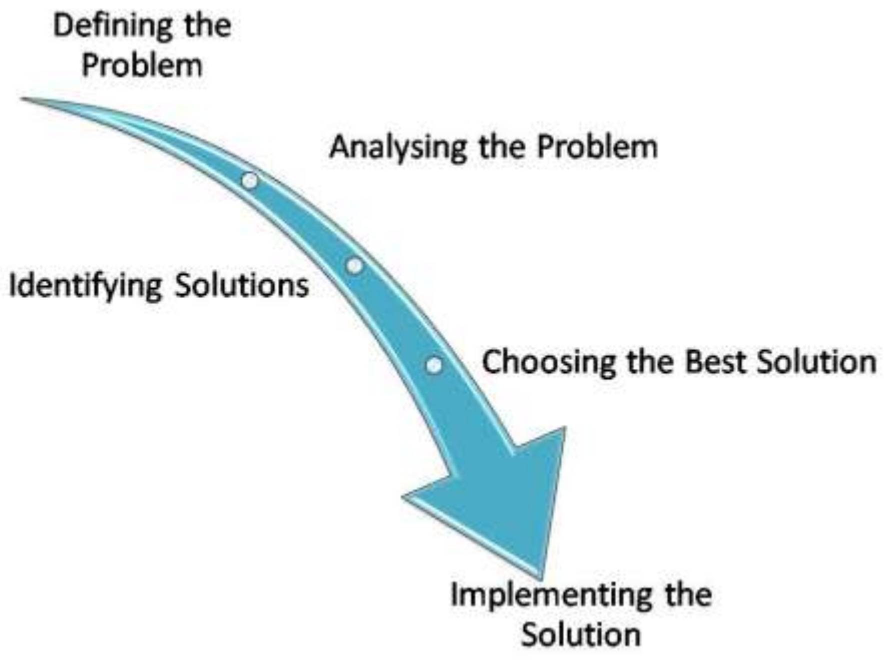
Chapter-03: Solving problems by searching.(For mid)

❖ **The process of solving problem:**



❖ **Properties of searching algorithm:**

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** Does the strategy find the optimal solution?
- **Time complexity:** How long does it take to find a solution?
- **Space complexity:** How much memory is needed to perform the search?

❖ **Types of searching algorithm:**

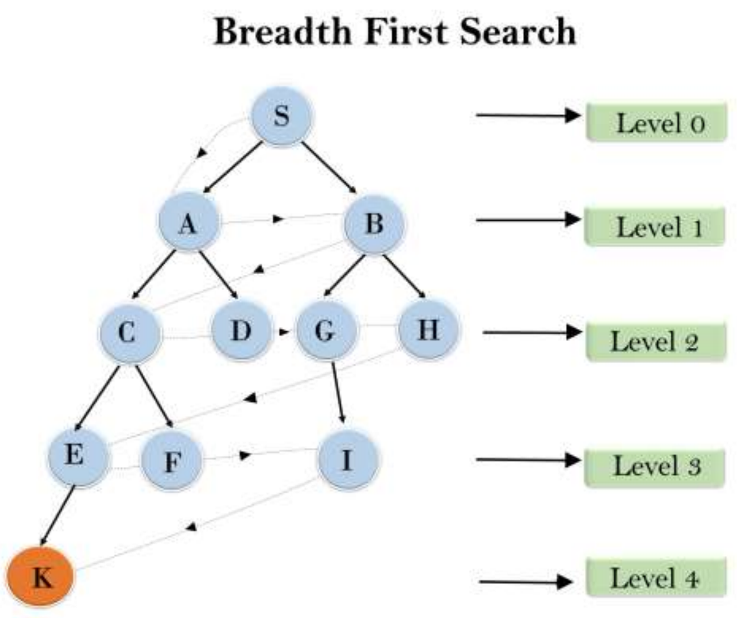
1. Uninformed search(Blind search)
2. Informed search(Heuristic search)

✪ ***Uninformed search:*** Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

1. Breadth-First-Search(BFS).
2. Depth-First-Search(DFS).
3. Uniform cost search(UCS).
4. Depth limited search(DLS).
5. Iterative Deepening DFS(IDDFS)
6. Bi-directional search algorithm.

Breadth-First-Search(BFS): It visits all the neighbors of a vertex before moving to the next level neighbors.

For example, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K.



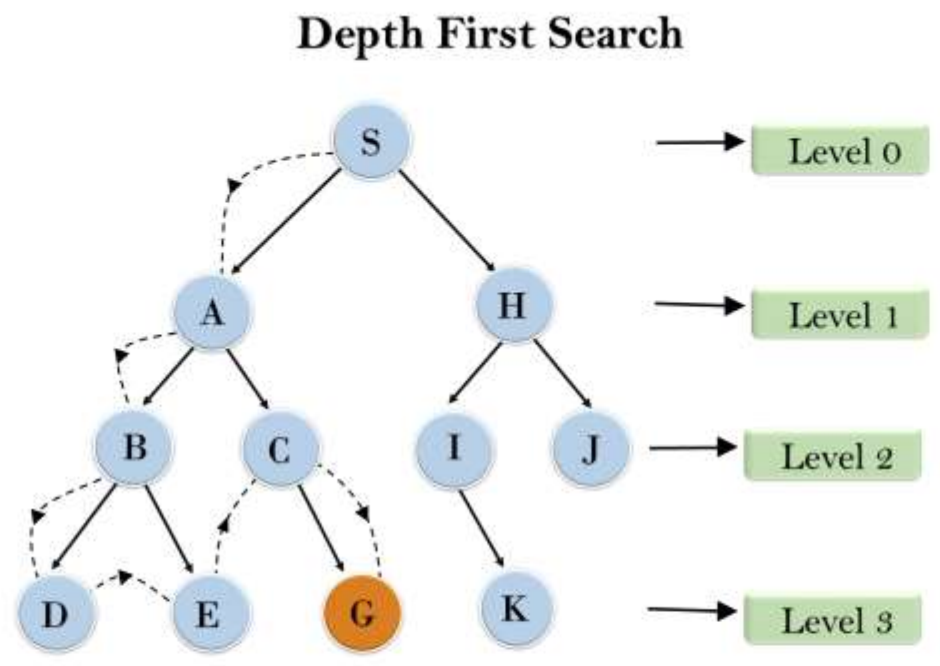
BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

- **Advantages of BFS:**
  - Simplest search strategies.
  - Complete. It there is a solution, BFS is guaranteed to find it.
  - If there are multiple solutions, then a minimal solution will be found.
  - Finds the path of minimal length to the goal.
- **Disadvantages of BFS:**
  - It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
  - BFS needs lots of time if the solution is far away from the root node.
- **Properties of BFS:**
  - **Completeness:** Yes, BFS is complete if the branching factor is finite.
  - **Optimality:** Yes, BFS is optimal if the cost is a non-decreasing function of the depth.
  - **Time complexity:** $O(b^d)$ where b is the branching factor and d is the depth of the shallowest solution.
  - **Space complexity:** $O(b^d)$ for frontier nodes.

- **For better understanding, please Click here and watch the video(up to 12:10 second).**

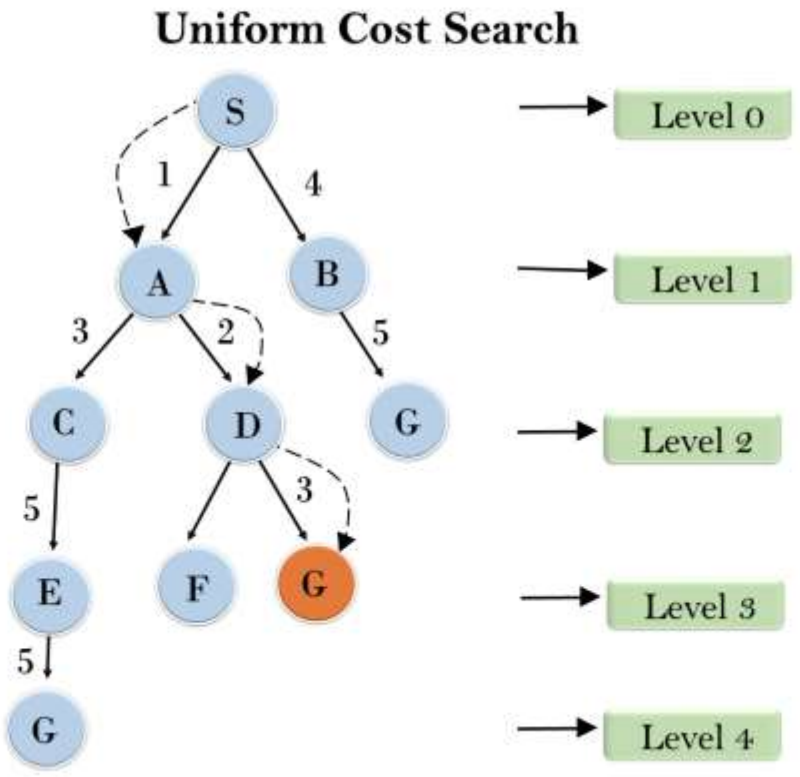
- **Depth-First-Search(DFS):** It's a graph traversal algorithm that explores as far as possible along each branch before backtracking.(LIFO)
For example, In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
Root node--->Left node ----> right node.



It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.
S---> A--->B----> D----> E----> C--->G

- **Advantage:**
  - DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
  - It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).
- **Disadvantage:**
  - There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
  - DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

- **Properties of DFS:**
  - **Completeness:** DFS is not complete if the search space has infinite depth or if there are loops in the graph, as it can get stuck in an infinite loop.
  - **Optimality:** DFS does not guarantee optimality; it may find a non-optimal solution if the depth of the goal node is not the shallowest in the search tree.
  - **Time complexity:** O(b^m), where b is the branching factor and m is the maximum depth of the search tree. In the worst case, DFS may explore all nodes.
  - **Space complexity:** The space complexity of Depth First Search (DFS) is O(bm) in the worst case, where b is the branching factor, and m is the maximum depth of the search tree.
- **For better understanding, please Click here and watch the video(From 12:11 second).**
- **Uniform cost search(UCS):** UCS explores nodes with the least cost first.

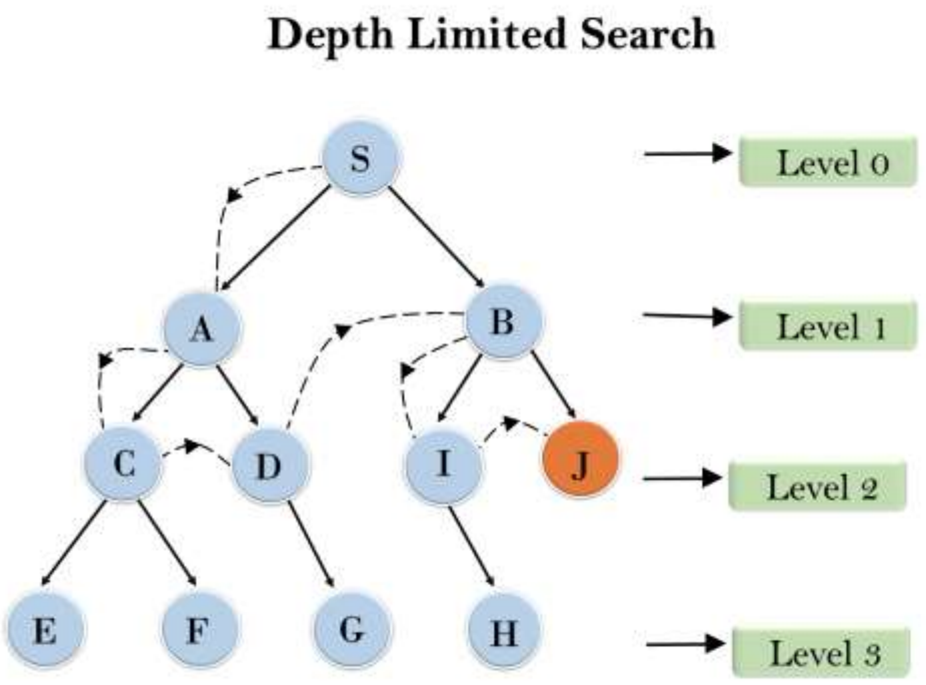


- **Properties of UCS:**
  - **Completeness:** Yes, UCS is complete if the step cost is greater than some positive constant.
  - **Optimality:** Yes, UCS is optimal if the step cost is a non-decreasing function of the path length.
  - **Time complexity:** O($b^{1+(C^*/\varepsilon)}$), where b is the branching factor, C is the cost of the optimal solution, and ε is the minimum step cost.

- **Space complexity:** $O(b^{1+(C^*/\varepsilon)})$, where b is the branching factor and d is the depth of the shallowest goal node.

- **What's the limitations of DFS and how can you address them?**
  DFS has a limitation called stack overflow, especially when dealing with deep or infinite graphs, as it uses recursion or an explicit stack to keep track of vertices. To address this, an iterative version of DFS can be implemented using an explicit stack to avoid recursion.

Depth limited search(DLS): DLS is a modification of DFS that limits the depth of exploration.

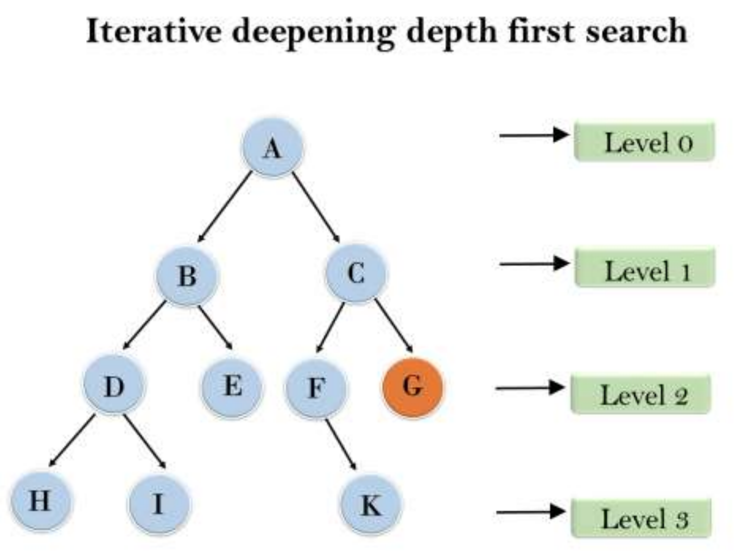


- **Properties of DLS:**
  - **Completeness:** Yes, DLS is complete if the solution is within the depth limit.
  - **Optimality:** DLS is not necessarily optimal; it finds the first solution within the depth limit, which might not be the best solution.
  - **Time complexity:** $O(b^l)$, where b is the branching factor and l is the depth limit. DLS explores up to a limited depth.
  - **Space complexity:** $O(b \times l)$, where b is the branching factor and l is the depth limit. DLS stores nodes on the current branch up to the depth limit.

- **For better understanding, please Click here and watch the video.**

- **Iterative Deepening DFS(IDDFS):** IDDFS combines depth-first search's exploration strategy with iterative deepening by gradually increasing the depth limit until a solution is found.

  Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



  - ✓ 1'st Iteration-----> A
  - ✓ 2'nd Iteration----> A, B, C
  - ✓ 3'rd Iteration------>A, B, D, E, C, F, G
  - ✓ 4'th Iteration------>A, B, D, H, I, E, C, F, K, G
  - ✓ In the fourth iteration, the algorithm will find the goal node.

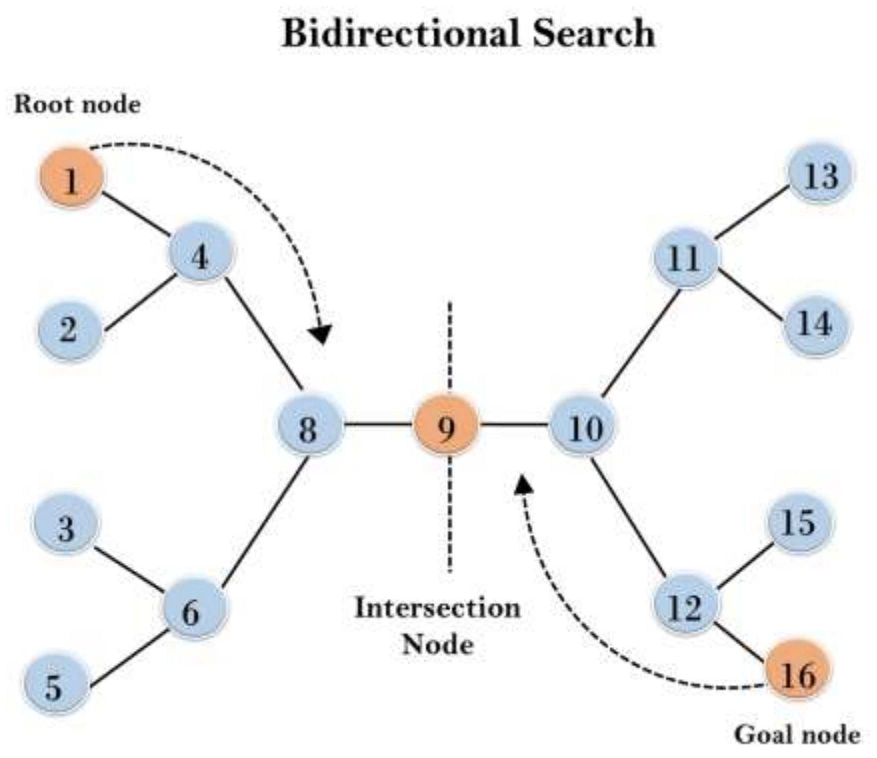
- **Properties of IDDFS:**
  - **Completeness:** IDDFS is complete as it will eventually find a solution if one exists, exploring deeper levels in each iteration.
  - **Optimality:** IDDFS is optimal when the path cost is a non-decreasing function of the depth because it explores shallower levels first.
  - **Time complexity:** O(b^d), where b is the branching factor and d is the depth of the shallowest goal node.
  - **Space complexity:** O(bd), where b is the branching factor and d is the maximum depth reached in any iteration.

- **For better understanding, please Click here and watch the video.**

Bidirectional Search: Bidirectional Search explores from both the start and goal nodes simultaneously, meeting in the middle.

For example, In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



- **Completeness:** Bidirectional Search is complete if we use BFS in both searches.
- **Time Complexity:** Time complexity of bidirectional search using BFS is O(b^d).
- **Space Complexity:** Space complexity of bidirectional search is O(b^d).
- **Optimality:** Bidirectional search is Optimal.

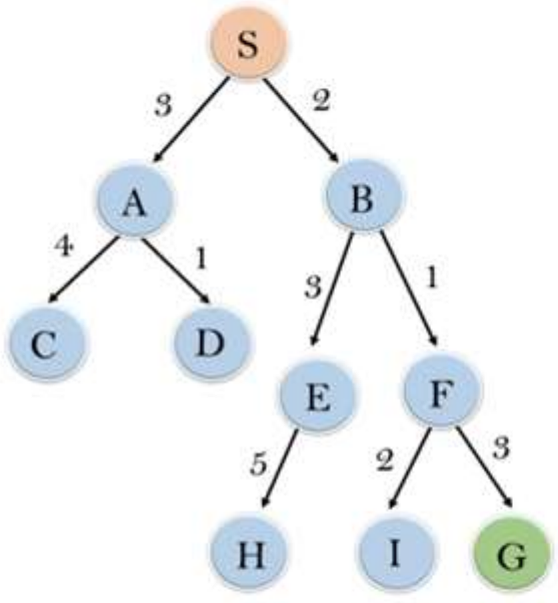
➢ **For better understanding, please Click here and watch the video.**

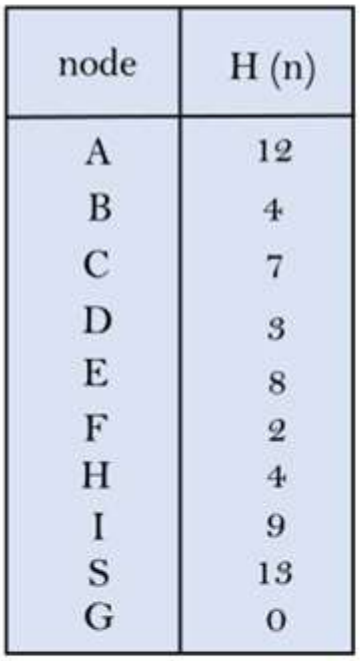
❖ **A problem can be defined formally by components:**

- **Initial state:** Any state can be designated as the initial state.
  - Note that goal can be reached from exactly half of the possible initial states.
- **Actions:** Movements of the blank space Left, Right, Up, or Down.
  - Different subsets of these are possible depending on where the blank is.
- **Transition model:** Given a state and action, this returns the resulting state;
- **Goal test:** This checks whether the state matches the goal configuration
- **Path Cost:** Each step costs 1, so the path cost is the number of steps in the path.

❖ **Optimal solution**: Is one whose measure of quality is close to the best that could theoretically be obtained.

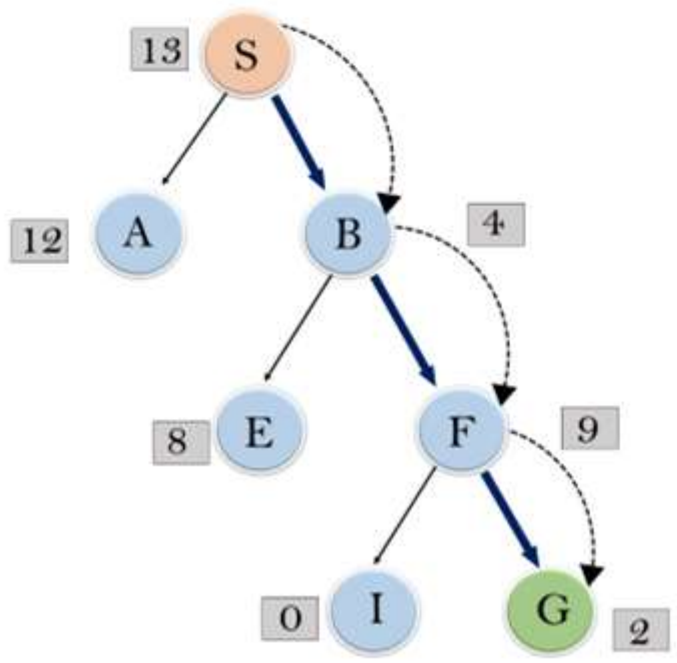
- ❖ **Frontier:** a set of leaf nodes.
- ❖ **Loopy path:** infinite, unsolvable.
- ❖ **Explored set:** keeps track of the nodes that have already bee expanded.
- ✪ ***Informed (Heuristic) search:*** Informed search uses problem-specific knowledge (heuristics) to find solutions efficiently, unlike uninformed search algorithms.
    - **Searching strategy:** defined by picking the order of node expansion.
    - **Heuristic function h(n):** estimated cost of the cheapest path from node n to a goal node.
        1. Best-first search algorithm (Greedy search).
        2. A* Search Algorithm.
- **Best-first search algorithm (Greedy search):** Best-first search is an informed search algorithm that selects nodes to explore based on a heuristic function, which estimates the cost to reach the goal. It always explores the most promising node first according to the heuristic evaluation.





node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



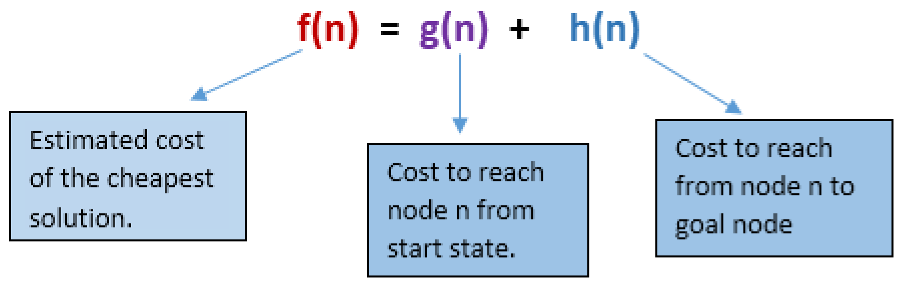
➢ **Properties of Best-first search algorithm:**

- **Time Complexity:** The worst case time complexity of Greedy best first search is O(b^m).
- **Space Complexity:** The worst case space complexity of Greedy best first search is O(b^m). Where, m is the maximum depth of the search space.
- **Completeness:** Greedy best-first search is also incomplete, even if the given state space is finite.
- **Optimality:** Greedy best first search algorithm is not optimal.

➢ **For better understanding, please Click here and watch the video.**

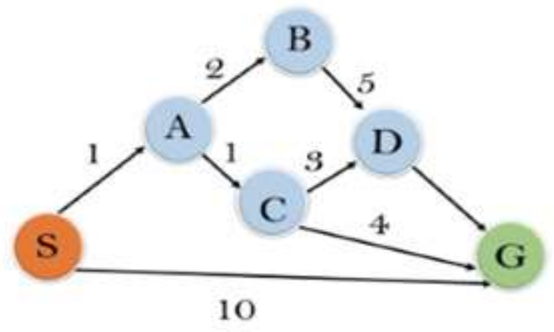
A* search algorithm: Minimizing the total estimated cost solution. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

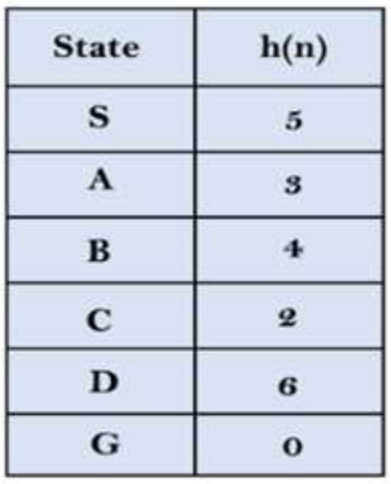
In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

In the below 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.

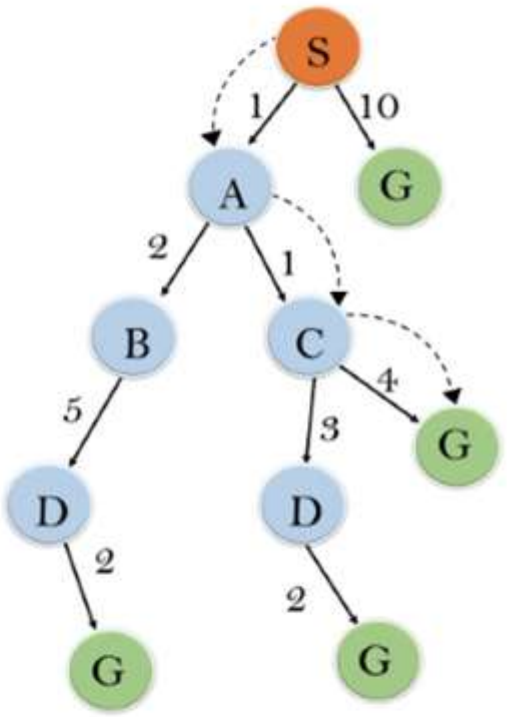
Here we will use OPEN and CLOSED li





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

Solution:



➢ **For better understanding, please Click here and watch the video.**

❖ **Optimality of A* Search Algorithm:**

- **Heuristic may be:**
  - ✓ h(n) > actual cost
  - ✓ h(n) = actual cost
  - ✓ h(n) < actual cost

(1) h(n) > actual cost [Optimum solution can be overlooked, optimum solution is not possible]

(2) h(n) = actual cost [Best case scenario, if h(n) approximates actual cost, searching uses minimum of node to the goal]

(3) h(n) < actual cost [Admissible, Consistent Heuristics]

❖ **Admissible heuristic:** never overestimates the cost to reach the goal.

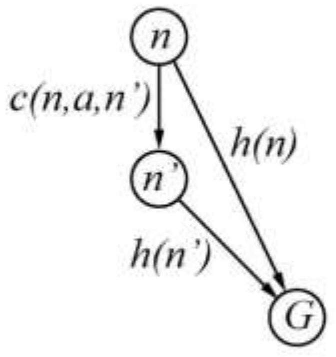
❖ **Consistent heuristic:**

A heuristic is *consistent* if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \ [= f(n)] \end{aligned}$$



I.e., $f(n)$ is nondecreasing along any path.

❖ **Prove that if a heuristic is consistent, it must be admissible.**

We can prove that consistency implies admissibility through induction.

Recall that consistency is defined such that $h(n) \leq c(n, n+1) + h(n+1)$.

Base Case: We begin by considering the $n-1$th node in any path where n denotes the goal state.

$$h(n-1) \leq c(n-1, n) + h(n) \qquad (1)$$

Because n is the goal state, by definition, $h(n) = h^*(n)$. Therefore, we can rewrite the above as

$$h(n-1) \leq c(n-1, n) + h^*(n)$$

and given that $c(n-1, n) + h^*(n) = h^*(n-1)$, we can see:

$$h(n-1) \leq h^*(n-1)$$

which is the definition of admissibility!

Inductive Step: To see if this is always the case, we consider the $n-2$nd node in any of the paths we considered above (e.g. where there is precisely one node between it and the goal state). The cost to get from this node to the goal state can be written as

$$h(n-2) \leq c(n-2, n-1) + h(n-1)$$

From our base case above, we know that

$$h(n-2) \leq c(n-2, n-1) + h(n-1) \leq c(n-2, n-1) + h^*(n-1)$$

$$h(n-2) \leq c(n-2, n-1) + h^*(n-1)$$

And again, we know that $c(n-2, n-1) + h^*(n-1) = h^*(n-2)$, so we can see:

$$h(n-2) \leq h^*(n-2)$$

By the inductive hypothesis, this holds for all nodes, proving that consistency does imply admissibility!

*******Solve exercise from Book.*******