1. **Implement A\* Search algorithm.**

**THEORY:**

**A\* Search Algorithm**

1. As we know, Human is a wanderer.He uses many routes to get from one place to another.

2. Since ancient times, humans used to apply a lot of calculation to  choose a optimal path for

destination. Like time, cost, long and short paths.

3. Actually A\* is a  part of informed search techniques(Algorithms) in Artificial

Intelligence.

4. Before moving on Informed search algorithm let&#39;s understand search technique first.

5. In the olden times it was very difficult to find the right path. today, we have algorithms that

can help us find the shortest paths virtually. We just need to add costs (time, money etc.) to

the graphs or maps and the algorithm finds us the path that we need to take to reach our

destination as quick as possible.This is what a search algorithm do.

6. Search Algorithm divided into two parts informed search and uninformed search or

blind.

7. Here we are discussing Informed Search Algorithm.

8. Informed search algorithm contains an array of knowledge such as how far we are from the

destination, path cost, how to reach to destination node, etc.

9. A\* algorithm is a type of  Informed Search Algorithm.

10. A\* mostly known for its completeness, optimality, and optimal efficiency.Meaning it&#39;s sure

that A\* will  find all  route from source  to destination , with least cost.

11. Since It is A\* is best and popular technique used in path-finding and graph traversals. It is

used many computer games and web-map.

How A\* is actually work

     F(n) = G(n) + H(n)

A\*  Algorithm use this formula to provide a optimal path

Here , n-node (in current state)

F- Least cost from source (start node ) to the destination (goal node)

G-Actual cost from start node to n node.

H-Estimated cost from n node to the Goal node (heuristic value of node).

**PROCEDURE / PROGRAMME** :

import os

clear=**lambda** : os.system('cls')

clear()

**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': 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': [('C', 1), ('E', 8)],

    'E': [('I', 5), ('J', 5)],

    'F': [('G', 1),('H', 7)] ,

    'G': [('I', 3)],

    'H': [('I', 2)],

    'I': [('E', 5), ('J', 3)],

}

aStarAlgo('A', 'J')

**OUTPUT:**

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

['A', 'F', 'G', 'I', 'J']