4.2. Experiment No. 2

Aim:

Implement A* star Algorithm for any game search problem.

Objective:

In this experiment, we will be able to do the following:

- To understand informed Search Strategies.
- To make use of Graph and Tree Data Structure for implementation of informed Search strategies.
- Study the A* algorithms for various graph traversal problem.

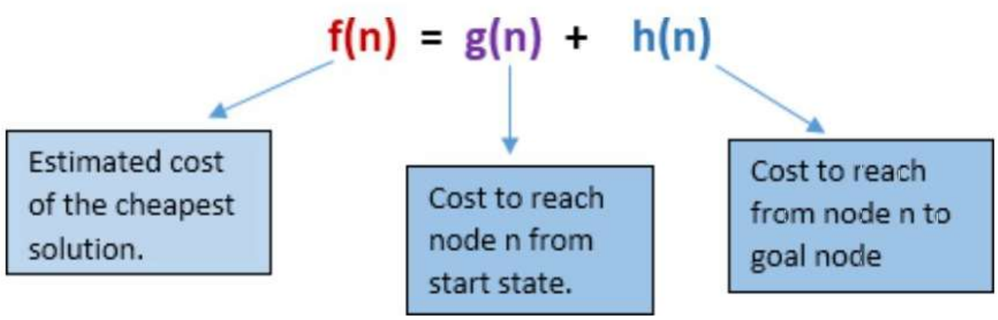
Theory:

A Search*

A* search is the most commonly known form of best-first search. Ituses 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 solve the problem efficiently.

A* search algorithm finds the shortest path through the search space using theheuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

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



Algorithm of A* search:

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

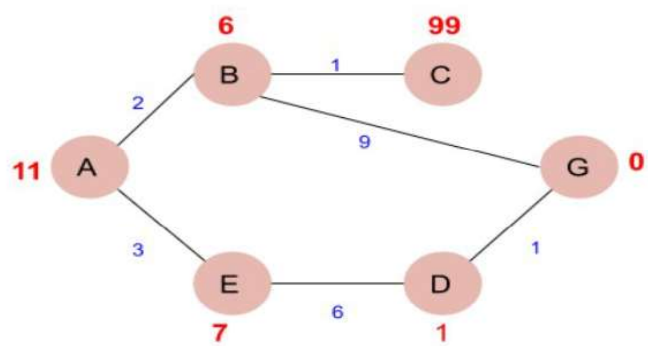
Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

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.

Implementation with Python

In this section, we are going to find out how the A* search algorithm can be used to find the most cost-effective path in a graph. Consider the following graph below.



The numbers written on edges represent the distance between the nodes, while the numbers written on nodes represent the heuristic values. Let us find the most cost-effective path to reach from start state A to final state G using the A* Algorithm.

Let's start with node A. Since A is a starting node, therefore, the value of g(x) for A is zero, and from the graph, we get the heuristic value of A is 11, therefore

$g(x) + h(x) = f(x)$

0+ 11 =11

Thus for A, we can write A=11

Now from A, we can go to point B or point E, so we compute f(x) for each of them

$A \rightarrow B = 2 + 6 = 8$

$A \rightarrow E = 3 + 6 = 9$

Since the cost for $A \rightarrow B$ is less, we move forward with this path and compute the f(x) for the children nodes of B

Since there is no path between C and G, the heuristic cost is set to infinity or a very high value

$A \rightarrow B \rightarrow C = (2 + 1) + 99 = 102$

$A \rightarrow B \rightarrow G = (2 + 9) + 0 = 11$

Here the path $A \rightarrow B \rightarrow G$ has the least cost but it is still more than the cost of $A \rightarrow E$, thus we explore this path further

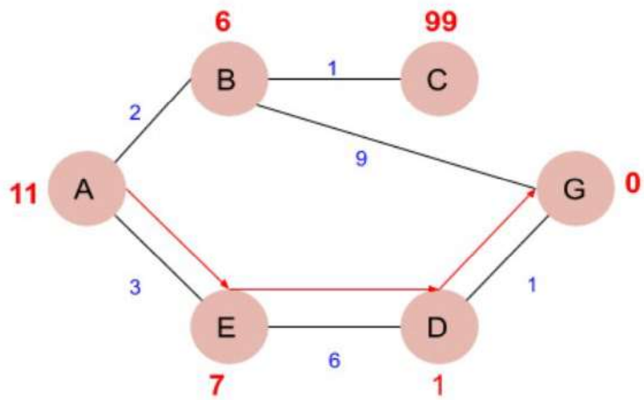
$A \rightarrow E \rightarrow D = (3 + 6) + 1 = 10$

Comparing the cost of $A \rightarrow E \rightarrow D$ with all the paths we got so far and as this cost is least of all we move forward with this path. And compute the f(x) for the children of D

$A \rightarrow E \rightarrow D \rightarrow G = (3 + 6 + 1) + 0 = 10$

Now comparing all the paths that lead us to the goal, we conclude that $A \rightarrow E \rightarrow D$

$\rightarrow G$ is the most cost-effective path to get from A to G.



Input:
```
# heuristic distance for all nodes
def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

}
```

Output:

Path Found: ['A','E','D','G']

Advantages:

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

Disadvantages:

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

Applications:

A- Star algorithm is used to determine the closest path the enemy can go to the tree.

Furthermore, the player must try to protect the tree from the enemy by touching the enemy or using a barrier to keep the tree alive.

Conclusion:

Successfully able to implement graph traversal problem using A* Algorithm.

Questions:

1. How is the heuristic function used in the A* algorithm, and what is its significance?

2. What is the role of a priority queue in the A* algorithm?

3. How does A* guarantee finding the optimal solution, and under what conditions does it hold true?

4. What is an admissible heuristic, and why is it important in the context of the A* algorithm?

5. Explain the similarities and differences between A* and the Best-First Search algorithm.