UEM
श्रद्धावान लभते ज्ञानम्
UNIVERSITY OF ENGINEERING & MANAGEMENT
Good Education, Good Jobs

UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

Course Name : AI & ML

- Prime considerations of Problem Solving:-
- Represent the problem as proper state space;
- Identify goal state and determine optimal path from starting state to goal state;
- Search problem is related to two facts “ what to search” and “where to search”

- **Search Algorithm Terminologies:**
- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - **Search Space:** Search space represents a set of possible solutions, which a system may have.
  - **Start State:** It is a state from where agent begins **the search**.
  - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

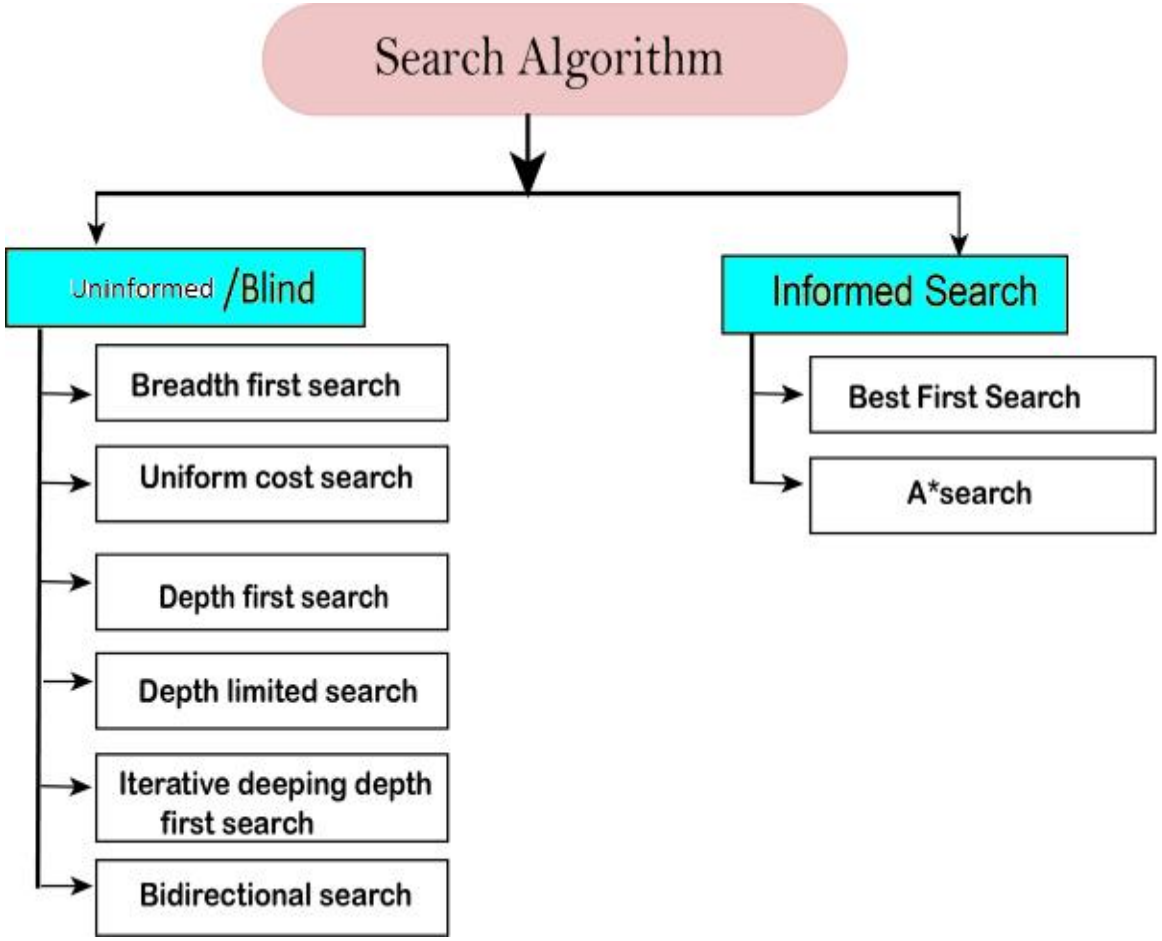
Properties of Search Algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least a solution exists for any random input.

Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.


Search Algorithm
Uninformed /Blind
Breadth first search
Uniform cost search
Depth first search
Depth limited search
Iterative deeping depth first search
Bidirectional search
Informed Search
Best First Search
A*search

Uninformed/Blind Search:

- The uninformed search does not contain any domain knowledge such as closeness, the location of the goal.
- It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.
- Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

Informed Search

- Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search.
- Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- Informed search is also called a Heuristic search.

- Different Approaches:-
- Generate & Test [BFS, DFS, Hill-Climbing, Simulated Annealing]
- Heuristic Search [Best-First-Search, A*, AO*]
- Adversary Search [MINIMAX Algorithm, Alpha-Beta Cutoff]

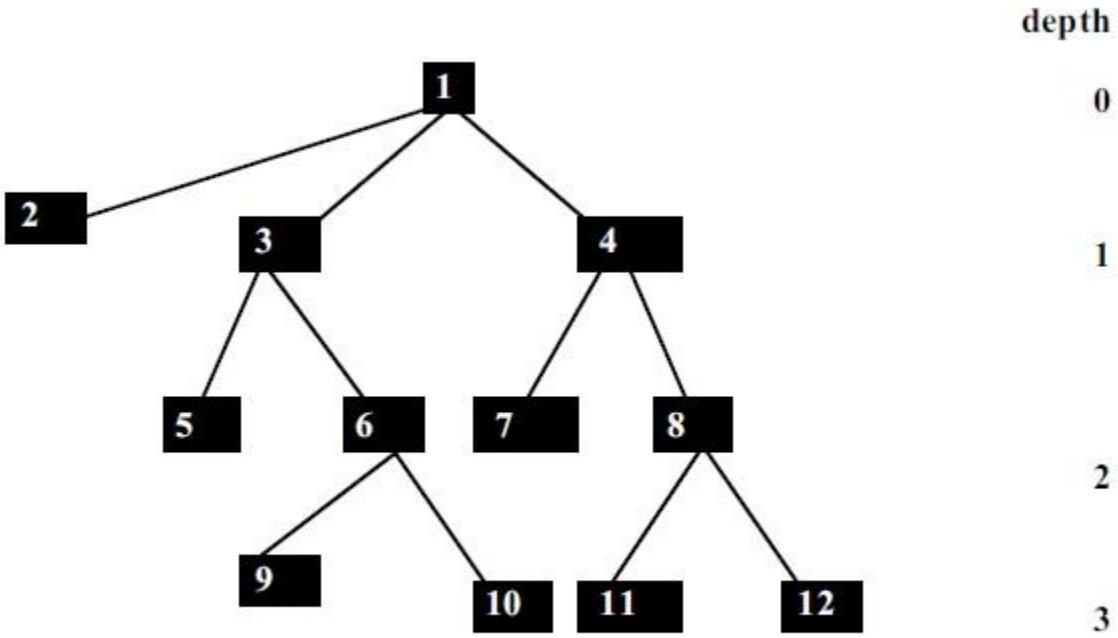
Procedure Generate & Test

- Begin
- Repeat
- Generate a new state and call it current-state;
- Until current-state = Goal;
- End.

Procedure Breadth-first-search

- **Begin**
- **i) Place the starting node in a queue;**
- **ii) Repeat**
- Delete queue to get the front element;
- **If** the front element of the queue = goal**,** return success and stop;
- **Else** do
- **Begin**
- insert the children of the front element, if exist, in any order at the rear end of the queue;
- **End**
- **Until** the queue is empty;
- **End.**

- **Procedure Depth first search**
- **Begin**
- 1. Push the starting node at the stack, pointed to by the stack-top;
- 2. **While** stack is not empty do
- **Begin**
- Pop stack to get stack-top element;
- **If** stack-top element = goal, return success and stop
- **Else** push the children of the stack-top element in any order into the stack;
- **End while;**
- **End.**



. The order of traversal in a tree of depth 3 by

Solution as per DFS (source : 1, Goal: 10)

Stack(open list)	Closed list (Already visited)
1	
2, 3, 4	1
3,4	1,2
5,6,4	1,2,3
6,4	1,2,3,5
9,10,4	1,2,3,5,6
10, 4	1,2,3,5,6,9
4	1,2,3,5,6,9,10

Goal reached. Path is : 1->2->3->5->6->9->10

Solution as per BFS (source : 1, Goal: 10)

Queue (open list)	Closed list (Already visited) (Parent)
1	
2, 3, 4	1(null)
3,4	1(null),2(1)
4,5,6	1(null),2(1),3(1)
5,6,7,8	1(null),2(1),3(1),4(1)
6,7,8	1(null),2(1),3(1),4(1), 5(3)
7,8,9,10	1(null),2(1),3(1),4(1), 5(3),6(3)
8,9,10	1(null),2(1),3(1),4(1), 5(3),6(3),7(4)
9,10,11,12	1(null),2(1),3(1),4(1), 5(3),6(3),7(4),8(4)
10,11,12	1(null),2(1),3(1),4(1), 5(3),6(3),7(4),8(4),9(6)
11,12	1(null),2(1),3(1),4(1), 5(3),6(3),7(4),8(4),9(6), 10(6)

Goal reached. Path is : 1- >3->6 ->10

Depth Limited Search Algorithm

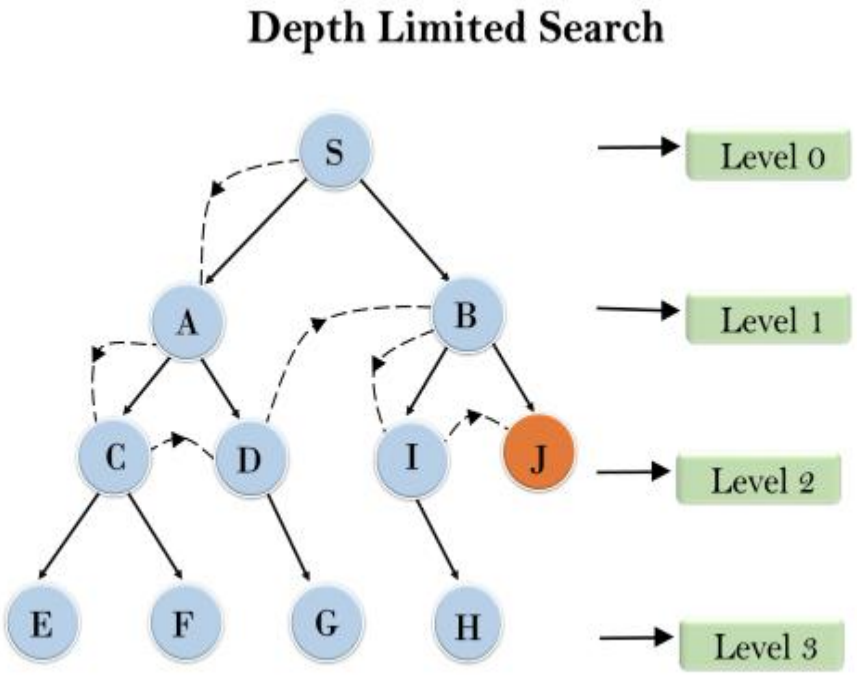
We are given a graph G and a depth limit ‘L'. Depth Limited Search is carried out in the following way:

1.Set STATUS=1(ready) for each of the given nodes in graph G.
2.Push the Source node or the Starting node onto the stack and set its STATUS=2(waiting).
3.Repeat steps 4 to 5 until the stack is empty or the goal node has been reached.
4.Pop the top node T of the stack and set its STATUS=3(visited).
5.Push all the neighbours of node T onto the stack in the ready state (STATUS=1) and
with a depth less than or equal to depth limit 'l' and set their STATUS=2(waiting).
(END OF LOOP)
6.END

When one of the following instances are satisfied, a Depth Limited Search can be terminated.

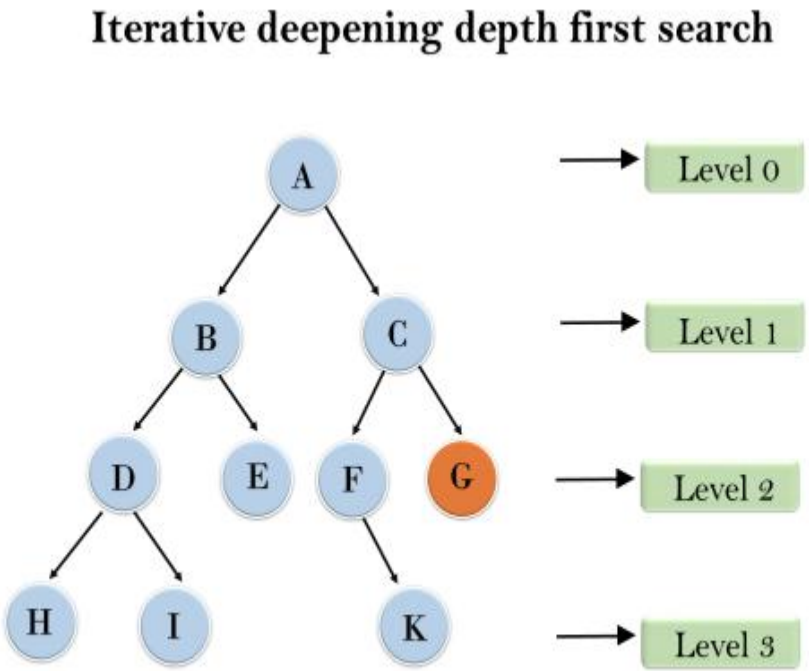
- When we get to the target node.
- Once all of the nodes within the specified depth limit have been visited.

Source: S; Target: J;Limit L =2
Push nodes with their level and check level limit.
Stack: (S,0)
Pop s, push (A,1), (B,1).
Pop A, push (C,2),(D,2)
Stack: (C,2), (D,2), (B,1)
Pop C, but not allowed to add E & F. Similarly pop D and don't add G. Pop B and push (I,2), J(2).
Stack: (I,2), J(2).
Pop I, nothing to add.
Pop J and reached goal.



- **Procedure Iterative-deepening DFS**
- **Begin**
- 1. Set current depth cutoff =1;
- 2. Put the initial node into a stack, pointed to by stack-top;
- 3. **While** the stack is not empty and the depth is within the given depth cut-off do
- **Begin**
- Pop stack to get the stack-top element;
- **if** stack-top element = goal, return it and stop
- **else** push the children of the stack-top in any order into the stack;
- **End While;**
- 4. Increment the depth cut-off by 1 and repeat
- through step 2;
- **End.**

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.



Thank You