Expno: 4

Date:

A* Search Problem

Aim:

To implement the A* Search algorithm to find the shortest path from start node to goal node in a graph

Algorithm:

Step 1 : Start

Step 2 : Input Graph as adjacency list where each node is connected to its neighbours with given weight

Step 3 : Initialize two sets : Open set for nodes to be evaluated and Closed-set for nodes

Step 4 : Choose the open-set choose the node with the 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 node to the start node to find the optimal path

Step 7 : Stop

Program:

```
Import heapq
def a_star (graph, start, goal, heuristics):
    open_set = []
    heapq.heappush (open_set, (0, start))
    g_score = { node: float ('int') for node in graph}
    g_score [start] = 0
    f_score = {node: float ('int') for
                                node in graph}
    f_score [start] = heuristics [start]
                    c = {}
    While open_set:
        curr = heapq.heappop (open_set)
    If curr == goal:
        return reconstruct_path (c, curr)
def reconstruct_path (c, curr):
        Path = [curr]
        While curr in c
            curr = c [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)]
```

```
'E': [('F', 2)]

'F': []
}

heuristic = {
'A': 6, 'B': 4, 'C': 4, 'D': 2, 'E': 1, 'F': 0}

start = input ("Enter the start_node:")
goal = input ("Enter the end_node:")
path = a_star (graph, start, goal, heuristics)
print ('shortest path:" Path)
```

Output:

Enter the start_node = A
Enter the end_node = F
Shortest path ['A', 'B', 'E', 'F']

Result:

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