Experiment 5

Aim: Implementation of A* search for problem solving.

Objective: To study the informed searching techniques and its implementation for problemsolving.

Theory:
Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.
Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.
Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.
1. f(n)= g(n).
Were, h(n)= estimated cost from node n to the goal.
The greedy best first algorithm is implemented by the priority queue.

A* search algorithm:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 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 evaluation function for n' and place into Open list.

Step 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.

Step 6: Return to **Step 2**.

Advantages:

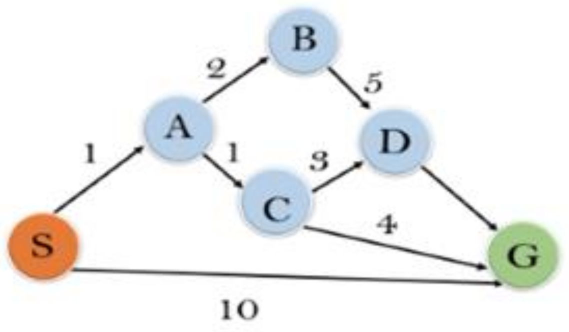
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

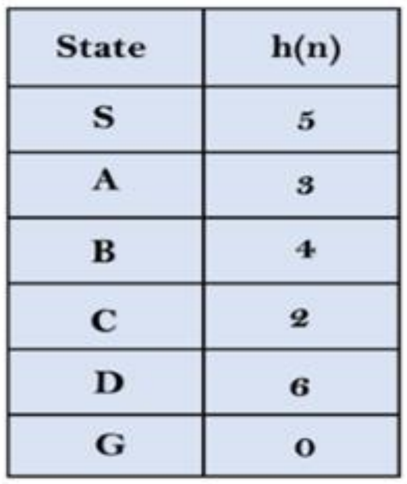
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

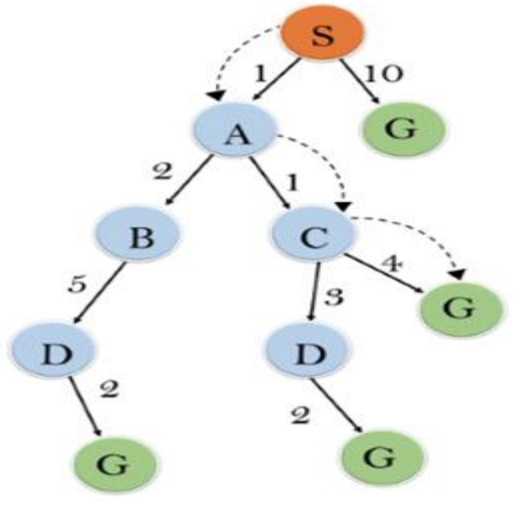
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.





State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Code:
```
import heapq
dict_hn = {'Arad': 336, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242, 'Eforie': 161,
        'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,
        'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu': 193,
        'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374}
dict_gn = dict(
    Arad=dict(Zerind=75, Timisoara=118, Sibiu=140),
    Bucharest=dict(Urziceni=85, Giurgiu=90, Pitesti=101, Fagaras=211),
    Craiova=dict(Drobeta=120, Pitesti=138, Rimnicu=146),
    Drobeta=dict(Mehadia=75, Craiova=120),
    Eforie=dict(Hirsova=86),
    Fagaras=dict(Sibiu=99, Bucharest=211),
    Giurgiu=dict(Bucharest=90),
    Hirsova=dict(Eforie=86, Urziceni=98),
    Iasi=dict(Neamt=87, Vaslui=92),
    Lugoj=dict(Mehadia=70, Timisoara=111),
    Mehadia=dict(Lugoj=70, Drobeta=75),
    Neamt=dict(Iasi=87),
    Oradea=dict(Zerind=71, Sibiu=151),
    Pitesti=dict(Rimnicu=97, Bucharest=101, Craiova=138),
    Rimnicu=dict(Sibiu=80, Pitesti=97, Craiova=146),
    Sibiu=dict(Rimnicu=80, Fagaras=99, Arad=140, Oradea=151),
    Timisoara=dict(Lugoj=111, Arad=118),
    Urziceni=dict(Bucharest=85, Hirsova=98, Vaslui=142),
    Vaslui=dict(Iasi=92, Urziceni=142),
    Zerind=dict(Oradea=71, Arad=75)
)
def a_star(start, goal, dict_hn, dict_gn):
    open_list = [(0, start)]
    g_values = {node: float('inf') for node in dict_hn}
    g_values[start] = 0
    parent_nodes = {}
    while open_list:
        f, current = heapq.heappop(open_list)
        if current == goal:
            return reconstruct_path(parent_nodes, current)
        for neighbor, cost in dict_gn[current].items():
            tentative_g = g_values[current] + cost
            if tentative_g < g_values[neighbor]:
                g_values[neighbor] = tentative_g
                f_value = tentative_g + dict_hn[neighbor]
                heapq.heappush(open_list, (f_value, neighbor))
                parent_nodes[neighbor] = current
    return None
def reconstruct_path(parents, current):
    path = [current]
    while current in parents:
        current = parents[current]
        path.append(current)
    path.reverse()
    return path
start_node = 'Arad'
goal_node = 'Bucharest'
path = a_star(start_node, goal_node, dict_hn, dict_gn)
if path:
    print("Path found:", path)
else:
    print("Path not found.")
```

Output:

```
PS D:\Vartak college\SEM 5\AI EXP\New folder> py .\astar.py
Path found: ['Arad', 'Sibiu', 'Rimnicu', 'Pitesti', 'Bucharest']
```

Conclusion:

Thus, we have learned to implement of A* search for problem solving. Informed search algorithms use heuristic knowledge to focus the exploration and find the goal node more efficiently. The heuristic function estimates the cost to reach the goal from a given node, guiding the search to expand the most promising nodes first.