Aim

To find the shortest path from a start node to goal node using A* search.

Algorithm:

step 1 : Start

step 2 - Input graph as adjacency list where each node is connected to the neighbours with given weight.

step 3 - Initialize two sets - Open-set for nodes to be evaluated and closed set for node

step -4 - choose the open set choose the node with lowest f-score (best estimated cost to goal).

step - 5 - If the current node is goal node terminates the search and reconstruct the path

step - 6 - The goal is reached - trace back the nodes to the start node to find the optimal path.

step -7 - Stop

Code :

```
import heapq,
def a_star(graph, start, goal, heuristics):
    open_set = []
    heapq.heappush (open_set, (0, start))
    g_score = {node: float('inf') for node in
        graph}
    g_score = [start] = 0.
    f_score [start] = heuristics [start]
    c = {}.
    while open_set:
        curr = heapq.heappop (open_set)
        if curr == goal:
            return reconstruct_path (c. curr)
        for neighbor, cost in graph [curr]:
            tentative_g_score = g_score [curr] + cost
            if tentative_g_score < g_score [neighbor]:
                came_from [neighbor] = current
                g_score [neighbor] = tentative_g_score
                f_score [neighbor] = tentative_g_score +
                                  heuristic [neighbor]
            heapq.heappush (open_set (f_score [neighbor],
                                  neighbor)).
    return "no path found".

def reconstruct_path (came_from):
    path = [curr]
    while current in came_from:
        curr = came_from [curr]
        path.append (curr)
    path.reverse ().
    return path.

if _name_ == "_main_"
    graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('D', 3), ('E', 1)]
    'C': [('E', 5)]
    'D': [('F', 1)]
```

```
'F' : [].
}

heuristics = {'A', 'B': 4, 'C': 4, 'D': 2, 'E': 1,

start = input ("Enter the start node")
goal = input ("Enter the end node")
path = a_star (graph, start, goal, heuristics)
print ("shortest path:" Path)
```

<u>Output</u>

Enter the start node : A
Enter the End node : F
Shortest Path : ['A', 'B', 'E', 'F']

<u>Result:</u>

Thus the program for A* search problem has been executed successfully.