EX: 4
Date:

A* Algorithm

Aim:

TO implement A* Algorithm search technique

Algorithm:

* get no of nodes, neighbors & heuristic value
* open list contain start node - set g-score 0 & f-score = g-score + h(n)
* Loop
  - POP node with small f-score
  - If this node is goal then backtrack repeat same.
  - If not then calculate g-score of neighbors
  - the g-score & f-score of neighbors
  - the came-from dictionary to track the path

If goal reached o/p is the path

Code:

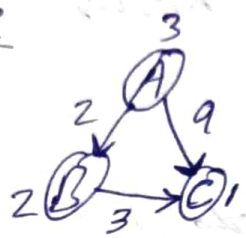
```
import heapq
def a_star(start, goal, graph, heur):
    def heuristic(a, b):
        return heur.get(a, float('inf'))
    def neighbors(node):
        return graph.get(node, [])
```

```
open_list=[]
heapq.heappush(open_list,(0+heuristic(start,
                          goal),0,start))

came_from={}
g_score={start:0}
f_score={start:heur(start,goal)}
while open_list:
    _,current_g,current=heapq.heappop(open_list
    if current==goal:
        Path=[]
        while current in came_from:
            Path.append(current)
            current=came_from[current]
        Path.append(start)
        return Path[::-1]
    for neighbor,cost in neighbor(current):
        tentative_g_score=g_score[current]
                          +cost
        If neighbor not in g_score or tentative
                          _g_score<
                          g_score[neighbor
            came_from[neighbor]=current
            g_score[neighbor]=tentative_g_score
                          +heuristic[neighbor,goal]
    return None
def main():
    graph={}
    heur={}
```

```
n = int (input ("enter no of nodes"))
for _ in range (n):
    node = input ("enter node")
    neighbore_input = input (f"enter neighbor cost")
                                        split()
    neighbors = [ (neighbors_input[i], int (neighbor_
                input [i+1])) for i in range
                (0, len (neighbor_input), 2)]
    graph [node] = neighbors
    heur_val = int (input) (f"enter h [n]")
    heur [node] = heur_val
Start = input ("enter start node")
goal = input ("enter goal node")
if start = goal
    Print ("start & goal are same")
    return
Path = a Star (start, goal, graph, heur)
If Path:
    Print ("Path found", Path)
else:
    Print ("no Path found")
If __name__ == "__main__":
    main()
```

Output:



Enter no of nodes : 3
Enter node : A
Enter neighbor : B 2 C 9
enter n (n) = 3
enter node : B
enter neighbor : C 3
enter n(n) = 2
enter node : C
enter neighbor :
enter n (n) : 1
enter start : A
enter goal node : C

A B C

Result:

Thus A* algorithm is succesfully executed & o/p is verified.