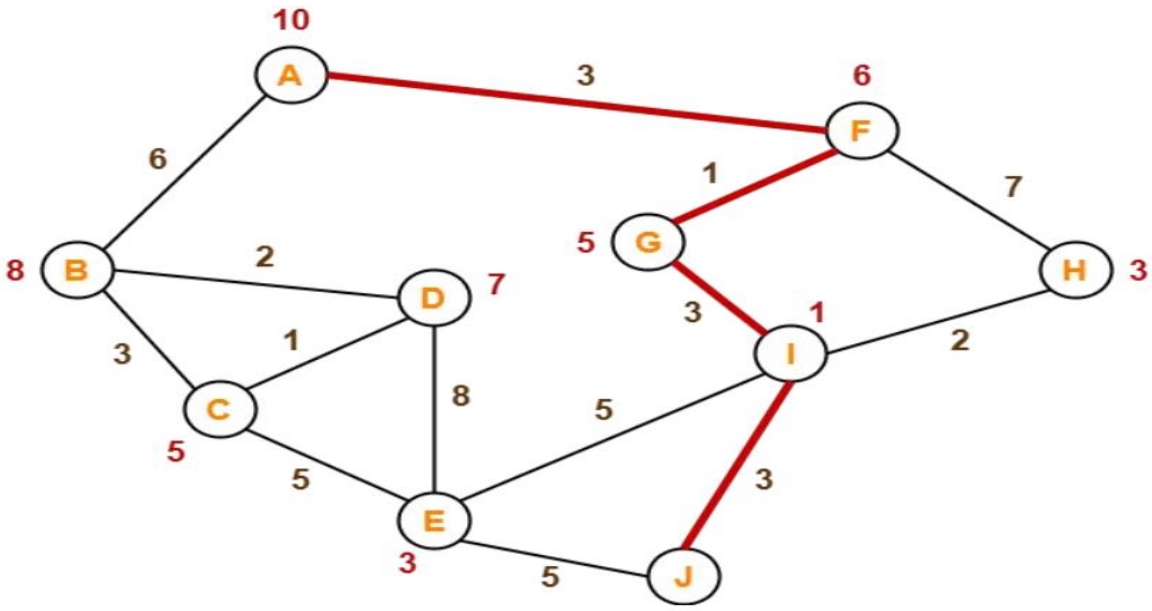
NAME: SUFIYAN KHAN
CLASS: TE AI-DS
ROLL NO: A12

DOP:19/07/2023

```
        return None

def heuristic(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0,
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('B', 2), ('E',8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G',1),  ('H',7)],
    'G': [('I', 3)],
    'H':[('I',2)],
    'I':[('J',3)],
}

# Example usage:
start_node = 'A'
stop_node = 'J'
aStarAlgo(start_node, stop_node)
```

Output:

NAME: SUFIYAN KHAN DOP:19/07/2023
CLASS: TE AI-DS
ROLL NO: A12




```
Path found: ['A', 'F', 'G', 'I', 'J']
```

CONCLUSION:

we've implemented the A* algorithm to find the shortest path between a start and stop node in a graph. The code has been corrected and demonstrated on an example graph. A* is a powerful algorithm used in navigation and AI, offering efficiency and accuracy for pathfinding tasks