CS 3011: Artificial Intelligence

Solving Problems by Searching

Instructors: Dr. Durgesh Singh

CSE Discipline, PDPM IIITDM, Jabalpur -482005

Iterative-deepening A* search (IDA*)

- IDA* gives us the benefits of A* without the requirement to keep all reached states in memory, at a cost of visiting some states multiple times.
- It is commonly used algorithm for problems that do not fit in memory.
- In IDA* the cutoff is the f-cost (g + h);
  - at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

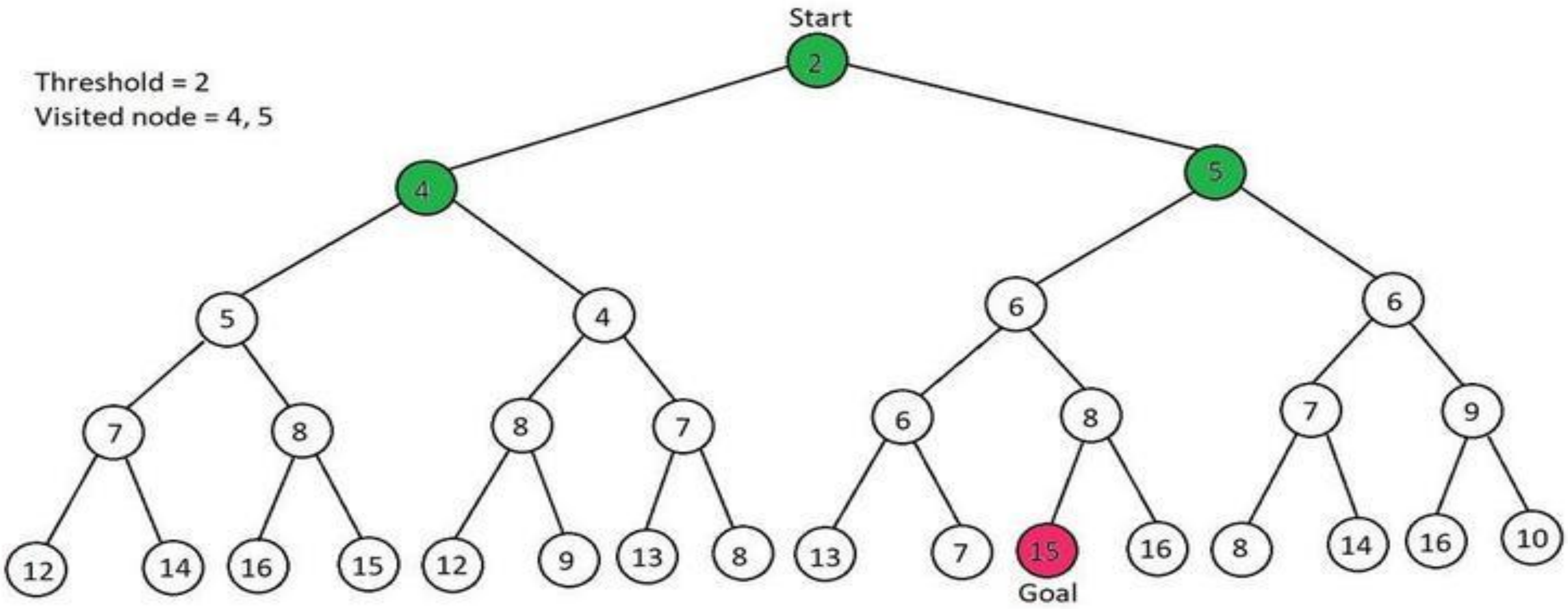
How IDA* Work?

- At each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n)=g(n)+h(n)$ exceeds a given threshold.
- This threshold starts at the estimate of the cost at the initial state, f(root) and increases for each iteration of the algorithm.
  - The threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold

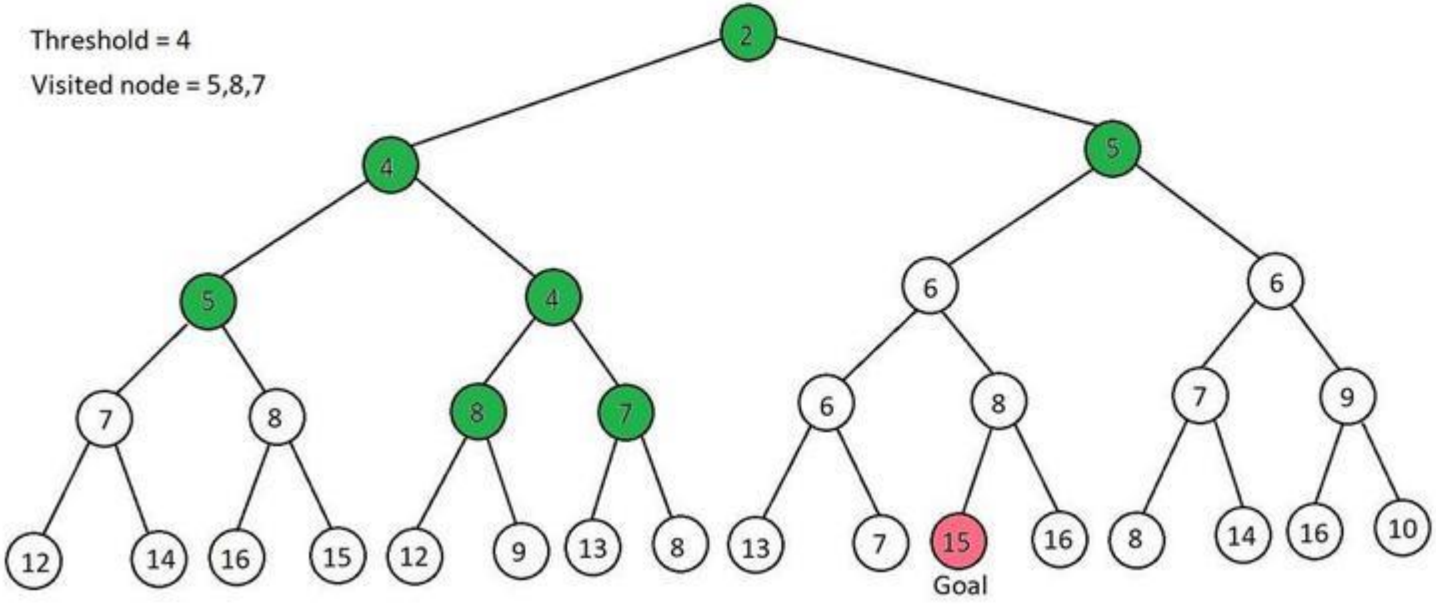
Step-by-Step Process of the IDA* Algorithm

1. **Initialization**: Set the root node as the current node and compute its f-score.
2. **Set Threshold**: Initialize a threshold based on the f-score of the starting node.
3. **Node Expansion**: Expand the current node’s children and calculate their f-scores.
4. **Pruning**: If the f-score exceeds the threshold, prune the node and store it for future exploration.
5. **Path Return**: Once the goal node is found, return the path from the start node to the goal.
6. **Update Threshold**: If the goal is not found, increase the threshold based on the minimum pruned value and repeat the process.

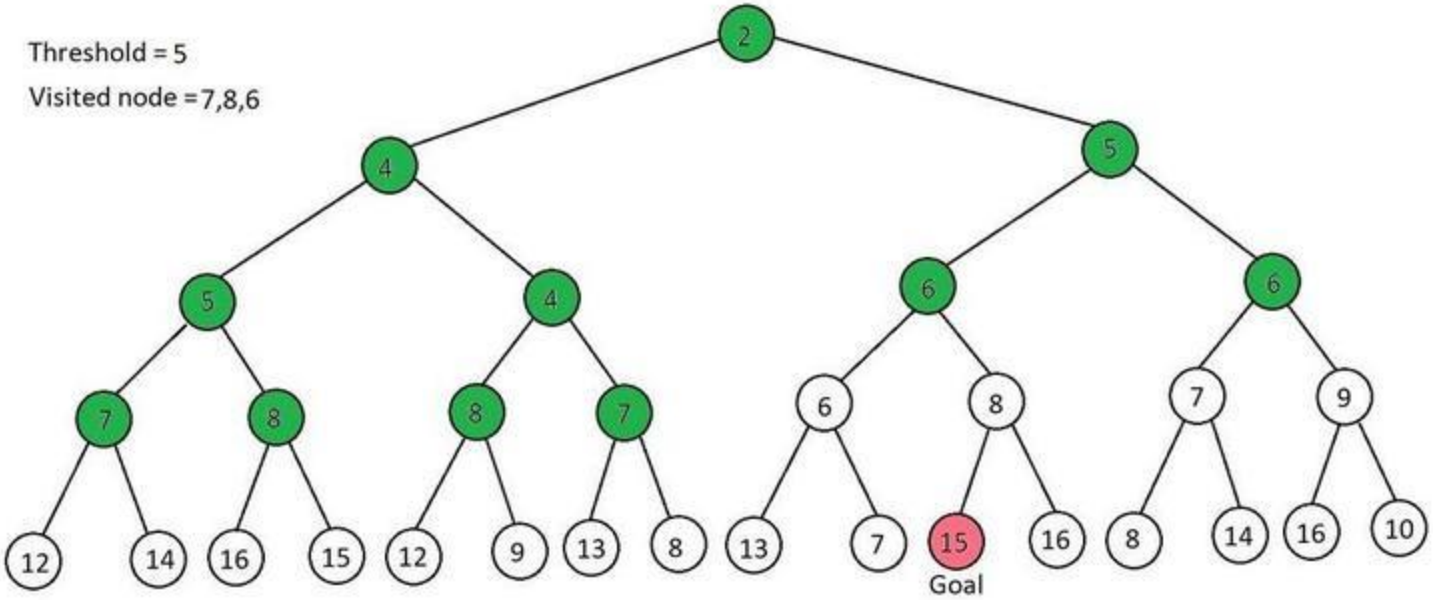
Example of IDA*: In the below tree, the **f score** is written inside the nodes



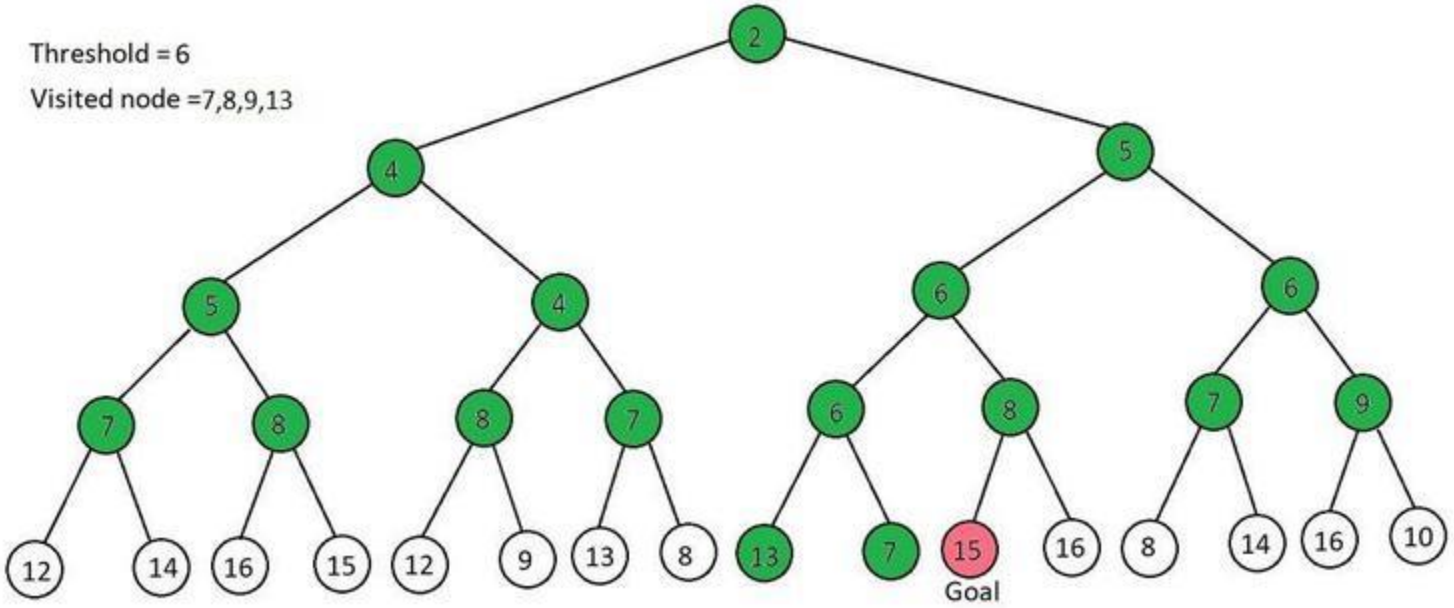
Example of IDA*: Iteration2



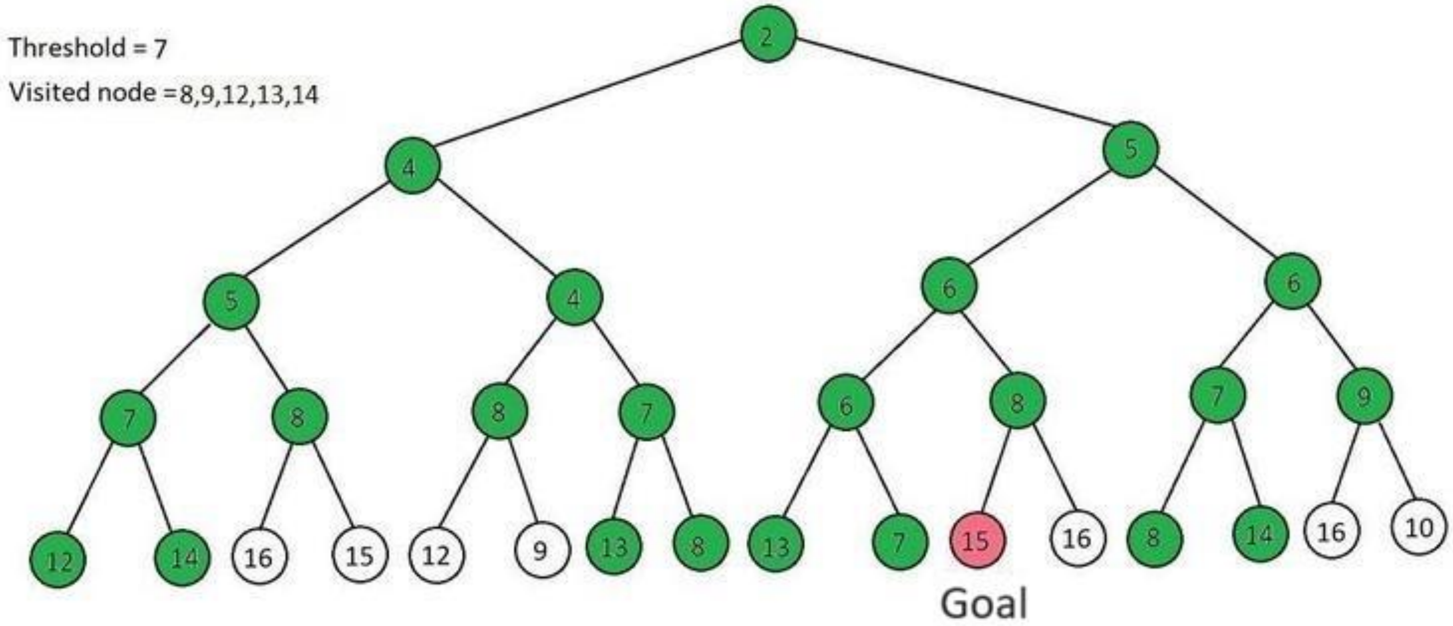
Example of IDA*: Iteration3



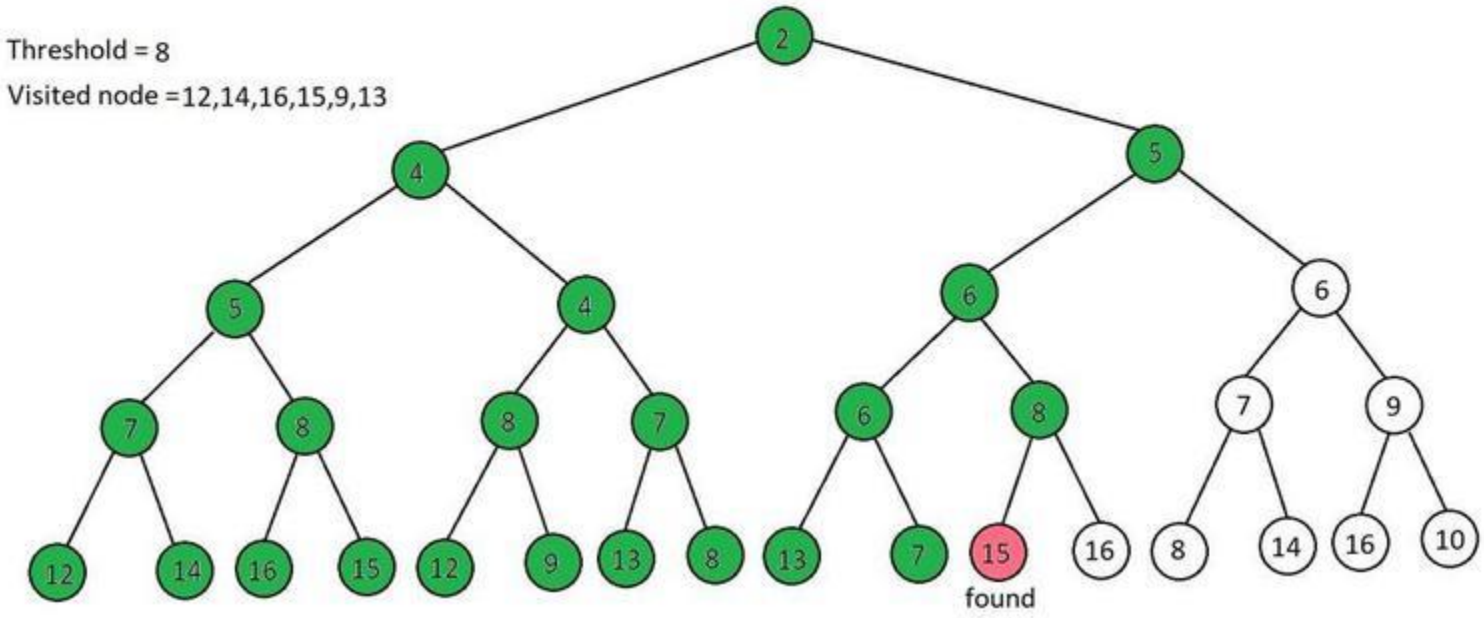
Example of IDA*: Iteration4



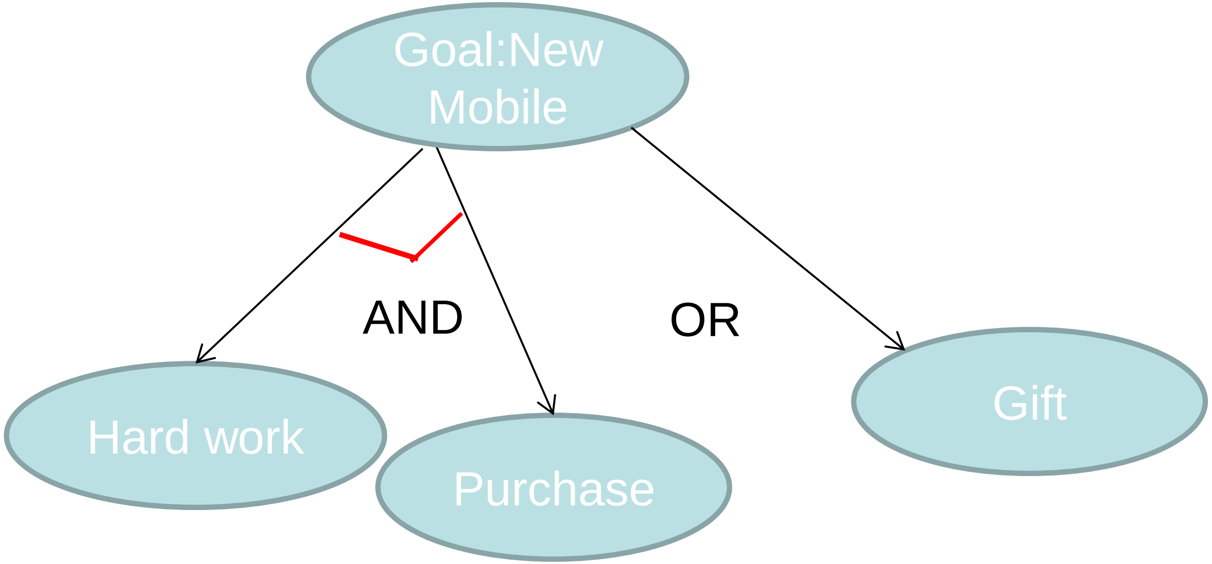
Example of IDA*: Iteration5



Example of IDA*: Iteration6



Example AND-OR Graph



AO* (AND –OR) Search Algorithm

- AO* algorithm uses the concept of AND-OR graphs to decompose any given complex problem into smaller set of sub problems which are further solved.
- AND-OR graphs are specialized graphs that are used in problems that can be broken down into sub problems
  - where AND side of the graph represent a set of task that need to be done together to achieve the goal
  - whereas the OR side of the graph represent the different ways of performing task to achieve the same main goal.

Working of AO* algorithm

- The AO* algorithm works on the formula given below :

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

where,

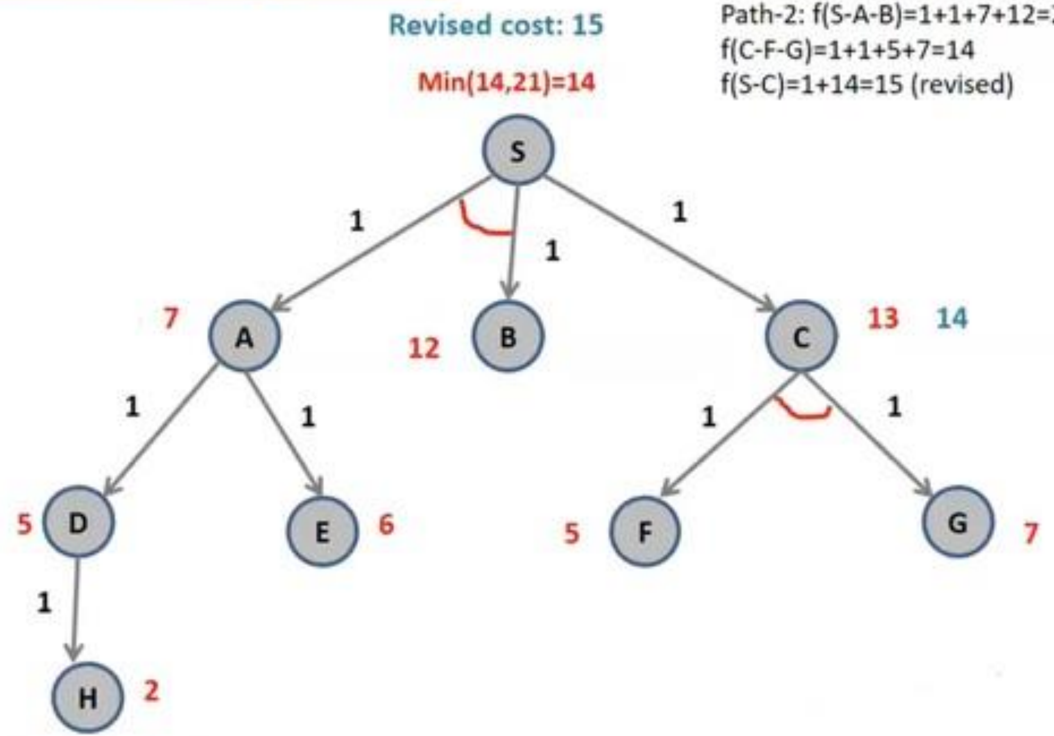
- g(n): The actual cost of traversal from initial state to the current state.
- h(n): The estimated cost of traversal from the current state to the goal state.
- f(n): The estimated total cost of traversal from the initial state to the goal state.

Difference Between A* and AO* Algorithm

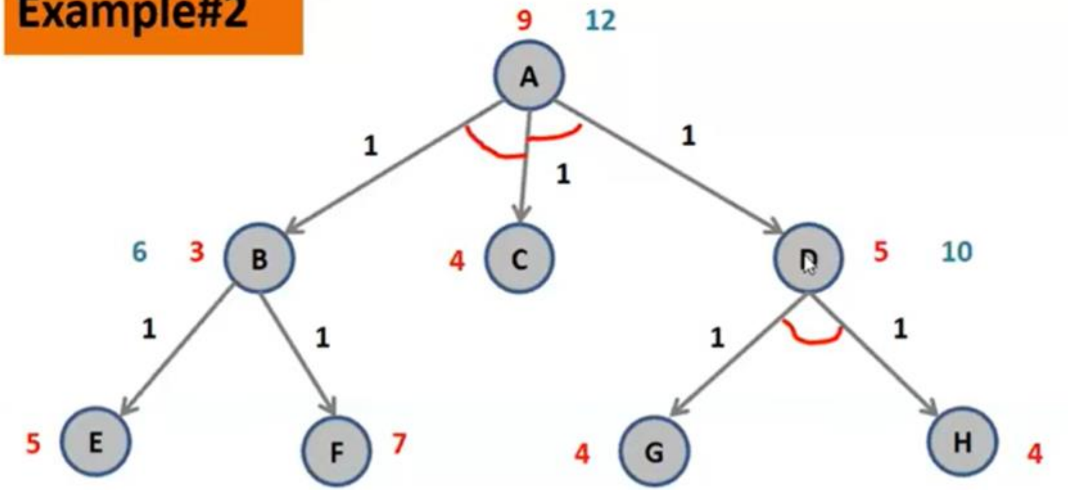
- An A* algorithm is an OR graph algorithm while the AO* algorithm is an AND-OR graph algorithm.
- A* algorithm guarantees to give an optimal solution while AO* doesn’t since AO* doesn’t explore all other solutions once it got a solution.

AO*: Example #1

f(n)=g(n)+h(n)
Path-1: f(S-C)= 1+13=14
Path-2: f(S-A-B)=1+1+7+12=21
f(C-F-G)=1+1+5+7=14
f(S-C)=1+14=15 (revised)



Example#2



Path -1: f(A-B-C)= 1+1+3+4= 9

f(B-E)=1+5=6 f(B-F)=1+7=8

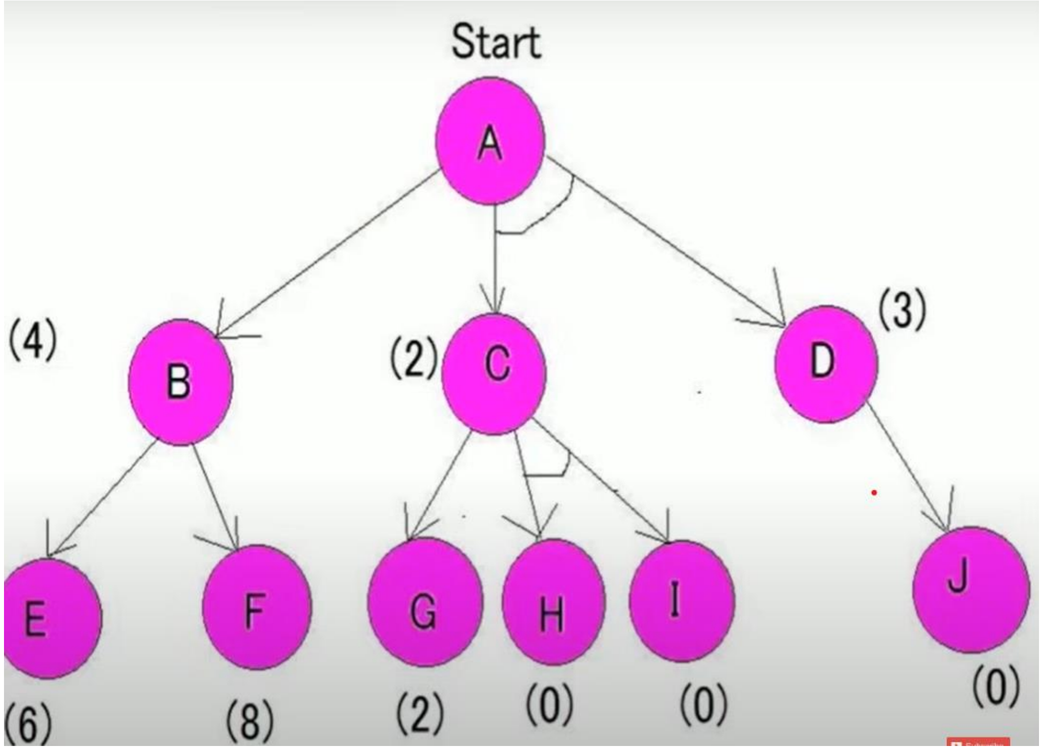
f(A-B-C)= 1+1+6+4= 12

Path-2: f(A-C-D)=1+1+4+5=11

f(D-G-H)= 1+1+4+4 =10

f(A-C-D)= 1+1+4+10= 16

Example: Edge cost=1, h(n) is given



AO* Performance

- **The AO* algorithm is not optimal** because it stops as soon as it finds a solution and does not explore all the paths.
- **AO* is complete**, meaning it finds a solution, if there is any.
- **Time complexity** is O(b^m)
- The AND feature in this algorithm reduces the demand for memory. The space complexity comes in polynomial order