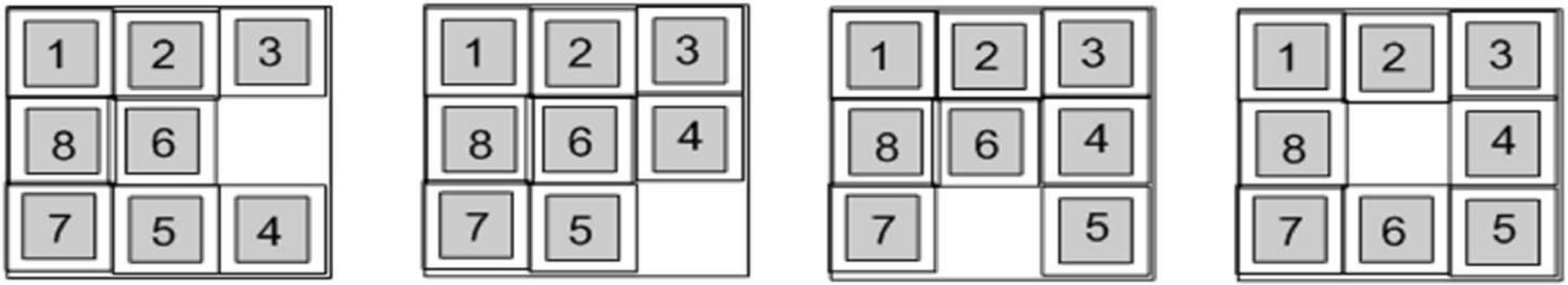
Artificial Intelligence

Lecture 27

Problem Solving by Search

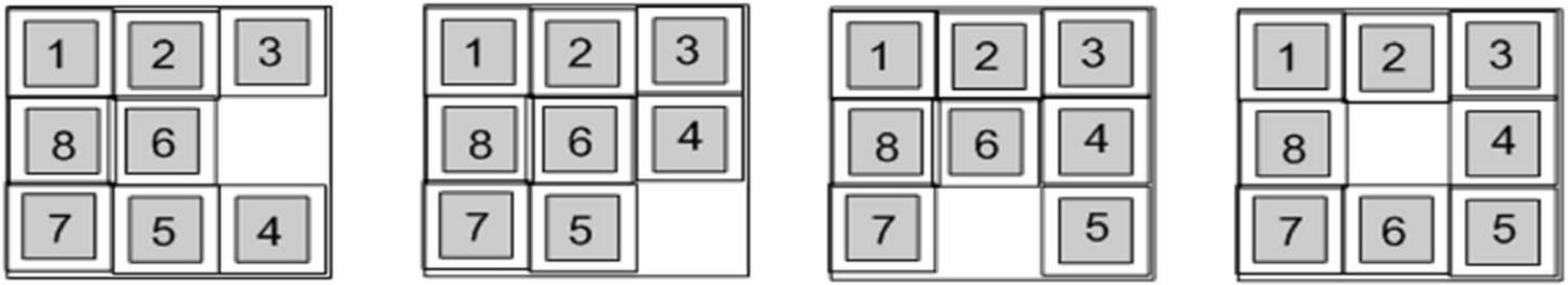
Prepared by:

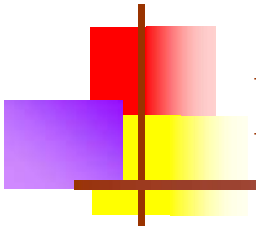
Md. Mijanur Rahman, Prof. Dr.
Dept. of CSE, JKKNIU
Email: mijanjkkniu@gmail.com



Lecture Outlines

- **Tree searches and search strategies:**
  - Depth Limited Search Algorithm
  - Iterative Deepening Depth First Search (IDDFS)
  - Informed or Heuristic Search
    - Best First Search Algorithm (Greedy search)
- **…**





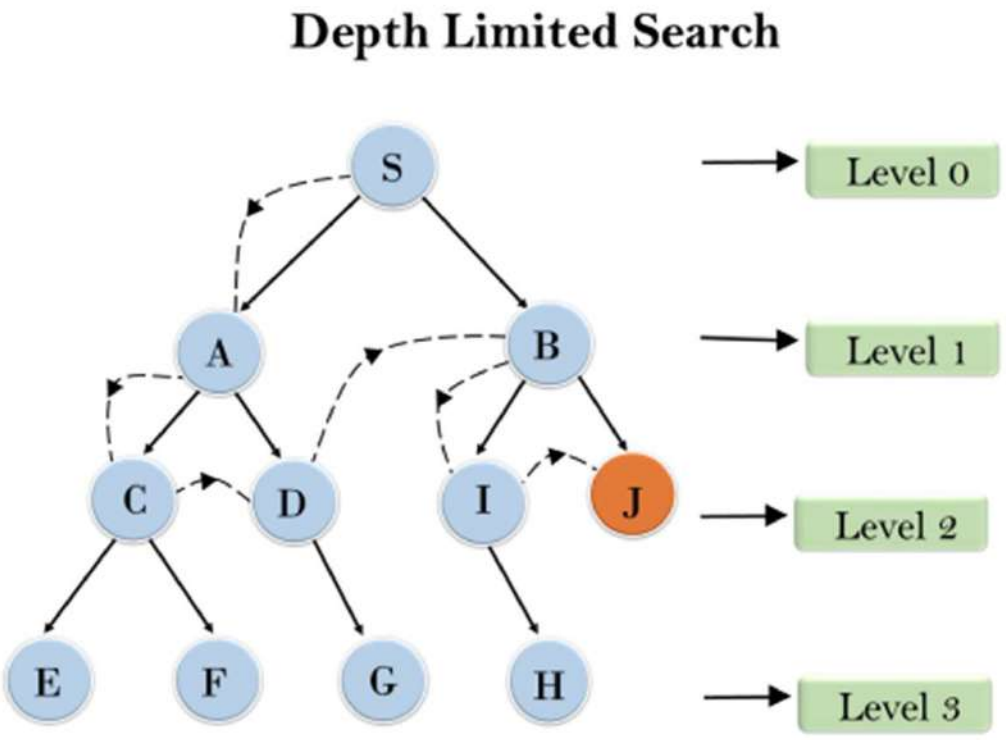
Depth Limited Search Algorithm…

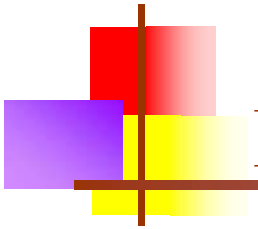
- **Depth Limited Search (DLS)**
- A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

- Depth-limited search can be terminated with two Conditions of failure:
  - Standard failure value: It indicates that problem does not have any solution.
  - Cutoff failure value: It defines no solution for the problem within a given depth limit.

Depth Limited Search Algorithm…

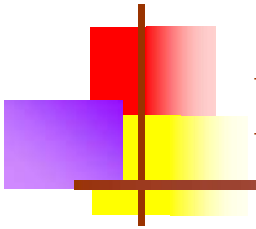
- **Example:**





Depth Limited Search Algorithm…

- This algorithm essentially follows a similar set of steps as in the DFS algorithm:
  1. The start node or node 1 is added to the beginning of the stack.
  2. Then it is marked as visited and if node 1 is not the goal node in the search, then we push second node 2 on top of the stack.
  3. Next, we mark it as visited and check if node 2 is the goal node or not.
  4. If node 2 is not found to be the goal node then we push node 4 on top of the stack
  5. Now we search in the same depth limit and move along depth-wise to check for the goal nodes.
  6. If Node 4 is also not found to be the goal node and depth limit is found to be reached then we retrace back to nearest nodes that remain unvisited or unexplored.
  7. Then we push them into the stack and mark them visited.
  8. We continue to perform these steps in iterative ways unless the goal node is reached or until all nodes within depth limit have been explored for the goal.
- When we compare the above steps with DFS we may found that DLS can also be implemented using the queue data structure.



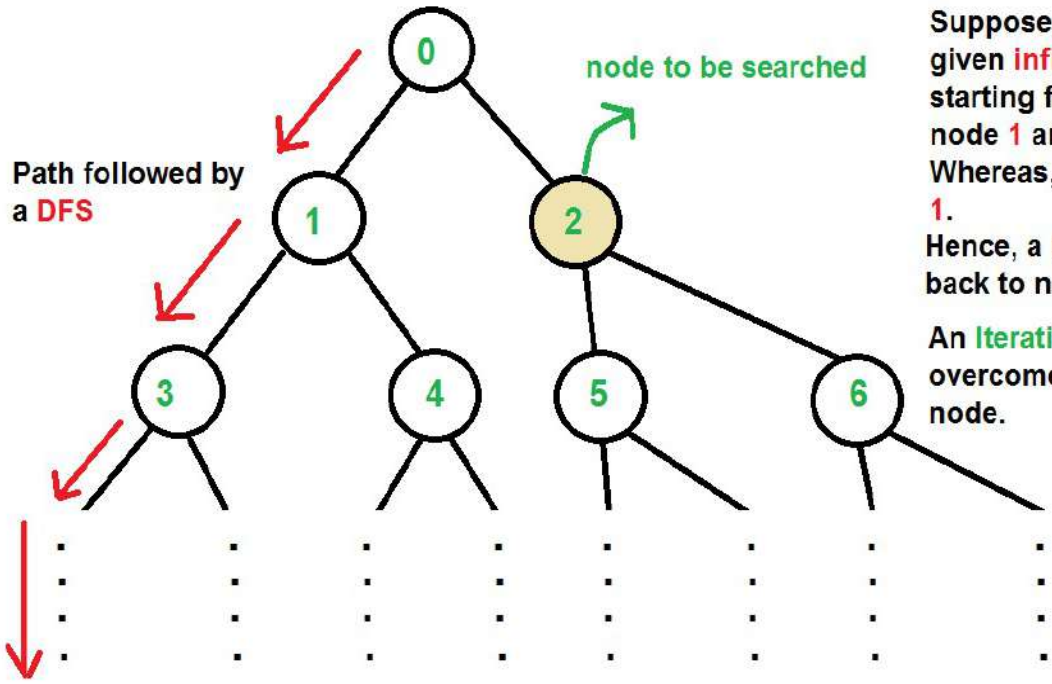
Depth Limited Search Algorithm

- **Advantages:**
  - Depth-limited search is Memory efficient.
- **Disadvantages:**
  - Depth-limited search also has a disadvantage of incompleteness.
  - It may not be optimal if the problem has more than one solution.

- **Completeness:** DLS search algorithm is complete if the solution is above the depth-limit.
- **Time Complexity:** Time complexity of DLS algorithm is **$O(b^{\ell})$**.
- **Space Complexity:** Space complexity of DLS algorithm is **$O(b \times \ell)$**.
- **Optimal:** Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

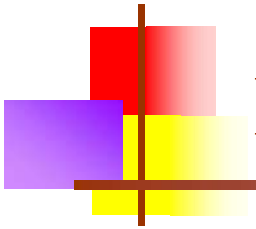
IDDFS…

- Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search (IDDFS)



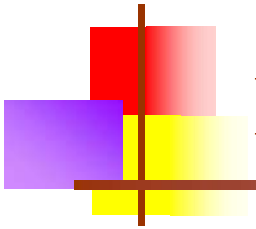
Suppose, we want to find node- '2' of the given infinite undirected graph/tree. A DFS starting from node- 0 will dive left, towards node 1 and so on.
Whereas, the node 2 is just adjacent to node 1.
Hence, a DFS wastes a lot of time in coming back to node 2.

An Iterative Deepening Depth First Search overcomes this and quickly find the required node.



IDDFS…

- **IDDFS** combines depth-first search's space-efficiency and breadth-first search's fast search (for nodes closer to root).
- **How does IDDFS work?**
- IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.
  - An important thing to be noted that we visit top level nodes multiple times. The last (or max depth) level is visited once, second last level is visited twice, and so on. It may seem expensive, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level.
  - So it does not matter much if the upper levels are visited multiple times.

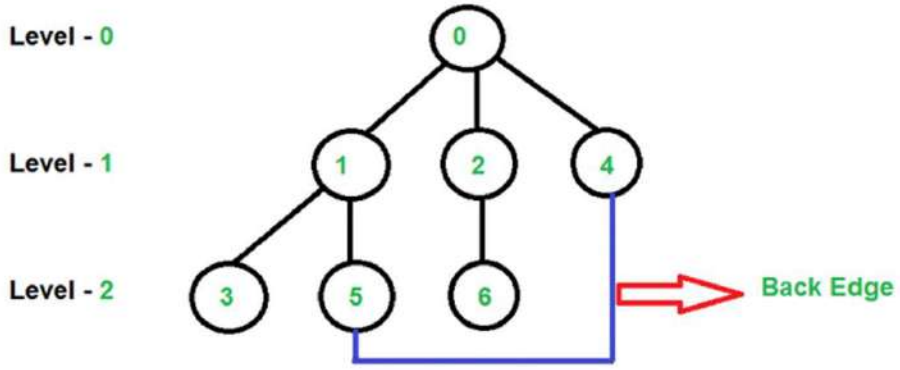


IDDFS…

- **Search Algorithm:**
- ```
  limit = 0;
  found = false;
  while (not found) {
      found = limitedDFS(root, limit, 0);
      limit = limit + 1;
  }
  ```
- ```
  //depth limited search
  ```
- ```
  boolean limitedDFS(Node node, int limit, int depth) {
      if (depth > limit) return failure;
      if (node is a goal node) return success;
      for each child of node {
          if (limitedDFS(child, limit, depth + 1))
              return success;
      }
      return failure;
  }
  ```
- This searches to depth 0 (root only), then if that fails it searches to depth 1, then depth 2, and so on.
  ```

IDDFS…

- **Illustration:**
- There can be two cases-
  - ***a) When the graph has no cycle:*** This case is simple. We can DFS multiple times with different height limits.
  - ***b) When the graph has cycles.*** This is interesting as there is no visited flag in IDDFS.

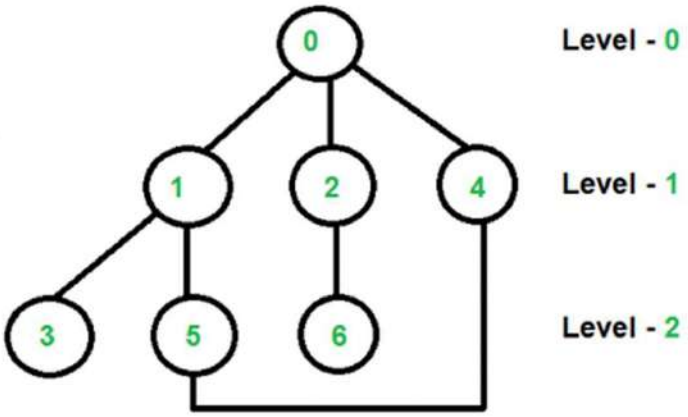


Although, at first sight, it may seem that since there are only 3 levels, so we might think that Iterative Deepening Depth First Search of level 3, 4, 5,...and so on will remain same. But, this is not the case. You can see that there is a cycle in the above graph, hence IDDFS will change for level-3,4,5..and so on.

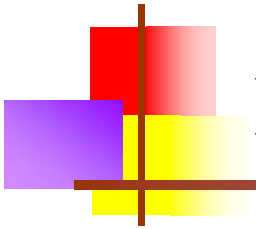
IDDFS…

- **Illustration:**
  - ***When the graph has cycles.***

Depth	Iterative Deepening Depth First Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1



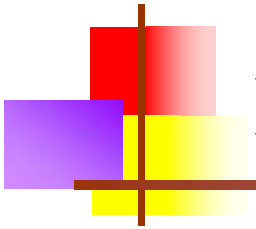
The explanation of the above pattern is left to the readers.



IDDFS…

- **Time Complexity:** Suppose we have a tree having branching factor ‘b’ (number of children of each node), and its depth ‘d’, i.e., there are $\mathbf{b^d}$ nodes.
- In an iterative deepening search, the nodes on the bottom level are expanded once, those on the next to bottom level are expanded twice, and so on, up to the root of the search tree, which is expanded d+1 times. So the total number of expansions in an iterative deepening search is-

$(d)b + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + b^d$
That is, Summation $[(d + 1 - i)b^i]$, from i = 0 to d
Which is same as $O(b^d)$



IDDFS…

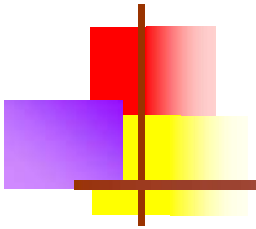
- **Time Complexity:**

$(d)b + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + b^d$
That is, Summation $[(d + 1 - i)b^i]$, from $i = 0$ to d
Which is same as $O(b^d)$

- After evaluating the expression, we find that asymptotically IDDFS takes the same time as that of DFS and BFS, but it is indeed slower than both of them as it has a higher constant factor in its time complexity expression.
- IDDFS is best suited for a complete infinite tree.

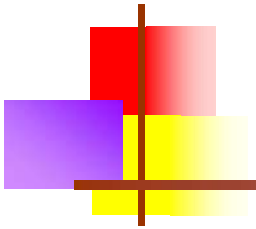
A comparison table between DFS, BFS and IDDFS

	Time Complexity	Space Complexity	When to Use ?
DFS	$O(b^d)$	O (d)	=> Don't care if the answer is closest to the starting vertex/root. => When graph/tree is not very big/infinite.
BFS	$O(b^d)$	$O(b^d)$	=> When space is not an issue => When we do care/want the closest answer to the root.
IDDFS	$O(b^d)$	O(bd)	=> You want a BFS, you don't have enough memory, and somewhat slower performance is accepted. In short, you want a BFS + DFS.



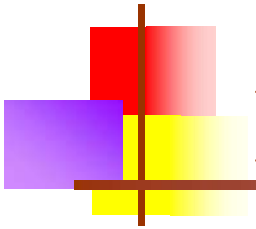
Time requirements for depth-first iterative deepening on binary tree

Nodes at each level	Nodes searched by DFS	Nodes searched by iterative DFS
1	1	1
2	+2 = 3	+3 = 4
4	+4 = 7	+7 = 11
8	+8 = 15	+15 = 26
16	+16 = 31	+31 = 57
32	+32 = 63	+63 = 120
64	+64 = 127	+127 = 247
128	+128 = 255	+255 = 502



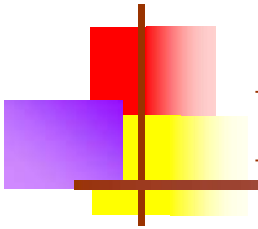
Time requirements on tree with branching factor 4

Nodes at each level	Nodes searched by DFS	Nodes searched by iterative DFS
1	1	1
4	+4 = 5	+5 = 6
16	+16 = 21	+21 = 27
64	+64 = 85	+85 = 112
256	+256 = 341	+341 = 453
1024	+1024 = 1365	+1365 = 1818
4096	+4096 = 5461	+5461 = 7279
16384	+16384 = 21845	+21845 = 29124



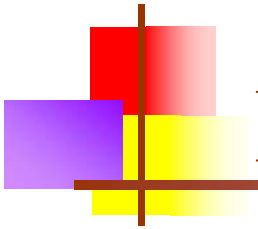
Informed Search

- 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.
- The informed search methods use an **evaluation** function to guide the search towards goal. This process uses an evaluation function:
  - Evaluation function: f(n)
  - Cost: g(n)=C
  - heuristic: h(n)=E (estimate of distance to goal)



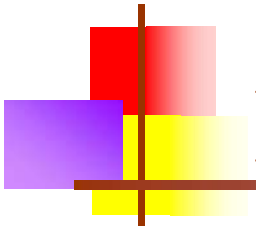
Heuristics Function

- 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 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.
- Admissibility of the heuristic function is given as:
  - $h(n) <= h^*(n)$
  - Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.



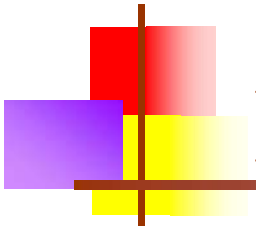
Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n).
- It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node *n* with the lowest heuristic value is expanded and generates all its successors and *n* is placed to the closed list. The algorithm continues unit a goal state is found.
- In the informed search, two main algorithms are:
  - **Best First Search Algorithm (Greedy search)**
  - **A* Search Algorithm**



Best-first Search Algorithm

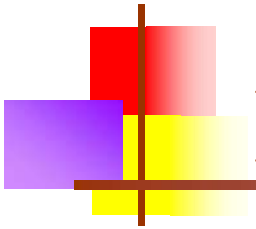
- Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the 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.
  - 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.



Best-first Search Algorithm

- **The best first algorithm is:**

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.



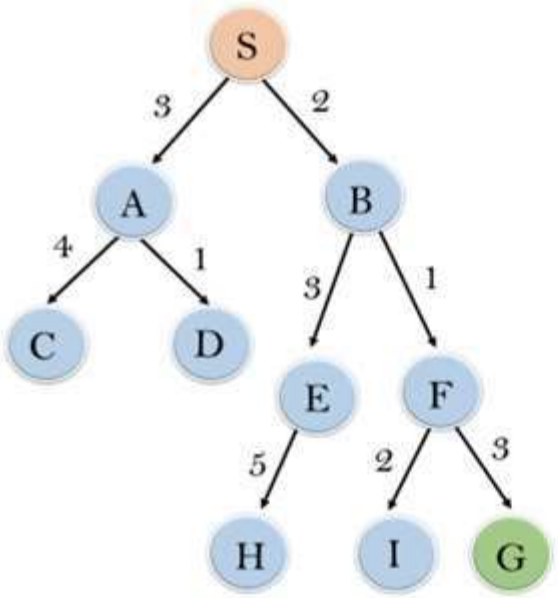
Best-first Search Algorithm

- Advantages:
  - Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
  - This algorithm is more efficient than BFS and DFS algorithms.

- Disadvantages:
  - It can behave as an unguided depth-first search in the worst case scenario.
  - It can get stuck in a loop as DFS.
  - This algorithm is not optimal.

Best-first Search Algorithm

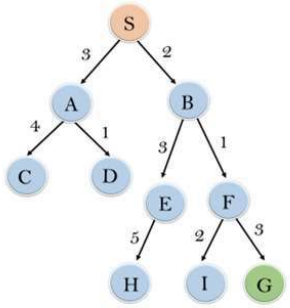
- Example:
  - Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.
  - In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists.



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

Best-first Search Algorithm

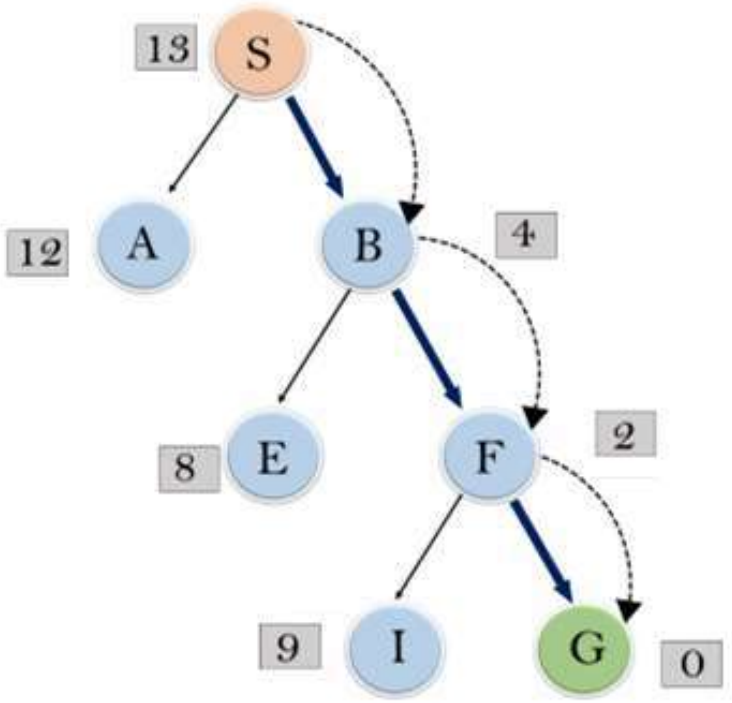
- Following are the iteration for traversing the given example.



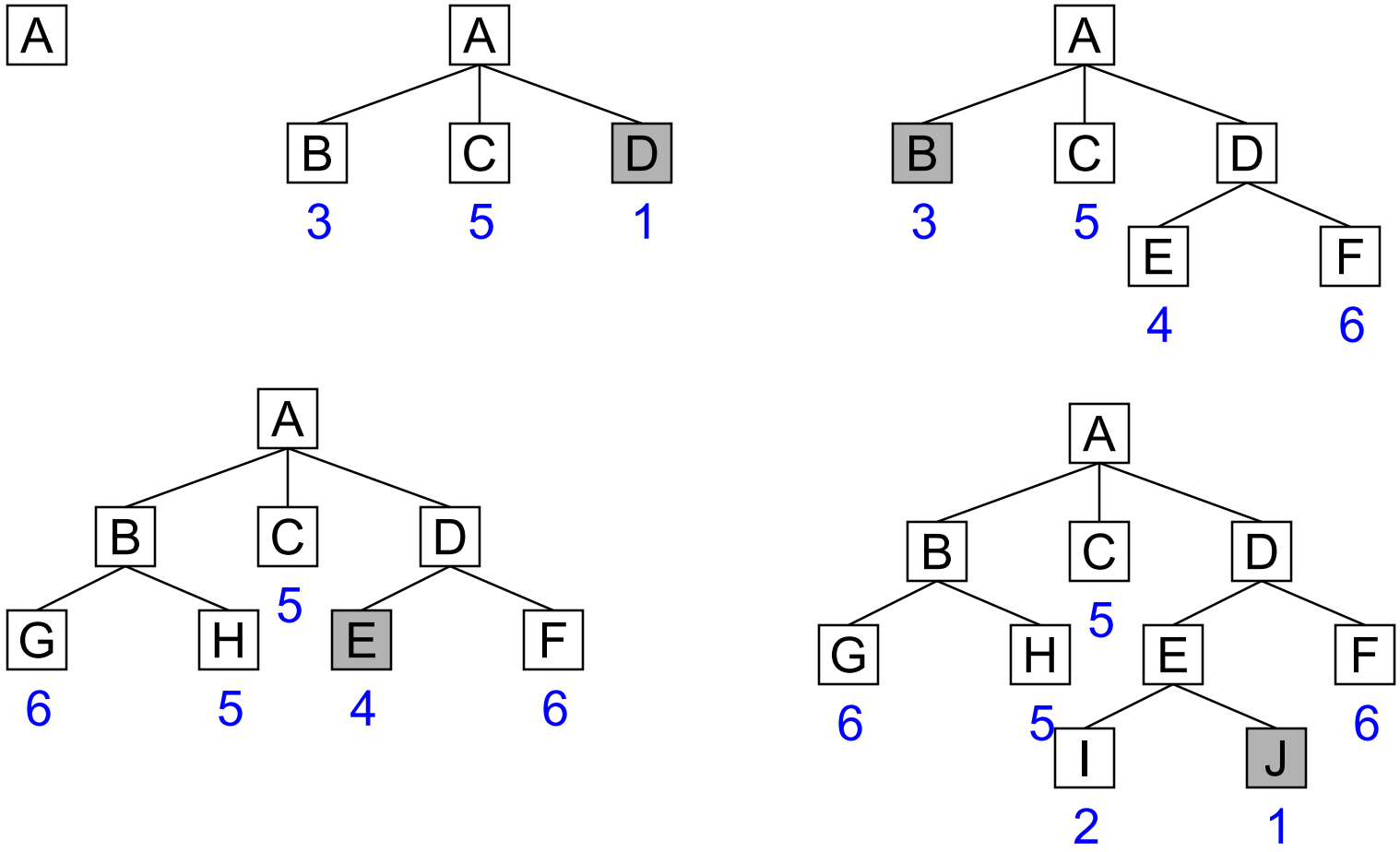
node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

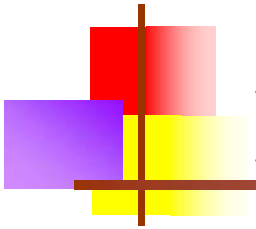
- Expand the nodes of S and put in the CLOSED list
- **Initialization:** Open [A, B], Closed [S]
- **Iteration 1:** Open [A], Closed [S, B]
- **Iteration 2:** Open [E, F, A], Closed [S, B]
  : Open [E, A], Closed [S, B, F]
- **Iteration 3:** Open [I, G, E, A], Closed [S, B, F]
  : Open [I, E, A], Closed [S, B, F, G]
- Hence the final solution path will be:

  S ----> B----->F----> G



Best-first Search Algorithm





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.
- **Complete:** Greedy best-first search is also incomplete, even if the given state space is finite.
- **Optimal:** Greedy best first search algorithm is not optimal.

Problem Solving by Search

TO BE CONTINUED…