Exp No 4
Date 28/9/24

A* Algorithm

AIM

→ To implement A* Algorithm search technique

Algorithm

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 calculat g_score of neighbor
  - The g_score & f_score of neighbor
  - The came_from dictionary to track the path
* If goal reached o/p is the path

CODE

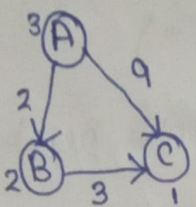
```
import heapq
def a_star(start, goal, graph, heur):
    def heuristic(a, b):
        return heur.get(a, float('inf'))

    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 neighbors(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(neighbo
                                  goal)

    return None

def main():
    graph = {}
```

```
n = int(input("enter no of nodes"))
for _ in range(n):
    node = input("enter node")
    neighbors_input = input(f"enter neighbor cost").
                                                split()
    neighbors = [(neighbors_input[i], int(neighbor_
                input[i+1])) for i in range
                (0, len(neighbors_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()
```

A7

Output



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

A B C

RESULT

Thus A* algorithm is successfully executed & O/P is verified