Exp. No: A*

Date:

AIM:

To implement A* search algorithm for finding shortest Path between two nodes in a graph based on cost from start node and heuristic value that estimate function cost to the goal.

ALGORITHM:

Step-1: Start

Step-2: Intialize list (Priority queue) with start node and its f_cost = 0.

Step-3: Create g-cost to track the lowest known cost for start node to each node, starting with 0.

Step-4: Use came_frame to store the parent of each node for path construction

Step-5: Loop until the list is empty

Step-6: Return the reconstructed path if the goal is reached (or) No path exists.

Step-7: Stop.

2207119

CODE:

```
import heapq
def a_star_search(graph, start, goal, heuristic):
    open_list = []
    heapq.heappush(open_list, (0, start))
    g_cost = {start: 0}
    came_from = {start: None}
```

```
while open_list :
    current_f_cost, current_node = heapq.heappop(open_list)

    if current_node == goal:
        return reconstruct_path(came_from, current_node)
for neighbor, cost in graph[current_node]:
    tentative_g_cost = g_cost[current_node] + cost

    if neighbour nor in g_cost or tentative_g_cost < g_cost[neighbour]
        g_cost[neighbour] = tentative_g_cost
        f_cost = tentative_g_cost + heuristic[neighbour]
        heapq.heapush(open_list, (f_cost, neighbor))
        came_from[neighbor] = current_node.

    retun None

def reconstruct_path(came_from, current):
    total_path = [current]
    while current in came_from and came_from[current]
                                    is Not None:
        current = came_from[current]
        total_path.append[current)
    return total_path[::-1]
if __name__ = "__main__":
    graph = {'A' = [('B', 1), ('C', 4)],
             'B' = [('A', 1), ('D', 2), ('E', 5)],
             'C' = [('A', 4), ('F', 3)],
             'D' = [('B', 2)],
             'E' = [('B', 5), ('F', 1)],
             'F' : [('C', 3), ('E', 1)]
            }
```

2207119

```
heuristic = { 'A': 7, 'B': 6, 'C': 2, 'D': 6, 'E': 1,
              'F': 0 }
start_node = 'A'
goal_node = 'F'
path = a_star_search (graph, start_node, goal_node,
                      heuristic)
if path:
    print ("Path found: {' -> '.join(path)}")
else:
    print ("No path found.")
```

Output:

Path found: A → C → F

2207011119

RESULT:

The program has been successfully executed and the output is verified.