EXP NO : 4 - A* SEARCH

AIM :

To implement A* search algorithm for finding shortest path between two nodes.

ALGORITHM :

1) start

2) Initialize List with start node and ~~is~~ its cost = 0.

3) create g-cost to track the Lowest known cost for start node to each node.

4) Store parent of each node.

5) Loop until List is empty.

6) print solution

7) stop.

CODE :

```
import heapq
def a-star-search ( graph, start, goal, heuristic):
        open-List = [ ]
        heapq.heappush (open-List , (0, start))

         came - from = { }
         g-score = {start : 0}

         while open-List :
              current - cost , current-node = heapq.heappop (open-List)
```

```
if ( current-node : : goal ) :
    return  reconstruct-path ( came-from , current-node)

for( neighbour , weight in graph [current-node] :
    tentative-g-score [neighbour] = tentative-g-score.
    came-over [neighbour] = current-node.
    f-score = tentative-g-score + heuristic ( neighbour, goal)
    heapq. heappush (open-list, (f-score, neighbour))
  return none

def  reconstruct-path ( came-from , current-node):
    total-path = [current-node]

    while  current-node in came-from :
        current-node = came-from [current-node]
        total-path. append (current-node)

    return  total-path [::-1]

def  heuristic (node, goal) :
    return 0

graph = {

        'A' = [('B', 1), ('C', 3)],
        'B' = [('A', 1), ('D', 1), ('E', 4)],
        'C' = [('A', 1), ("F', 2) ],
        'D' = [('B', 2)]
        'E' = [ ('B', 4), ('F', 1)]
        'F' = [ ('C', 2) , ('E', 1)]

}
```

220701077

```
start-node = 'A'
goal-node = 'F'
path = a-star-search (graph, start-node, goal-node,
                                  heuristic)

if path:
    print ("shortest path:", path)
else:
    print ("No path found")
```

OUTPUT:

```
shortest path : ['A', 'B', 'D']
```

RESULT:

Thus the program to implement A* search algorithm is successfully executed and output is verified.