Problem Solving by Search

Mr.R.S.Mirajkar

DYPCET, Kolhapur

A search problem

- Many interesting problems in science and engineering are solved using search

- A search problem is defined by:

A search space – The set of objects among which we search for the solution

Examples: routes between cities

A goal condition – Characteristics of the object we want to find in the search space? –

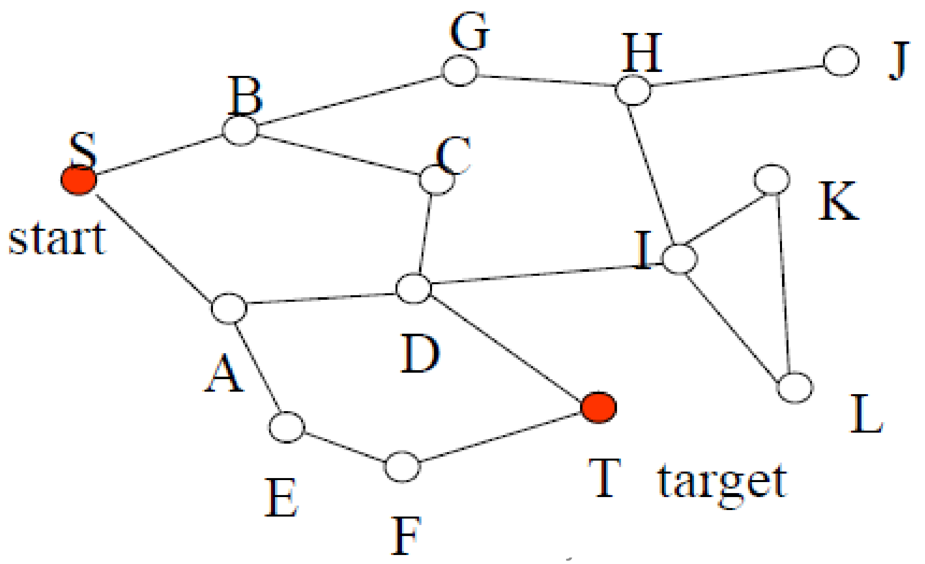
Examples: Path between cities A and B

Graph representation of a search problem

- Search problems can be often represented using **graphs**
- Typical example: **Route finding**

– Map corresponds to the graph, nodes to cities, links valid moves via available connections

– **Goal:** find a route (sequence of moves) in the graph from S to T



Graph Search Problems

- Search problems can be often represented as graph search problems:
- **Initial state** – State (configuration) we start to search from (e.g. start city, initial game position)
- **Operators:** – Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle)
- **Goal condition:** – Defines the target state (destination, winning position)

- **Search space** is now defined indirectly through:

The initial state + Operators

- There are two ways in which the AI problem can be represented.
  - **State Space Representation**
  - Problem Reduction

State Space Representation

- State Space Representation consist of defining an INITIAL State (from where to start), the GOAL State (The destination) and then we follow certain set of sequence of steps (called States).

- **State:** AI problem can be represented as a well formed **set of possible states.**
- State can be **Initial State**
- i.e. starting point, Goal State i.e. destination point and various other possible states between them which are formed by applying certain set of rules
- **Space:** In an AI problem the exhaustive set of all possible states is called space.
- **Search:** It takes the initial state to goal state by applying certain set of valid rules while moving through space of all possible states.

- Initial State
- Set of valid rules
- Goal State

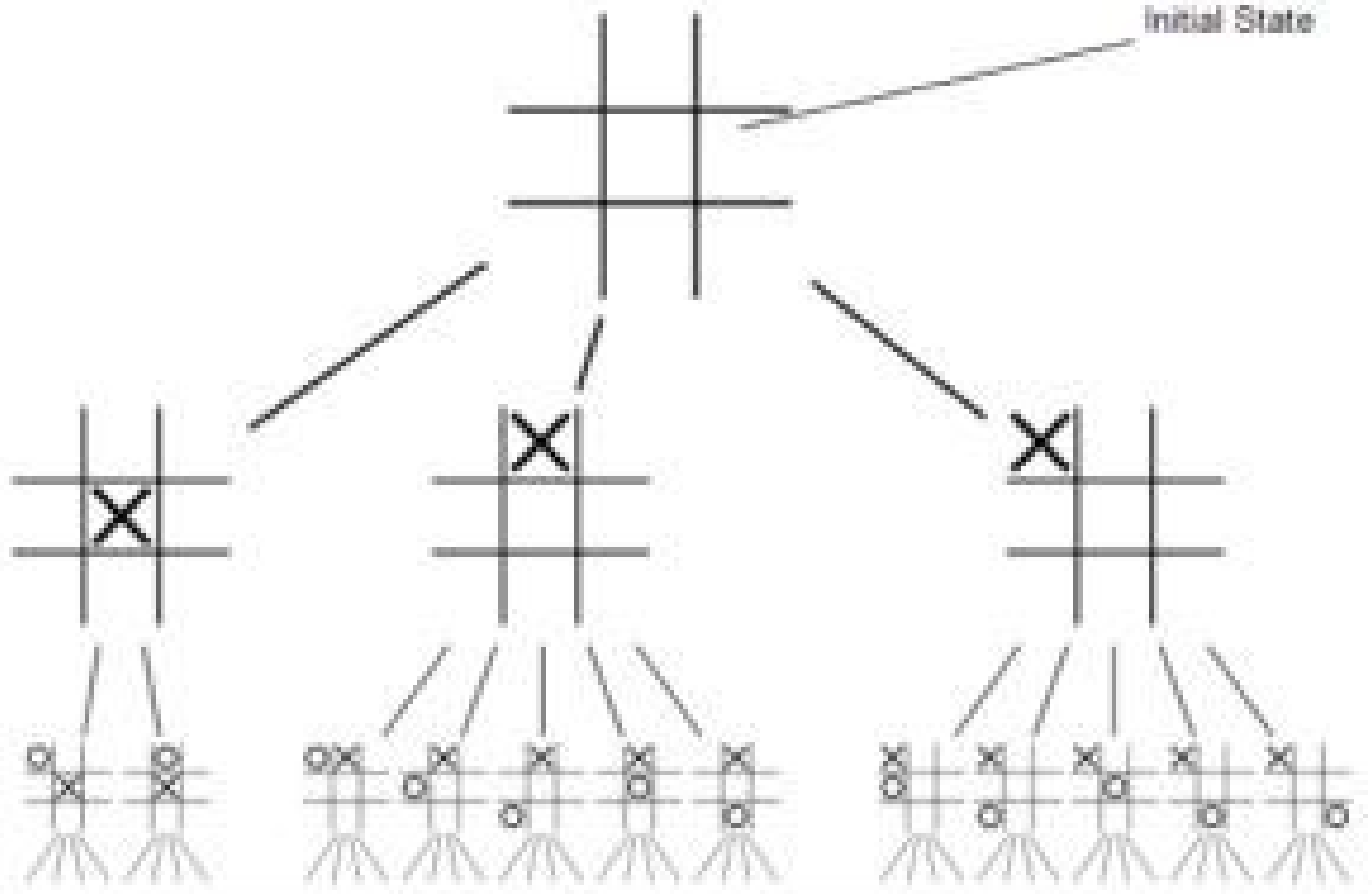
- A **set of all possible states for a given problem is known as state space representation of the problem.**

- **For example:** In chess game: The initial position of all the pieces on a chess board defines the **initial state.**
- The rules of playing chess defines the set of legal rules and Goal state is defined by any possible board position corresponding to checkmate or a draw state.
- There can be more than one Goal state possible.

State Space Representation of Tic Tac Toe game

- Starting from the initial state as we move on applying rules of putting X (cross) or O (Zero) we keep on generating the states,
- Hence the set of states **all such generated states is called space**, unless we reach one of the goal states that can be a win situation or a draw situation.
- The new state is generated from the earlier one is by applying the a **control strategy**

State Space Representation of Tic Tac Toe Game

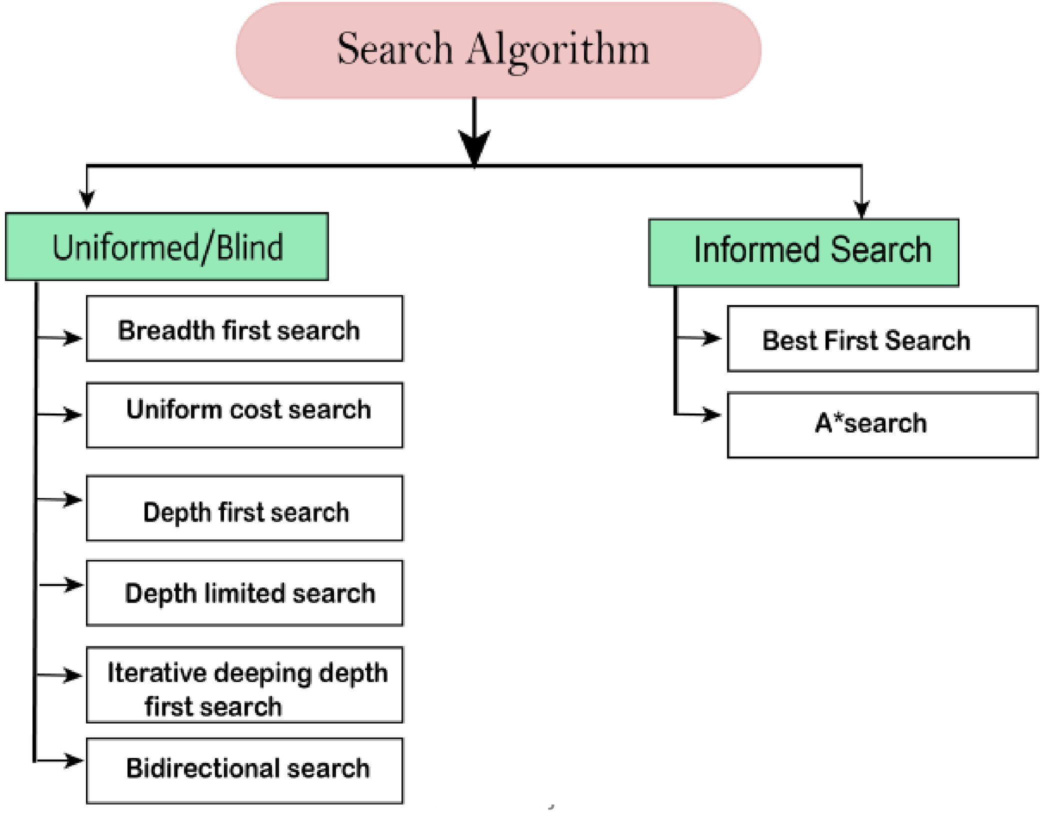


- State space representation are very advantageous in AI problems as the whole state space is given it **becomes easy to find the solution** path that leads from initial state to goal state.

- The basic job is to create such algorithms which can search through the problem space and find out the best solution path.

Types of search algorithms

- Based on the search problems we can classify the search algorithms into **uninformed (Blind search)** search and **informed search (Heuristic search)** algorithms.



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.

- **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**.

Breadth-first Search

- Breadth-first search is the most common search strategy for traversing a tree or graph.
- This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and **expands all successor node at the current level before moving to nodes of next level.**
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

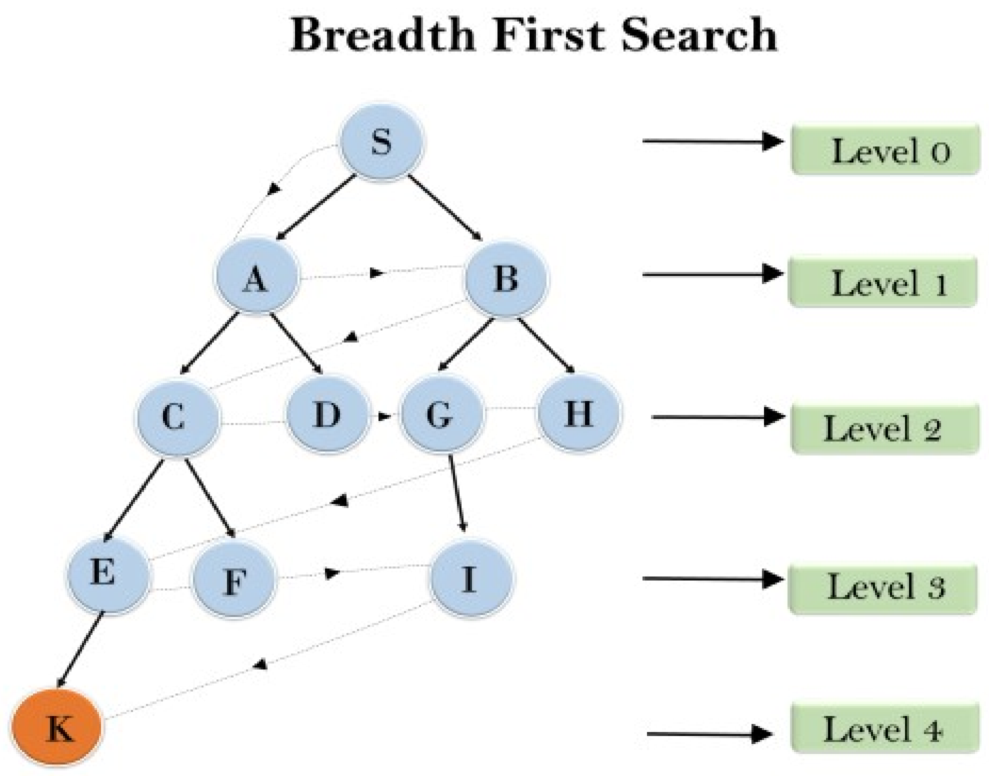
- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- 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.

Example: 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



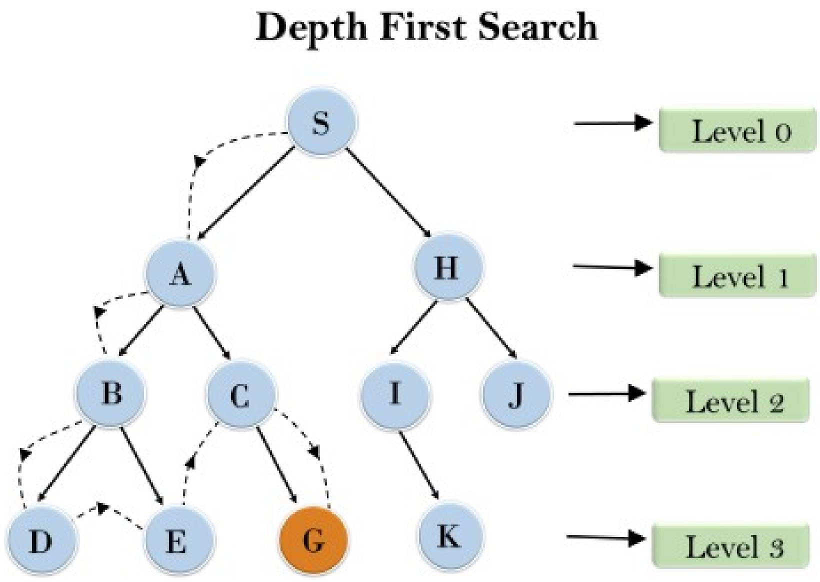
Depth-first Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and **follows each path to its greatest depth node** before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

- **Advantages:**
- 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.

Example:

- 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.



Iterative deepening depth-first Search:

- **Combination of DFS and BFS algorithms.**
- This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "**depth limit**", and it keeps increasing the depth limit after each iteration until the goal node is found.
- Restriction on each level instead of going to leaf node (like in DFS)
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- **The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.**

- **Advantages:**

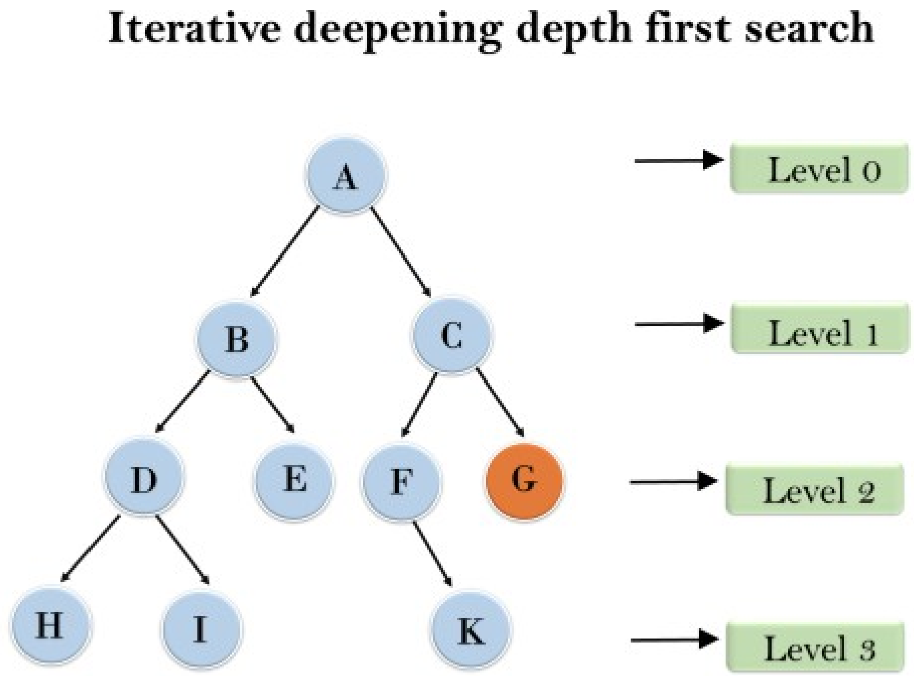
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

- **Disadvantages:**

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

- Following tree structure is showing the iterative deepening depth-first search.
- IDDFS algorithm performs various iterations until it does not find the goal node.
- Uninformed Search Algorithms
- 1'st Iteration-----> A
- 2'nd Iteration----> A, B, C
- 3'rd Iteration------>A, B, D, E, C, F, G



- In the 3rd iteration, the algorithm will find the goal node.

Bidirectional Search Algorithm:

- Bidirectional search algorithm runs two simultaneous searches, one form initial state called as **forward-search** and other from goal node called as **backward-search**, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

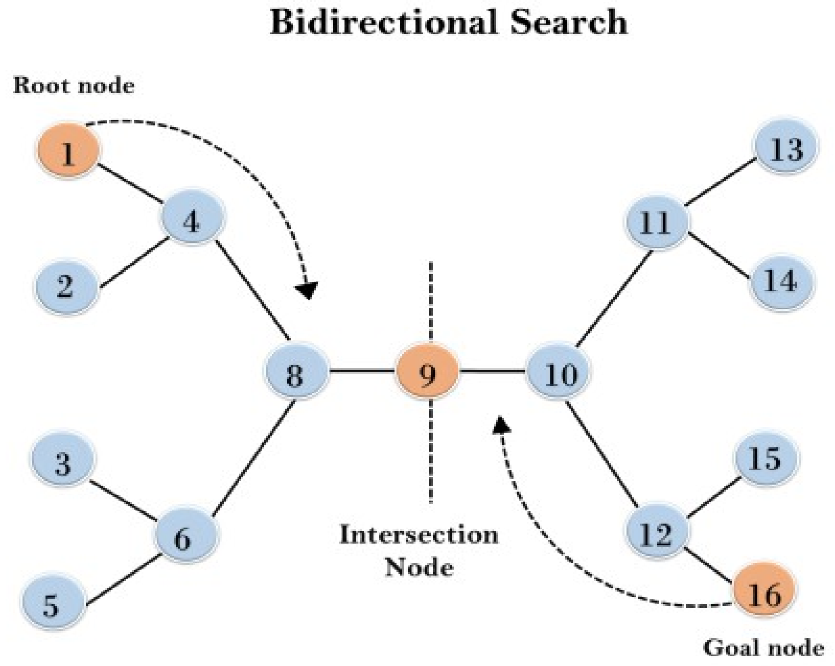
- **Advantages:**

  - Bidirectional search is fast.
  - Bidirectional search requires less memory

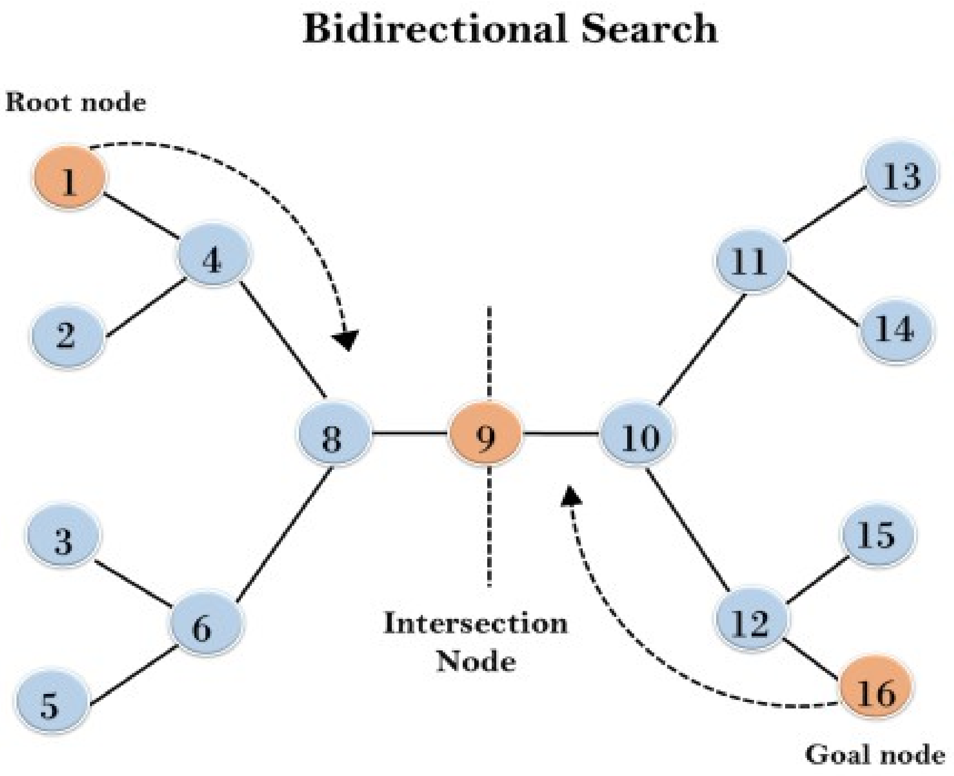
- **Disadvantages:**

  - Implementation of the bidirectional search tree is difficult.
  - In bidirectional search, one should know the goal state in advance.

- To find if there exists a path from vertex 1 to vertex 16.
- Execute two searches, one from vertex 1 and other from vertex 14.
- When both forward and backward search meet at vertex 9,
- we know that we have **found a path from node 1 to 16** and search can be terminated now.
- We can clearly see that we have successfully avoided unnecessary exploration.



- **Example:**
- 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.



Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a **recursive or backtracking** algorithm which is used in decision-making and game theory.

- **It provides an optimal move for the player assuming that opponent is also playing optimally.**

- Mini-Max algorithm **uses recursion** to search through the game-tree.

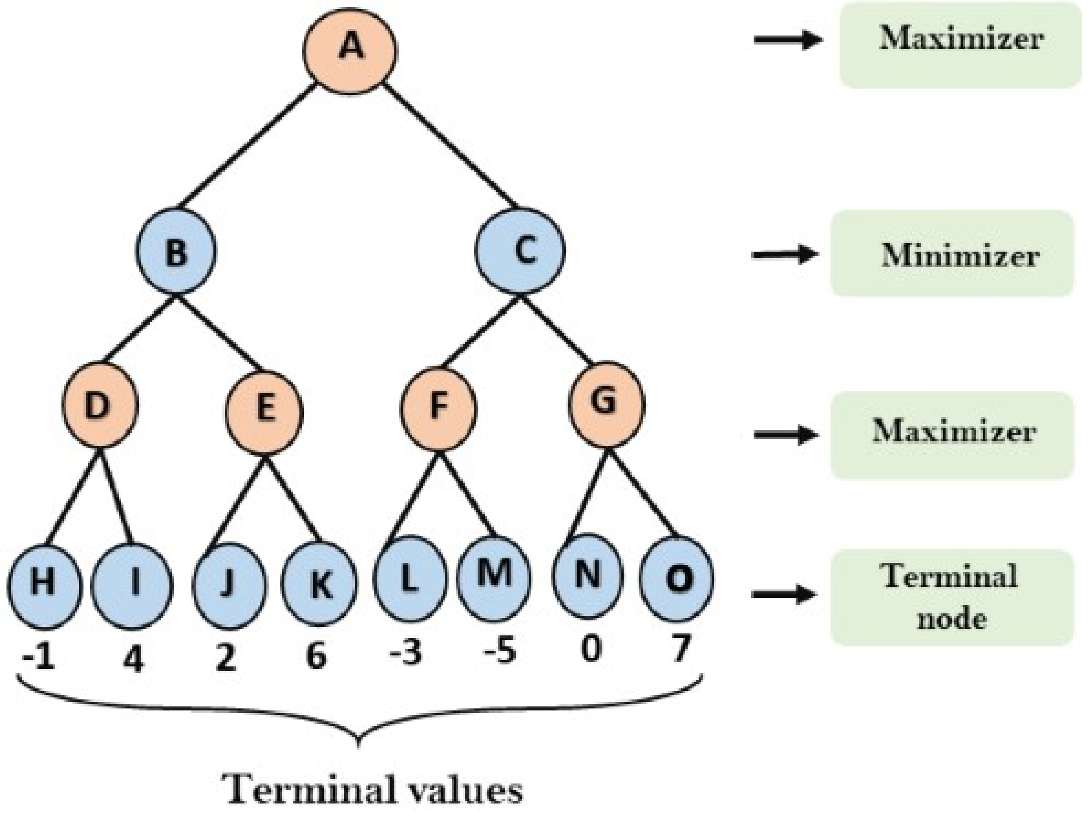
- Game playing in AI- Chess, Checkers, tic-tac-toe, go, and various two-players game.

- In this algorithm, two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The mini-max algorithm performs a **depth-first search** algorithm for the exploration of the complete game tree.
- The mini-max algorithm proceeds all the **way down to the terminal node of the tree, then backtrack the tree as the recursion**.

Working of Min-Max Algorithm:

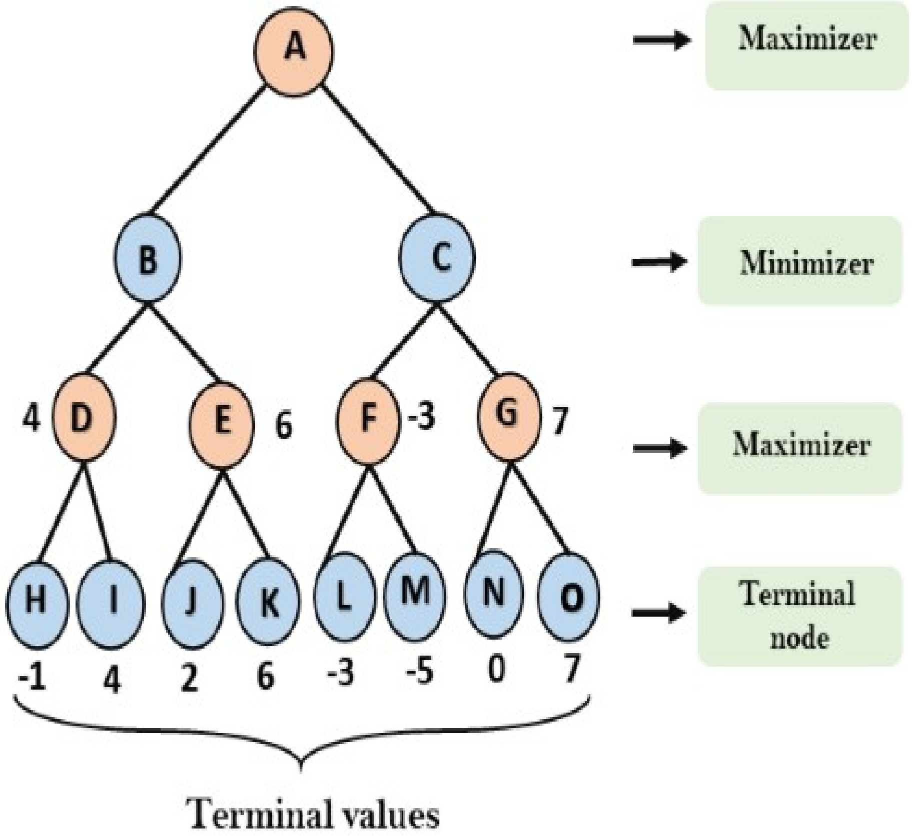
- Example of game-tree – Two player game.
- One is called Maximizer and other is called Minimizer.
- Maximizer will try to get the **Maximum possible score**, and Minimizer will try to get the **minimum possible score**.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will **compare those value and backtrack** the tree until the initial state occurs.

- **Step-1:** In the first step, the algorithm generates the entire game-tree and apply the **utility function to get the utility values for the terminal states**.
- The agents use the utility theory for making decisions. It is the **mapping from lotteries to the real numbers**.
- Suppose **maximizer takes first turn which has worst-case initial value = - infinity**, and **minimizer will take next turn which has worst-case initial value = + infinity**.

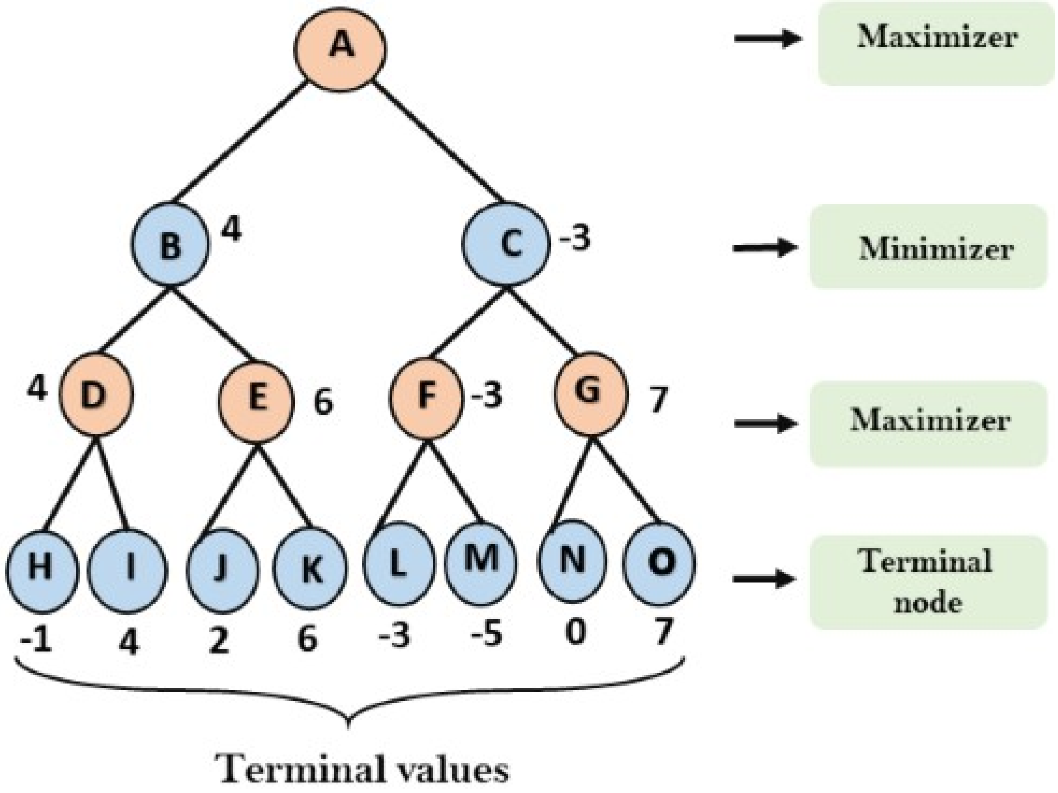


- **Step 2:** Now, first we find the utilities value for the **Maximizer**, its initial value is **-∞**,
- Compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.

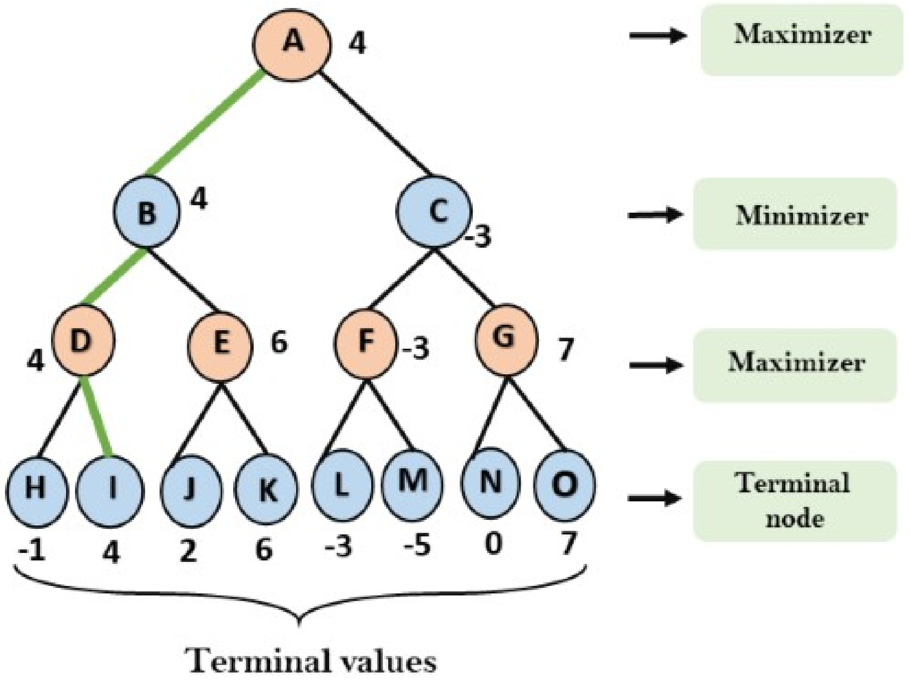
- For node D max(-1,- -∞) => max(-1, 4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3, -5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



- **Step 3:** In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.
- For node B= min(4, 6) = 4
- For node C= min (-3, 7) = -3



- **Step 4:** Now it's a turn for **Maximizer**, and it will again choose the maximum of all nodes value and find the maximum value for the root node.
- In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.
- For node A max(4, -3)= 4



Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm.
- It is an optimization technique for the minimax algorithm.
- In minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree.
- Since we cannot eliminate the exponent, but we can cut it to half.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is **called pruning.**
- This involves two threshold parameter **Alpha and beta** for future expansion, so it is called **alpha-beta pruning**.
- It is also called as Alpha-Beta Algorithm.

- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.
- Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

- The main condition which required for alpha-beta pruning is:
- **$\alpha >= \beta$**

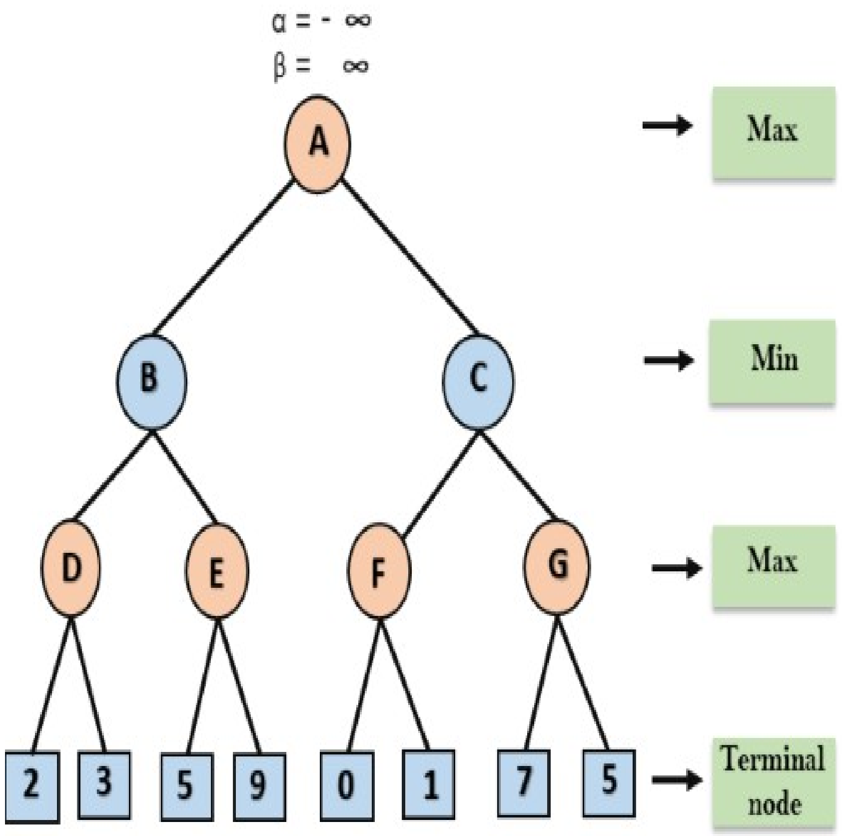
Key points about alpha-beta pruning:

- The **Max player will only update the value of alpha.**
- The **Min player will only update the value of beta.**
- **While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.**
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

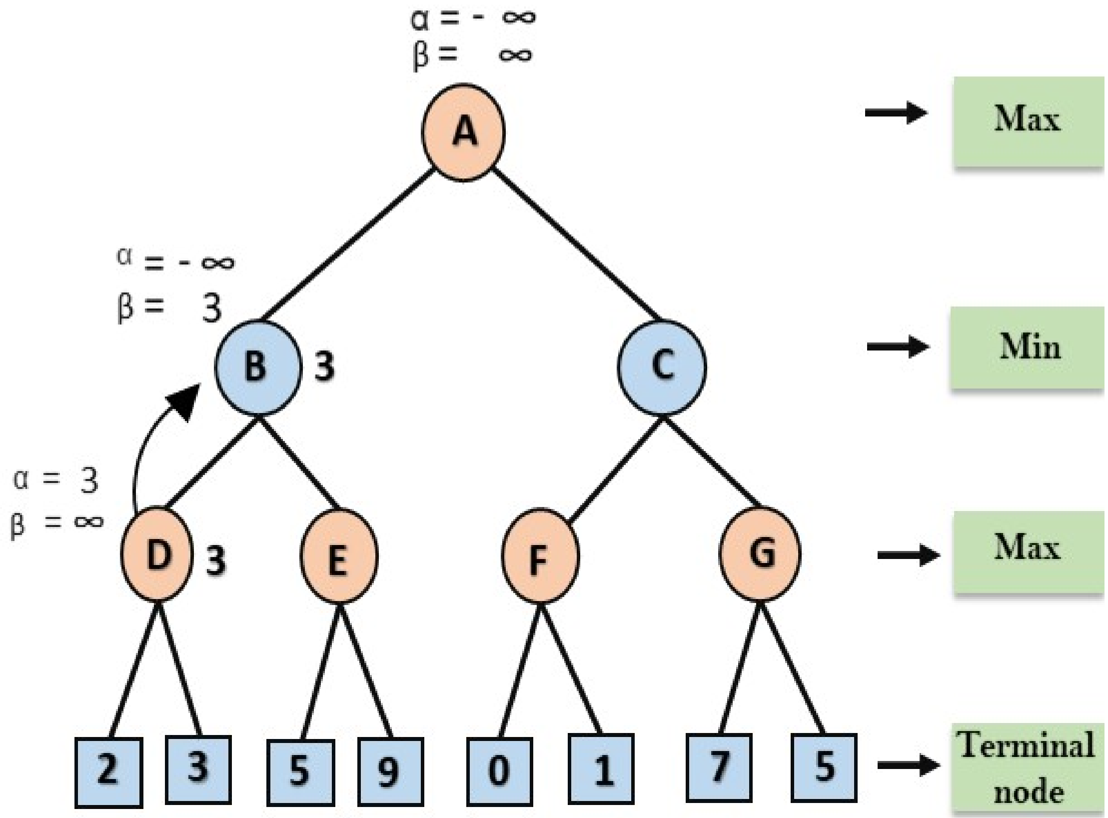
- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

- Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



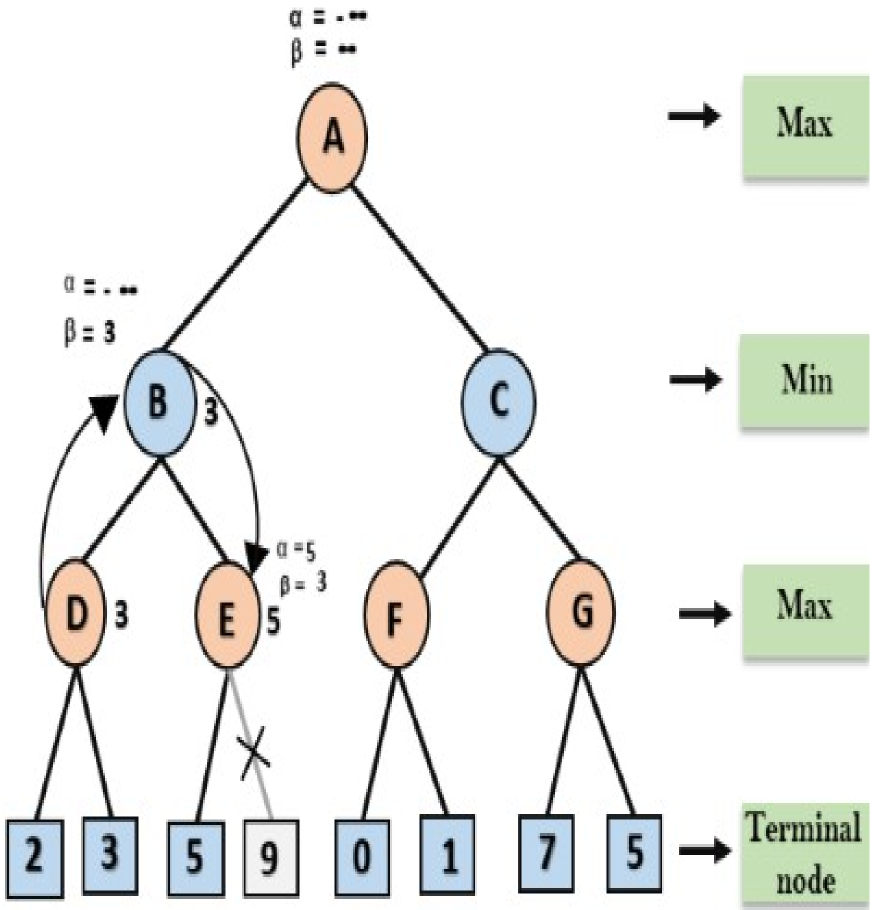
- **Step 2:** At Node D, the value of α will be calculated as its turn for Max.
- The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

- **Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



- In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

- **Step 4:** At node E, Max will take its turn, and the value of alpha will change.
- The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

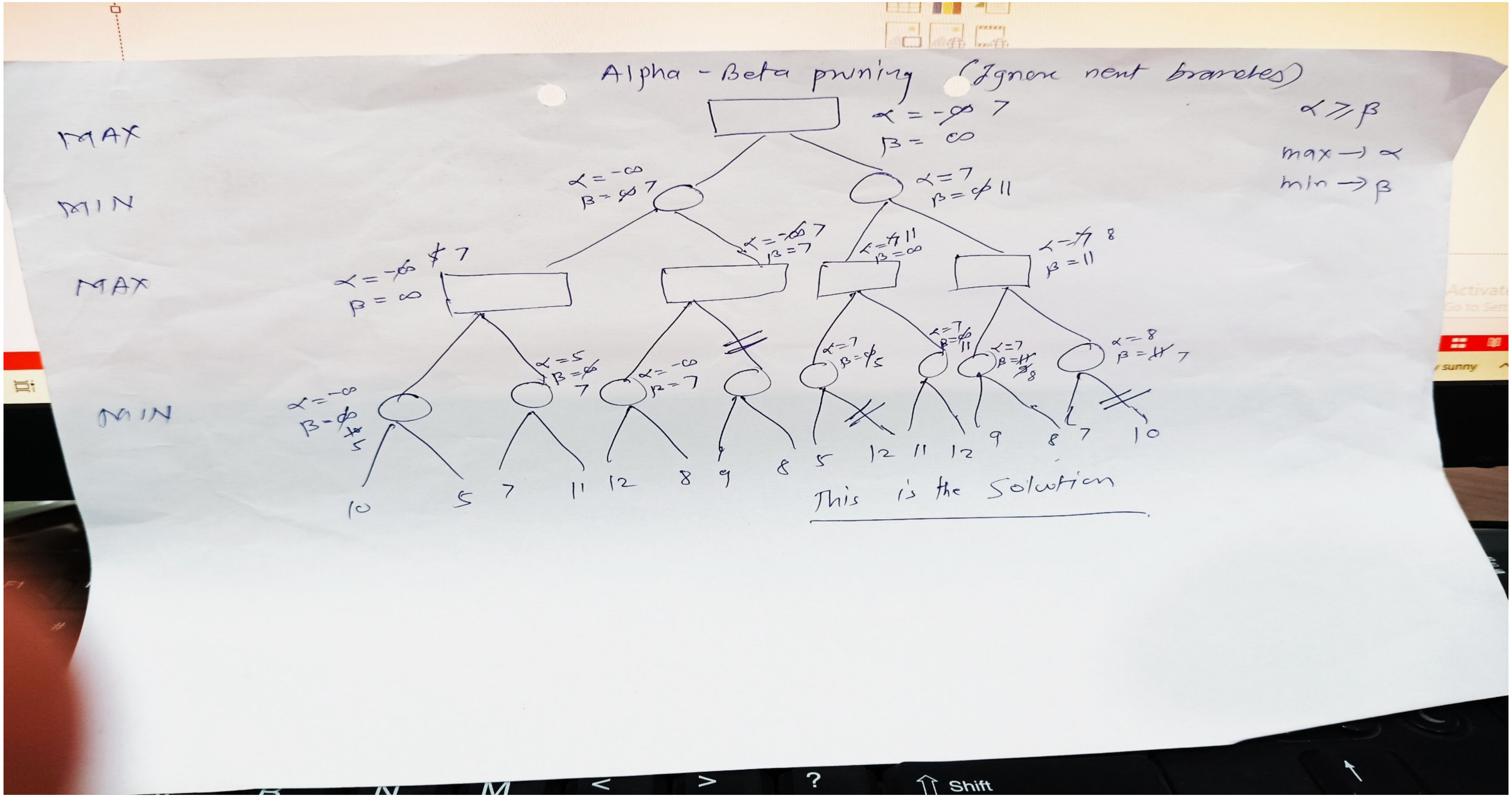


- **Step 5:** At next step, algorithm again backtrack the tree, from node B to node A.
- At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

- At node C, α=3 and β= +∞, and the same values will be passed on to node F.

- **Step 6:** At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3,

Example 2 -



Alpha - Beta pruning (Ignore next branches)
α ≥ β
max → α
min → β
MAX
MIN
MAX
MIN
10
5
7
11
12
8
9
8
5
12
11
12
9
8
7
10
This is the Solution

A* Algorithm

- A* search is the most commonly known form of best-first search.
- It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides optimal result faster.
- A* 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.

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

Estimated cost of the cheapest solution.

Cost to reach node n from start state.

Cost to reach from node n to goal node

Tutorial No.2

- What is Search Problem? How it is represented using State space model?
- Describe Uninformed Search Algorithms
- Explain Alpha- Beta Pruning in Mini-max Algorithm.
- What is A* Algorithm ? Explain working with example.