Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 3

Aim: Implementation of A* Search Algorithm.

Theory:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands fewer search trees and provides optimal results faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In the A* search algorithm, we use the search heuristic as well as the cost to reach the node. Hence, we can combine both costs as follows, and this sum is called a fitness number.

Algorithm of A* search:

1. Place the starting node in the OPEN list.
2. Check if the OPEN list is empty or not, if the list is empty then return failure and stop.
3. Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is the goal node, then return success and stop, otherwise.
4. Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute the evaluation function for n' and place it into the Open list.
5. Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
6. Return to Step 2.

Code:

```
def a_star_algorithm(graph, h, start, stop):
    open_list = set([start])
    closed_list = set([])
    poo = {}
    poo[start] = 0
    par = {}
    par[start] = start
    while len(open_list) > 0:
        n = None
        for v in open_list:
            if n == None or poo[v] + h[v] < poo[n] + h[n]:
                n = v
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop:
            reconst_path = []
            while par[n] != n:
                reconst_path.append(n)
                n = par[n]
            reconst_path.append(start)
            reconst_path.reverse()
            return reconst_path, poo[stop]
        for (m, weight) in graph[n]:
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                par[m] = n
                poo[m] = poo[n] + weight
            else:
                if poo[m] > poo[n] + weight:
                    poo[m] = poo[n] + weight
                    par[m] = n
            if m in closed_list:
                closed_list.remove(m)
                open_list.add(m)
        open_list.remove(n)
        closed_list.add(n)
    print('Path does not exist!')
    return None

nodes = int(input('Enter the number of nodes: '))
graph = {}
h = []
```

```
for i in range(0, nodes):
    node_name = input(f'Enter node {i + 1} name: ')
    h[node_name] = int(input('Enter its huristic value: '))
    connected = int(input(f'Enter number of nodes connected to node {node_name}: '))
    connected_nodes = []
    for j in range(0, connected):
        sub_node = input('Enter the node: ')
        distance = int(input('Enter it\'s distance: '))
        connected_nodes.append((sub_node, distance))
        graph[node_name] = connected_nodes

source_node = input('Enter the source node: ')
goal_node = input('Enter the goal node: ')
path, cost = a_star_algorithm(graph, h, source_node, goal_node)
print('The path is: ', end='')
for i in path:
    print(i, end='')
    if i != path[-1]:
        print('->', end='')
    else:
        print()

print(f'The final path cost is: {cost}')
```

Output:

```
PS C:\Users\HP\VSC> python -u "c:\Users\HP\VSC\Artificial Intelligence\A_Star.py"
Enter the number of nodes: 10
Enter node 1 name: A
Enter its huristic value: 10
Enter number of nodes connected to node A: 2
Enter the node: B
Enter it's distance: 6
Enter the node: F
Enter it's distance: 3
Enter node 2 name: B
Enter its huristic value: 8
Enter number of nodes connected to node B: 2
Enter the node: C
Enter it's distance: 3
Enter the node: D
Enter it's distance: 2
Enter node 3 name: C
Enter its huristic value: 5
Enter number of nodes connected to node C: 2
Enter the node: D
Enter it's distance: 1
Enter the node: E
Enter it's distance: 5
Enter node 4 name: D
Enter its huristic value: 7
Enter number of nodes connected to node D: 1
Enter the node: E
Enter it's distance: 8
Enter node 5 name: E
Enter its huristic value: 3
Enter number of nodes connected to node E: 2
Enter the node: I
Enter it's distance: 5
Enter the node: J
Enter it's distance: 5
Enter node 6 name: F
Enter its huristic value: 6
Enter number of nodes connected to node F: 2
Enter the node: G
Enter it's distance: 1
Enter the node: H
Enter it's distance: 7
Enter node 7 name: G
Enter its huristic value: 5
Enter number of nodes connected to node G: 1
Enter the node: I
Enter it's distance: 3
Enter node 8 name: H
Enter its huristic value: 3
Enter number of nodes connected to node H: 1
Enter the node: I
Enter it's distance: 2
Enter node 9 name: I
Enter its huristic value: 1
Enter number of nodes connected to node I: 1
Enter the node: J
Enter it's distance: 3
Enter node 10 name: J
Enter its huristic value: 6
Enter number of nodes connected to node J: 0
Enter the source node: A
Enter the goal node: H
The path is: A->F->H
The final path cost is: 10
```

Conclusion:

Learnt about A* Search Algorithm along with its steps and implemented it in code.