

# Game Playing

Mr. Feb

We will discuss about basic two players game with no uncertainty.

Like in cards you do not know which cards the opponent is having...

We will discuss the case where we know opponent will take a move from n-numbers of moves.. which are known to you.

Like Tic Tac Toe or chess game

# MiniMax Algorithm

A game can be formally defined as a kind of search problem with the following components:

- **The initial state**, which includes the board position and an indication of whose move it is.
- **A set of operators**, which define the legal moves that a player can make.
- **A terminal test**, which determines when the game is over. States where the game has ended are called **terminal states**.
- **A utility function** (also called a **payoff function**), which gives a numeric value for the outcome of a game. In chess, the outcome is a win, loss, or draw, which we can represent by the values  $+1$ ,  $-1$ , or  $0$ . Some games have a wider variety of possible outcomes; for example, the payoffs in backgammon range from  $+192$  to  $-192$ .
- Backtracking/recursive Algorithm
- Depth first search algorithm is used for exploration of complete game tree



# MiniMax Algorithm

- Depth first search algorithm is used for exploration of complete game tree

Mr. Feb



# MiniMax Algorithm

- Backtracking/recursive Algorithm

Mr. Feb

MAX (X)

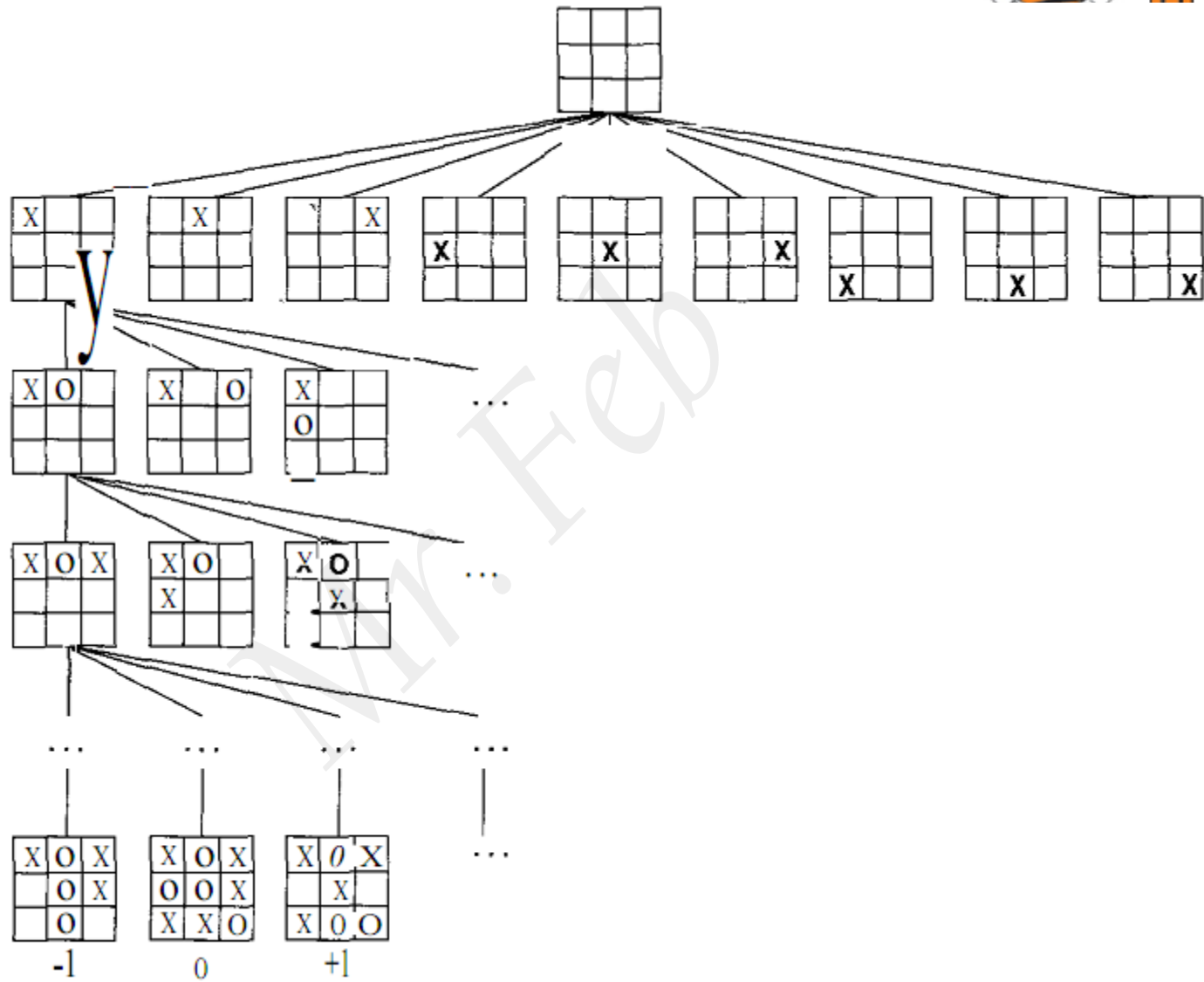
MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility



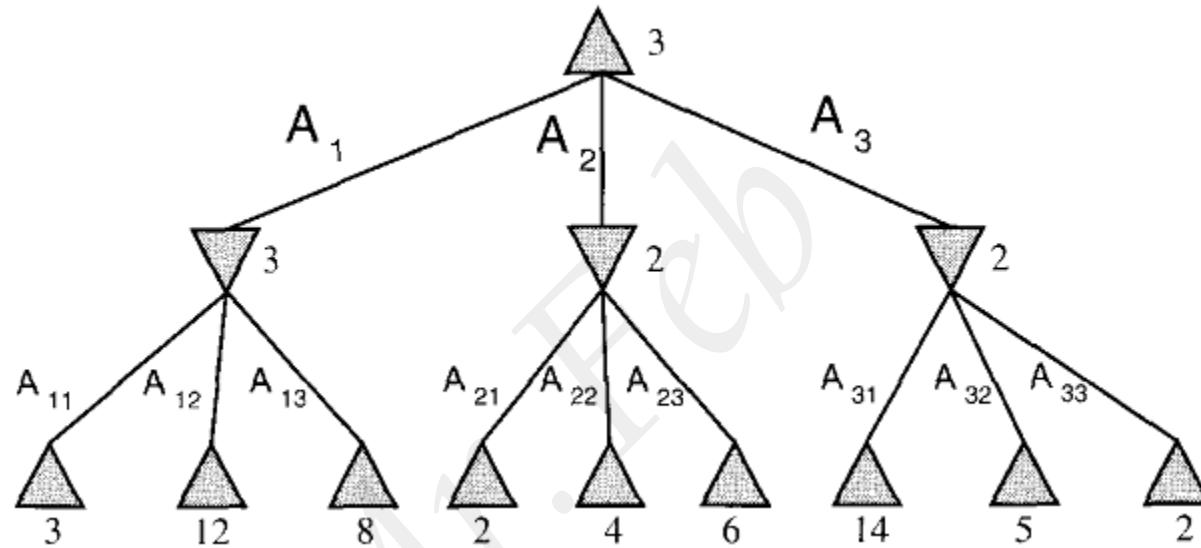
# Process of MINIMAX Algorithm

The **minimax** algorithm is designed to determine the optimal strategy for MAX, and thus to decide what the best first move is. The algorithm consists of five steps:

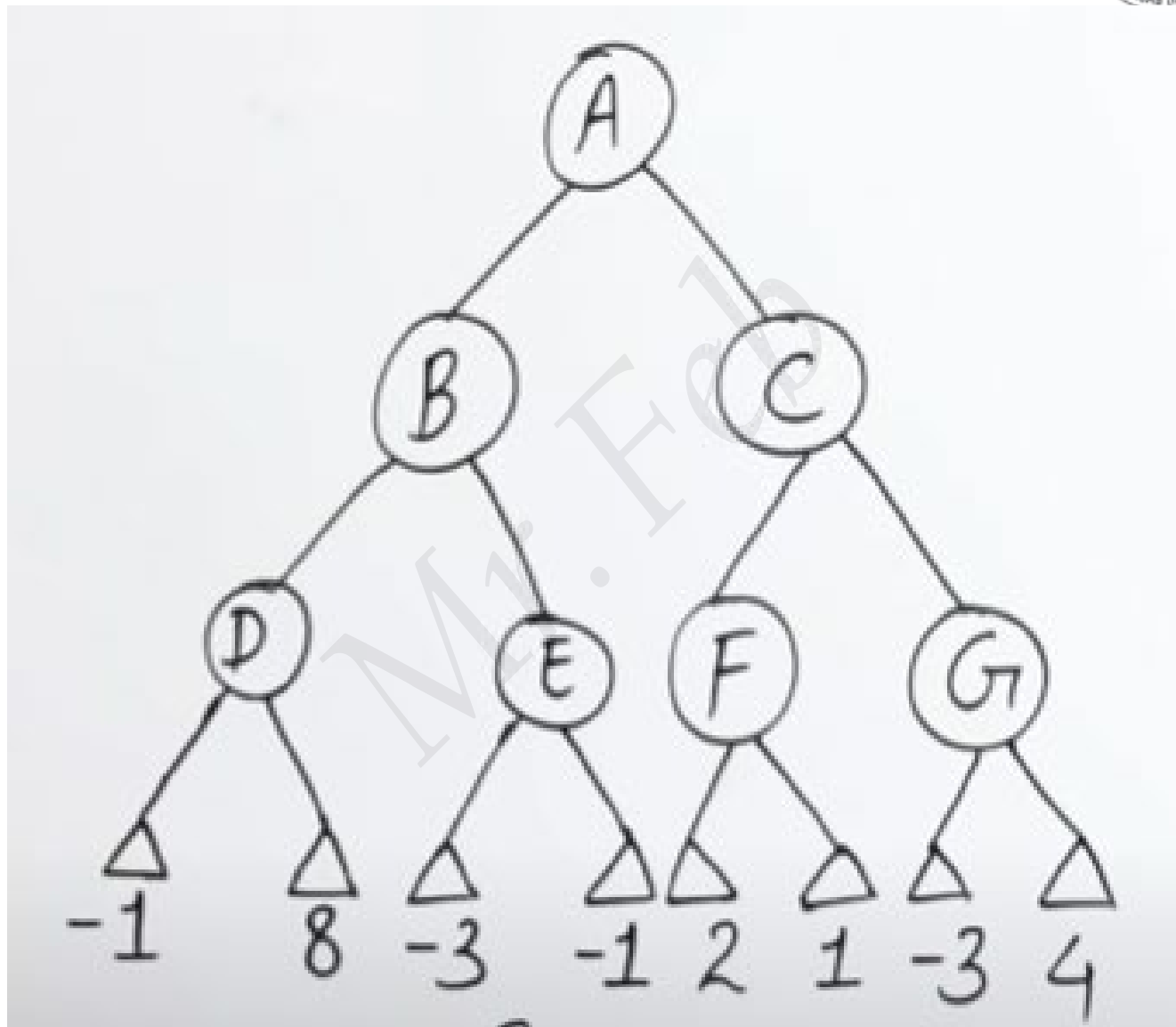
- Generate the whole game tree, all the way down to the terminal states.
- Apply the utility function to each terminal state to get its value.
- Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree. Consider the leftmost three leaf nodes in Figure 5.2. In the V node above it, MIN has the option to move, and the best MIN can do is choose  $A_{11}$ , which leads to the minimal outcome, 3. Thus, even though the utility function is not immediately applicable to this V node, we can assign it the utility value 3, under the assumption that MIN will do the right thing. By similar reasoning, the other two V nodes are assigned the utility value 2.
- Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Eventually, the backed-up values reach the top of the tree; at that point, MAX chooses the move that leads to the highest value. In the topmost A node of Figure 5.2, MAX has a choice of three moves that will lead to states with utility 3, 2, and 2, respectively. Thus, MAX's best opening move is  $A_1$ . This is called the **minimax decision**, because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it.

MAX

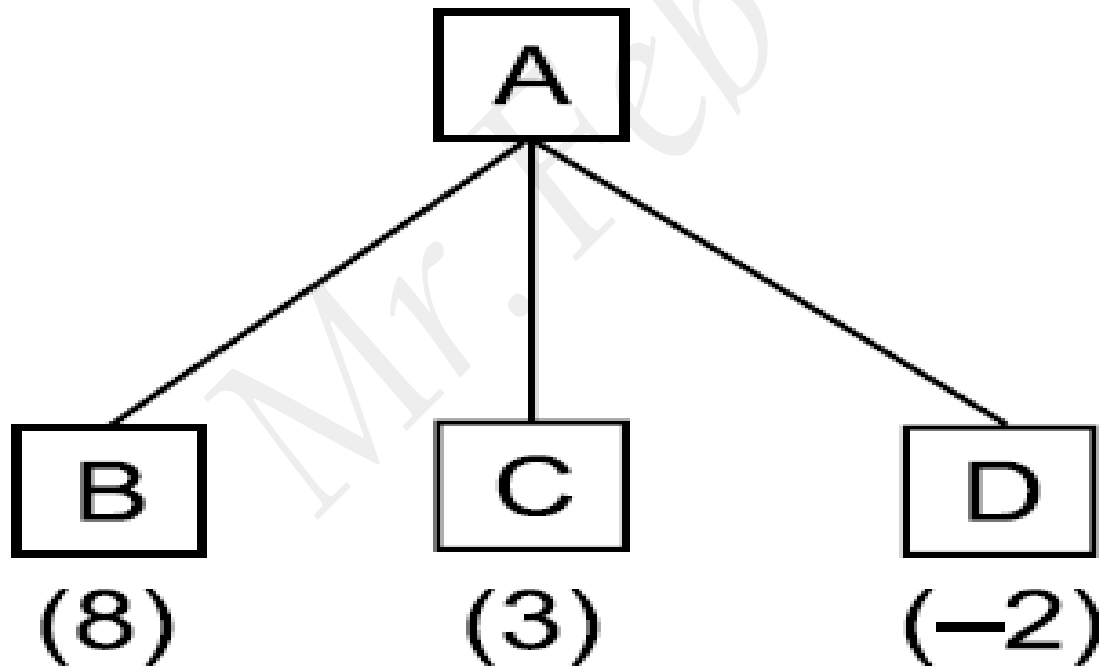
MIN



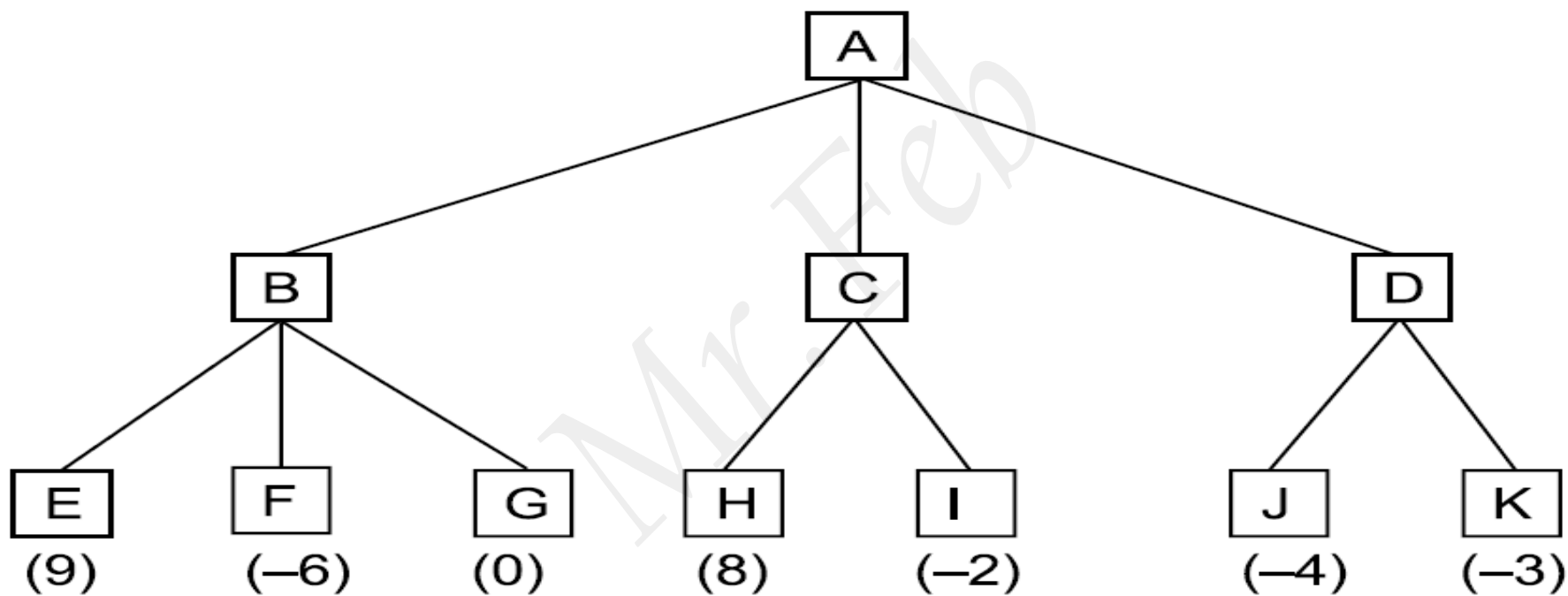




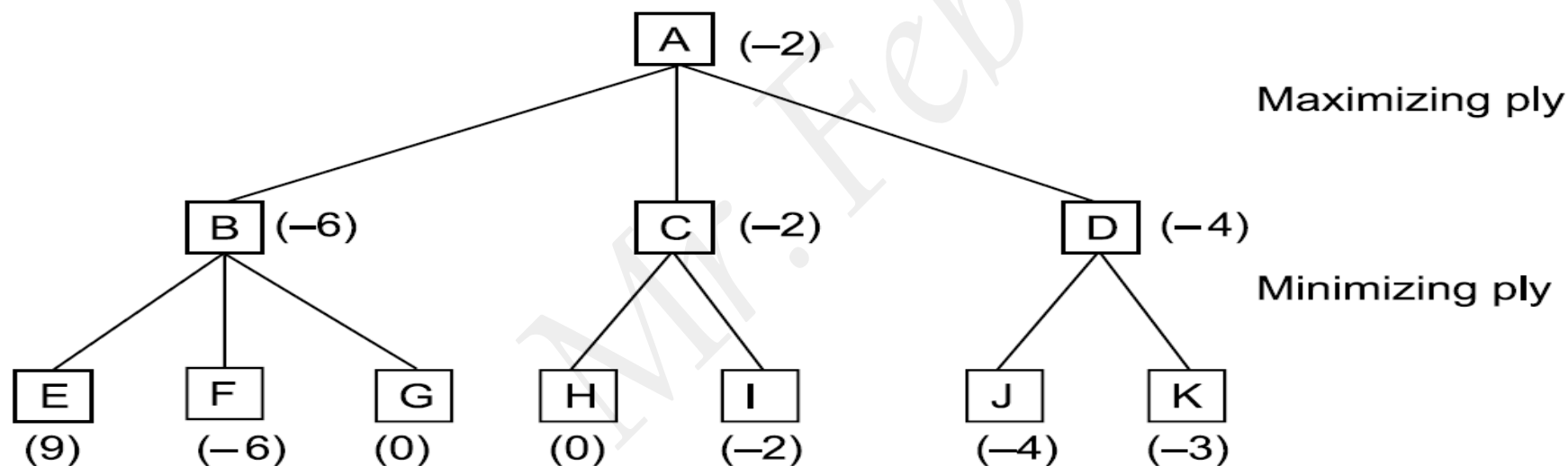
# One – Ply Search



# Two – Ply Search

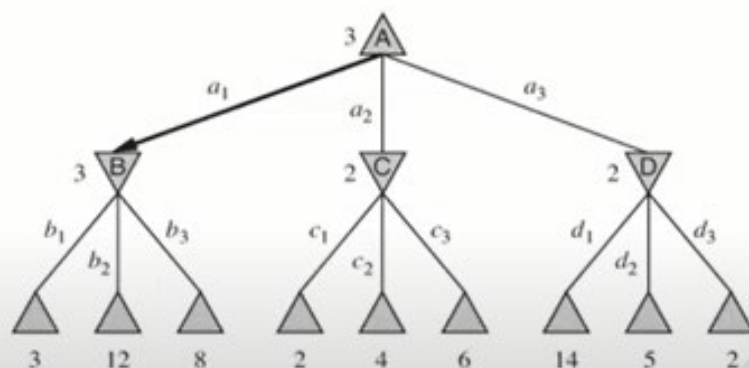


# Backing Up the Values of a Two – Ply Search



MAX


MIN



**function** MINIMAX-DECISION(*state*) **returns** an action  
**return**  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$

**function** MAX-VALUE(*state*) **returns** a utility value  
**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
 $v \leftarrow -\infty$   
**for each** *a* **in** ACTIONS(*state*) **do**  
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
**return** *v*

**function** MIN-VALUE(*state*) **returns** a utility value  
**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
 $v \leftarrow \infty$   
**for each** *a* **in** ACTIONS(*state*) **do**  
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
**return** *v*



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

**function** MINIMAX-DECISION(*state*) *returns an action*  
**return**  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$

---

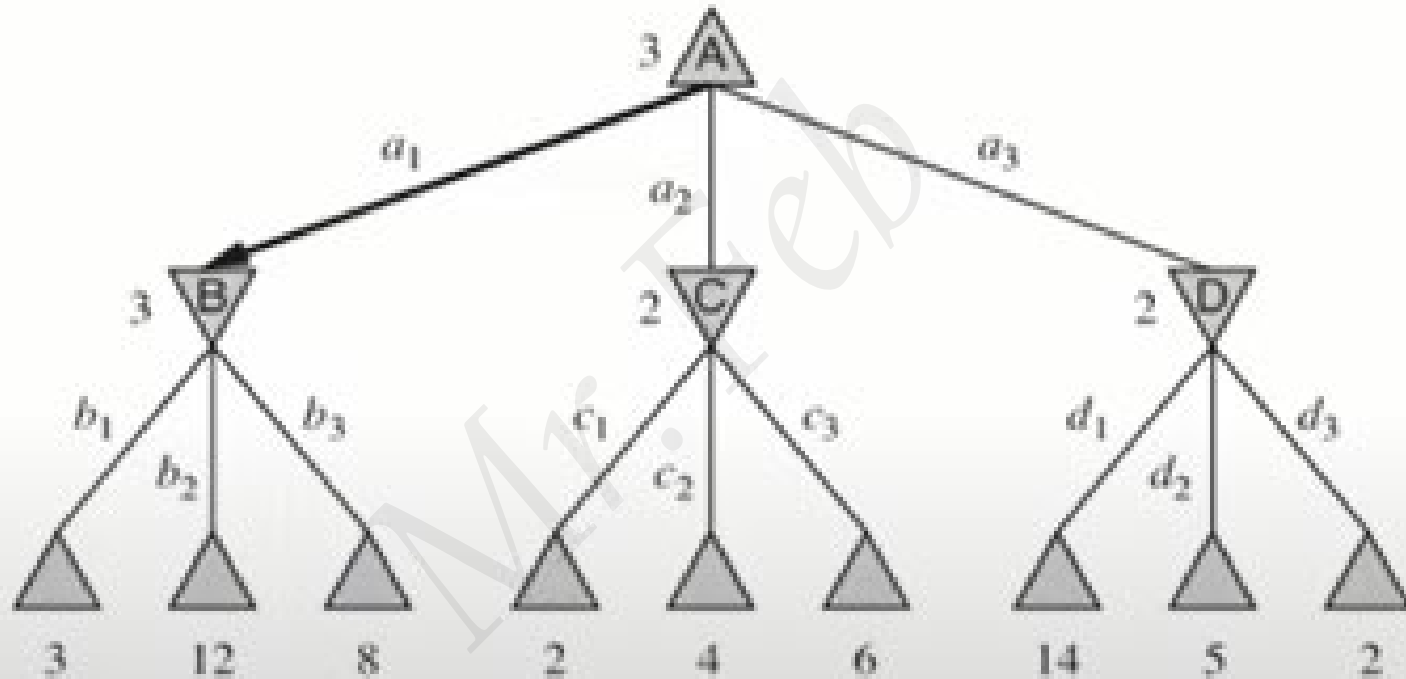
**function** MAX-VALUE(*state*) *returns a utility value*  
**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
 $v \leftarrow -\infty$   
**for each** *a* **in** ACTIONS(*state*) **do**  
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
**return** *v*

---

**function** MIN-VALUE(*state*) *returns a utility value*  
**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
 $v \leftarrow \infty$   
**for each** *a* **in** ACTIONS(*state*) **do**  
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
**return** *v*

MAX

MIN





Mr. Feb



General algorithm applied on game tree for making decision of win/lose is \_\_\_\_\_.

- |           |                             |
|-----------|-----------------------------|
| <b>a.</b> | DFS/BFS Search Algorithms   |
| <b>b.</b> | Heuristic Search Algorithms |
| <b>c.</b> | Greedy Search Algorithms    |
| <b>d.</b> | MIN/MAX Algorithms          |

Which search is similar to minimax search?

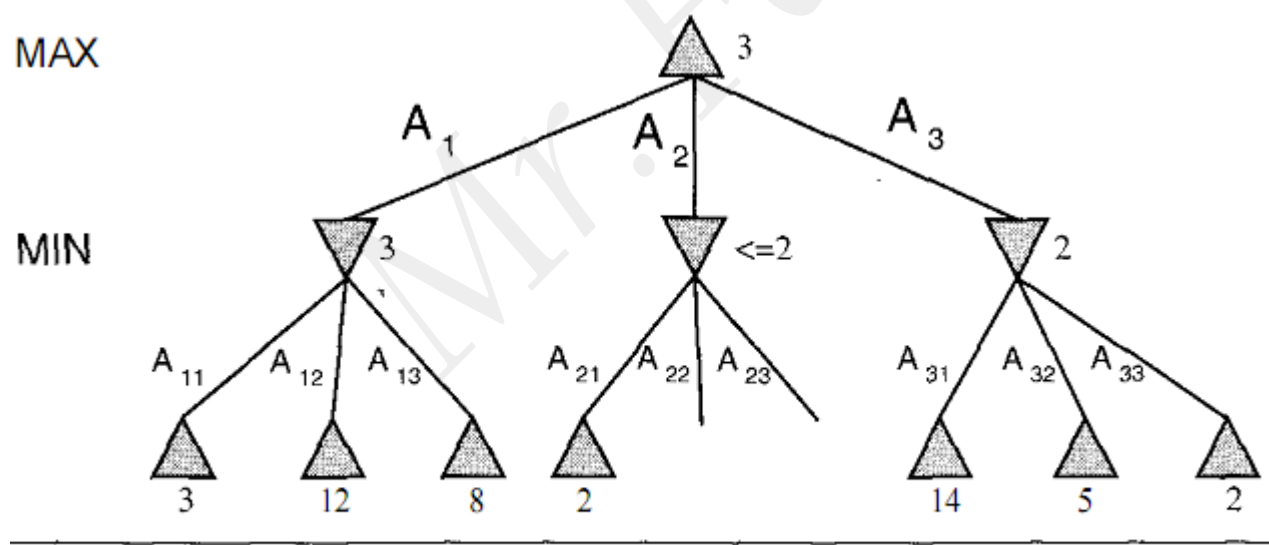
- |           |                      |
|-----------|----------------------|
| <b>a.</b> | Hill-climbing search |
| <b>b.</b> | Depth-first search   |
| <b>c.</b> | Breadth-first search |
| <b>d.</b> | All of the mentioned |



Mr. Feb

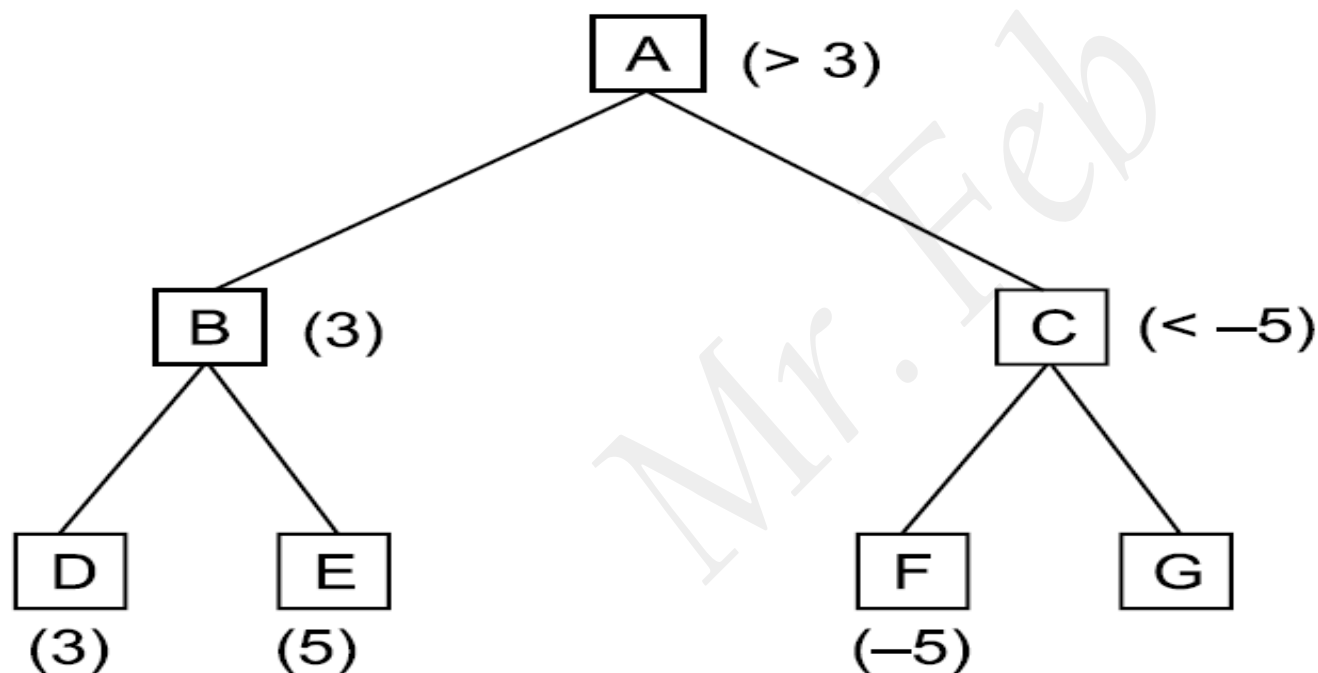
# ALPHA-BETA PRUNING

Fortunately, it is possible to compute the correct minimax decision without looking at every node in the search tree. The process of eliminating a branch of the search tree from consideration without examining it is called **pruning** the search tree. The particular technique we will examine is called **alpha-beta pruning**. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.



The two-ply game tree as generated by alpha-beta.

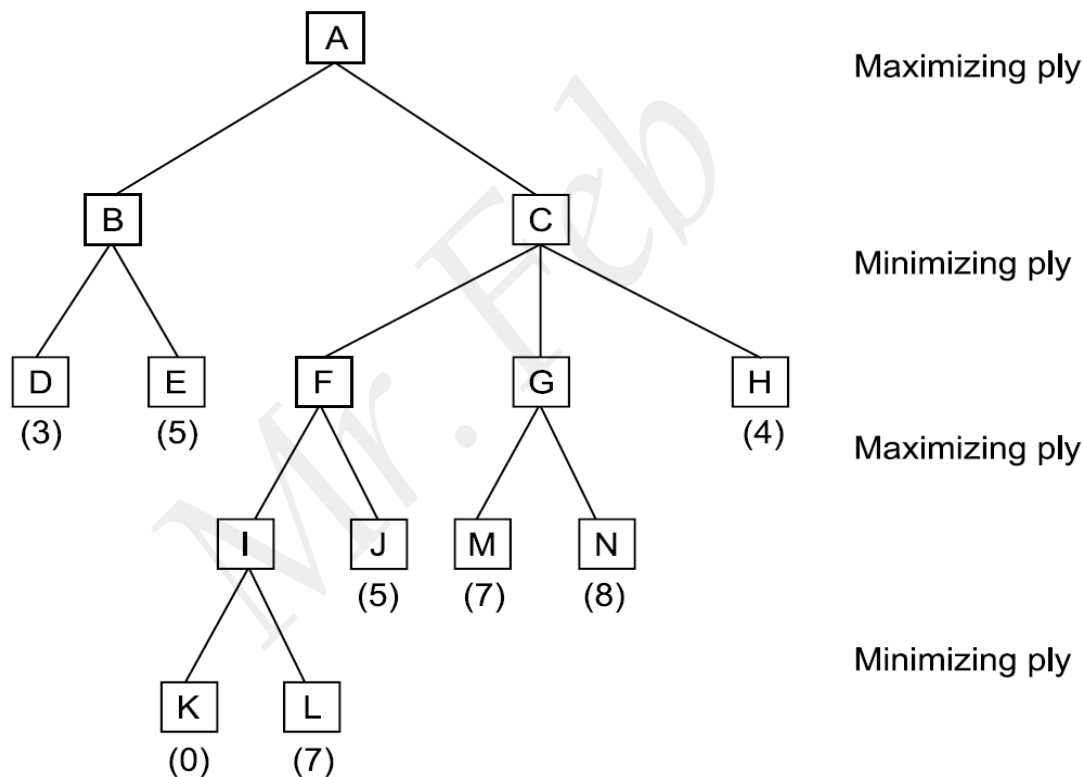
# Alpha – Beta Pruning



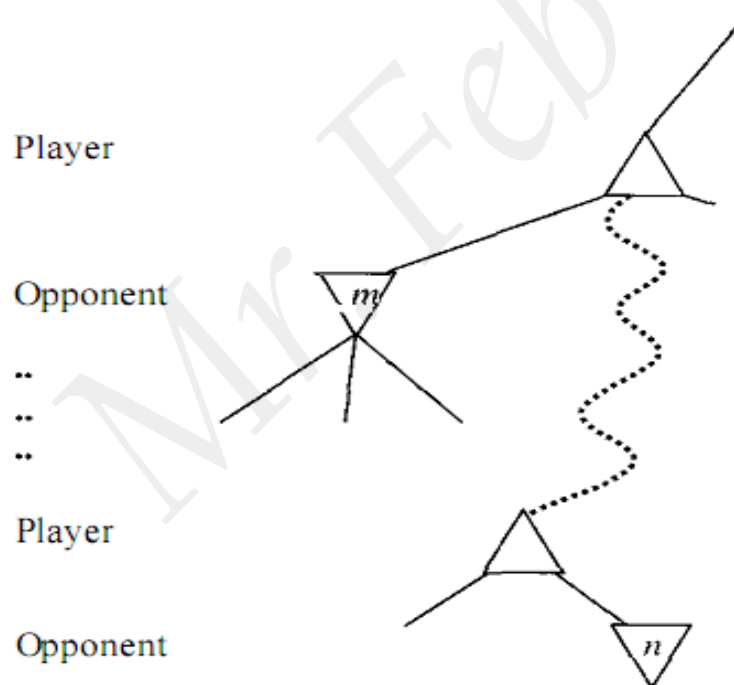
Maximizing ply

Minimizing ply

# Alpha and Beta Cutoffs



The general principle is this. Consider a node  $n$  somewhere in the tree (see Figure 5.7), such that Player has a choice of moving to that node. If Player has a better choice  $m$  either at the parent node of  $n$ , or at any choice point further up, then  $n$  will never be reached in actual play.



**Figure 5.7** Alpha-beta pruning: the general case. If  $m$  is better than  $n$  for Player, we will never get to  $n$  in play.

Which value is assigned to alpha and beta in the alpha-beta pruning?

- a) Alpha = max
- b) Beta = min
- c) Beta = max
- d) Both Alpha = max & Beta = min

Mr. Feb



**function** MAX-VALUE(*stategame*,  $\alpha$ ,  $\beta$ ) **returns** the minimax value of *state*

**inputs:** *state*, current state in game

*game*, game description

$\alpha$ , the best score for MAX along the path to *state*

$\beta$ , the best score for MIN along the path to *state*

**if** CUTOFF-TEST(*state*) **then return** EVAL(*state*)

**for each** *s* **in** SUCCESSORS(*state*) **do**

$\alpha \leftarrow \text{MAX}(\alpha, \text{MIN-VALUE}(s, \text{game}, \alpha, \beta))$

**if**  $\alpha \geq \beta$  **then return**  $\beta$

**end**

**return**  $\alpha$

---

**function** MIN-VALUE(*state*, *game*,  $\alpha$ ,  $\beta$ ) **returns** the minimax value of *state*

**if** CUTOFF-TEST(*state*) **then return** EVAL(*state*)

**for each** *s* **in** SUCCESSORS(*state*) **do**

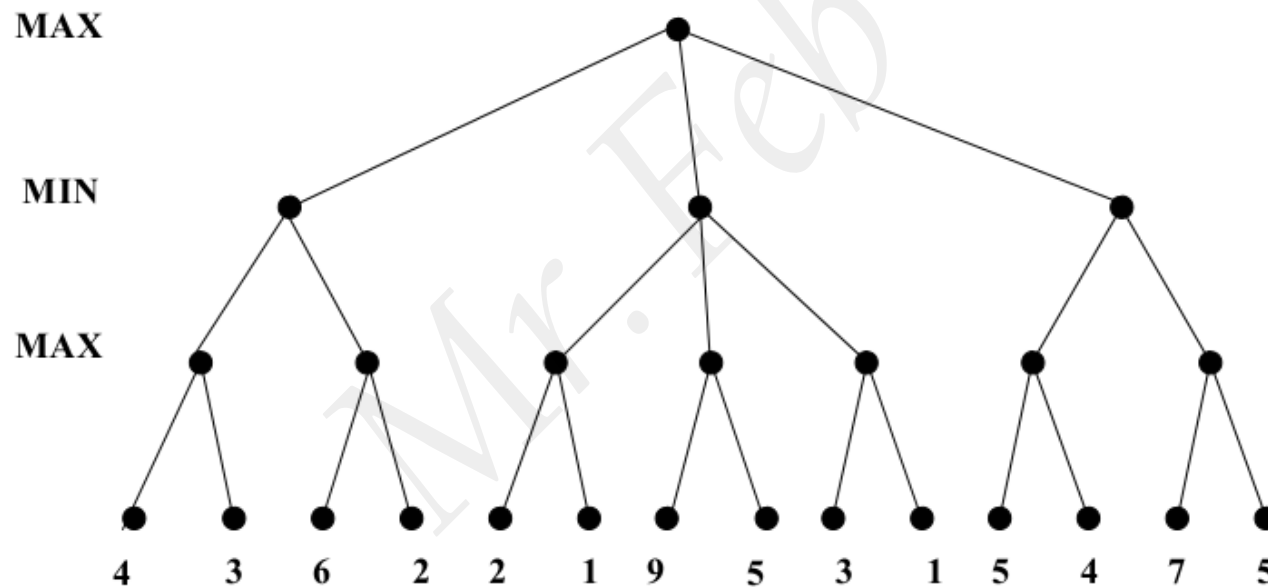
$\beta \leftarrow \text{MIN}(\beta, \text{MAX-VALUE}(s, \text{game}, \alpha, \beta))$

**if**  $\beta < \alpha$  **then return**  $\alpha$

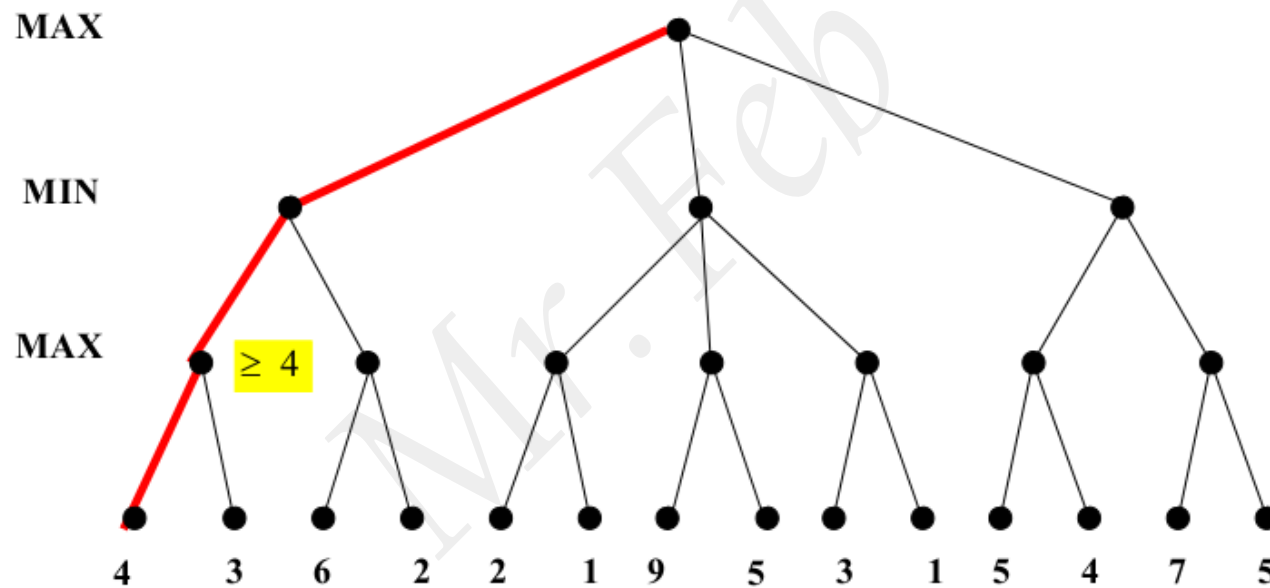
**end**

**return**  $\beta$

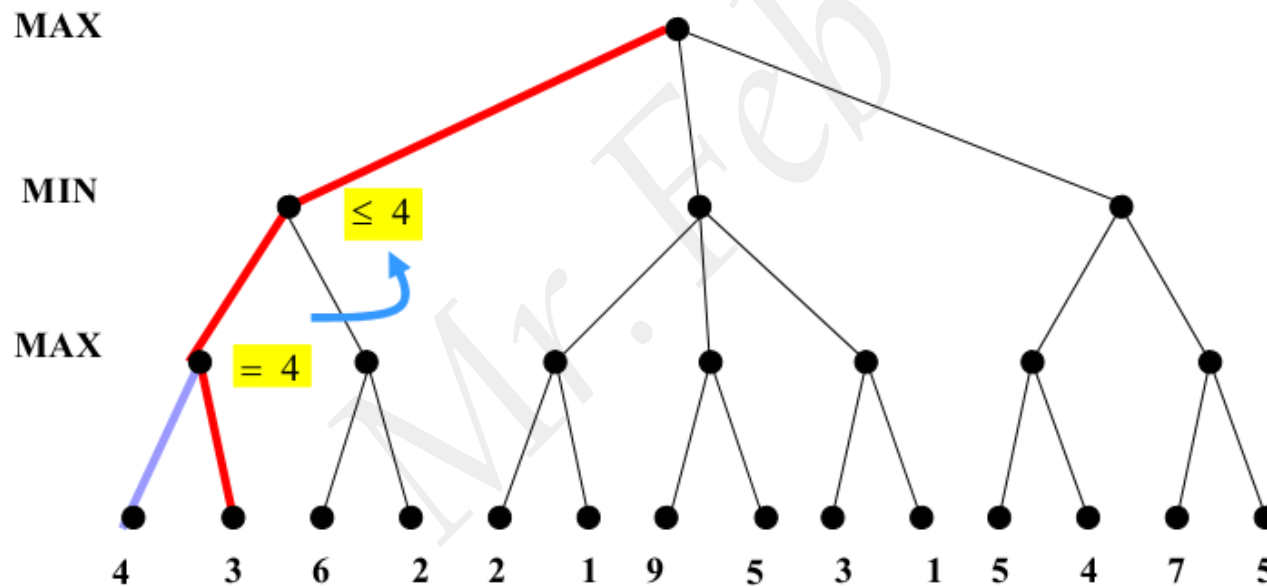
## Alpha beta pruning. Example



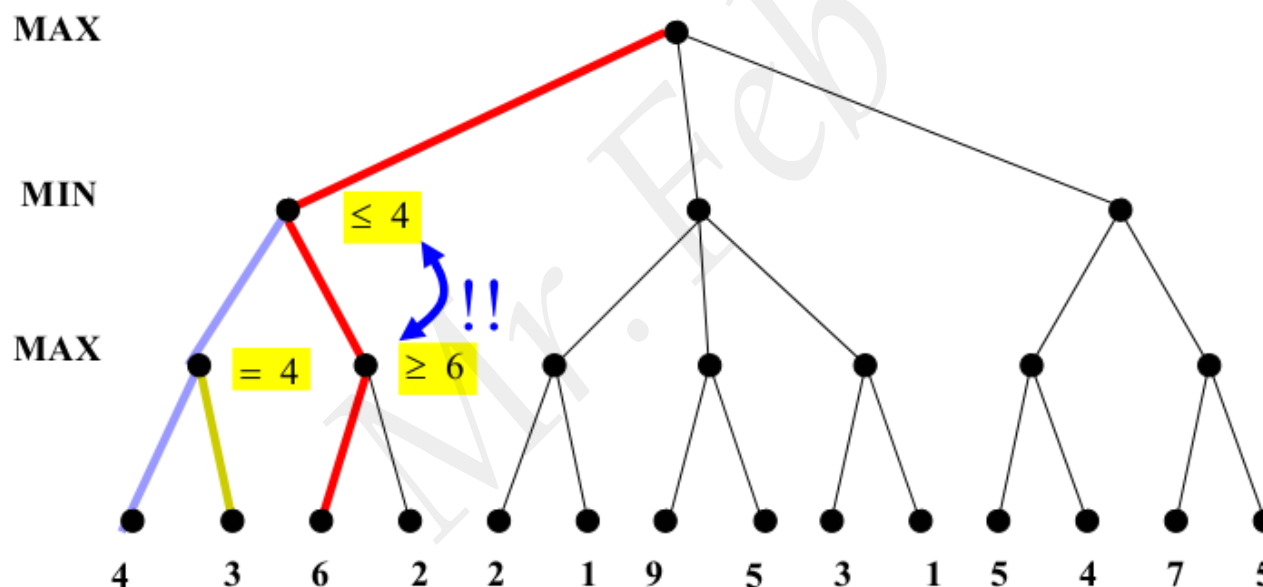
## Alpha beta pruning. Example



## Alpha beta pruning. Example

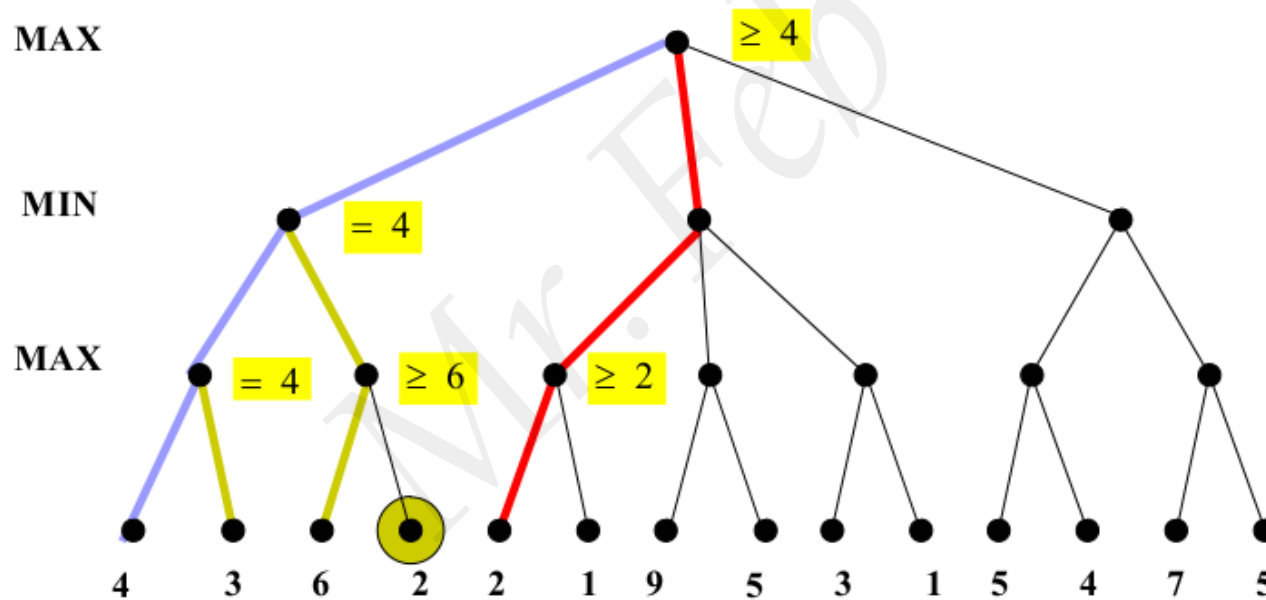


## Alpha beta pruning. Example

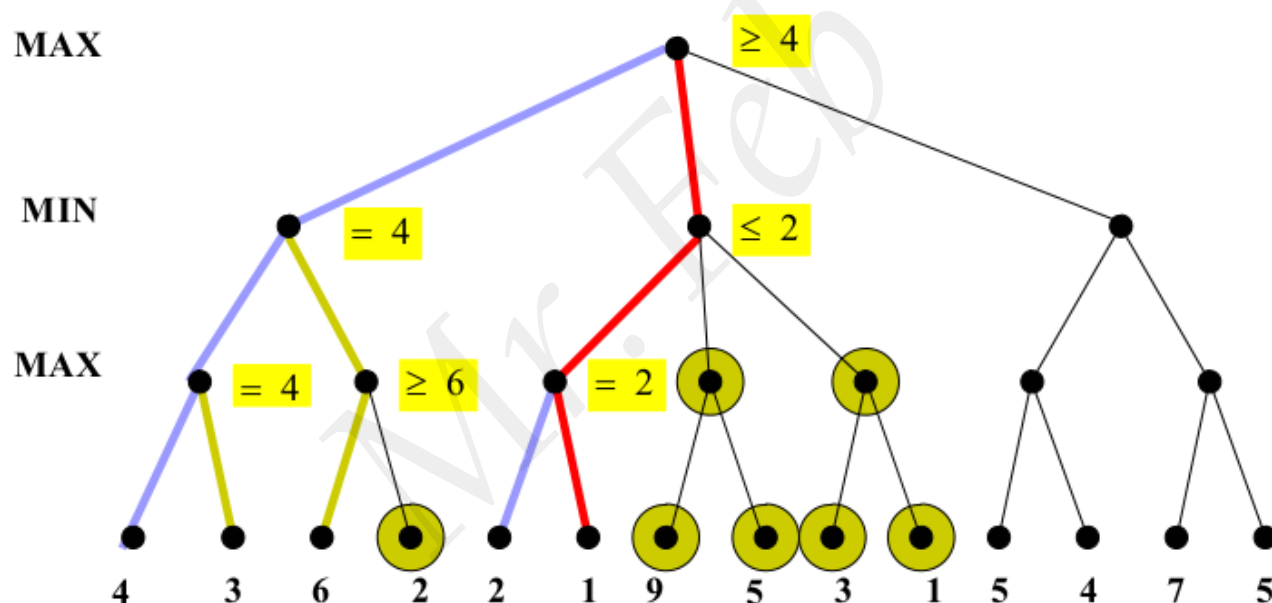




## Alpha beta pruning. Example

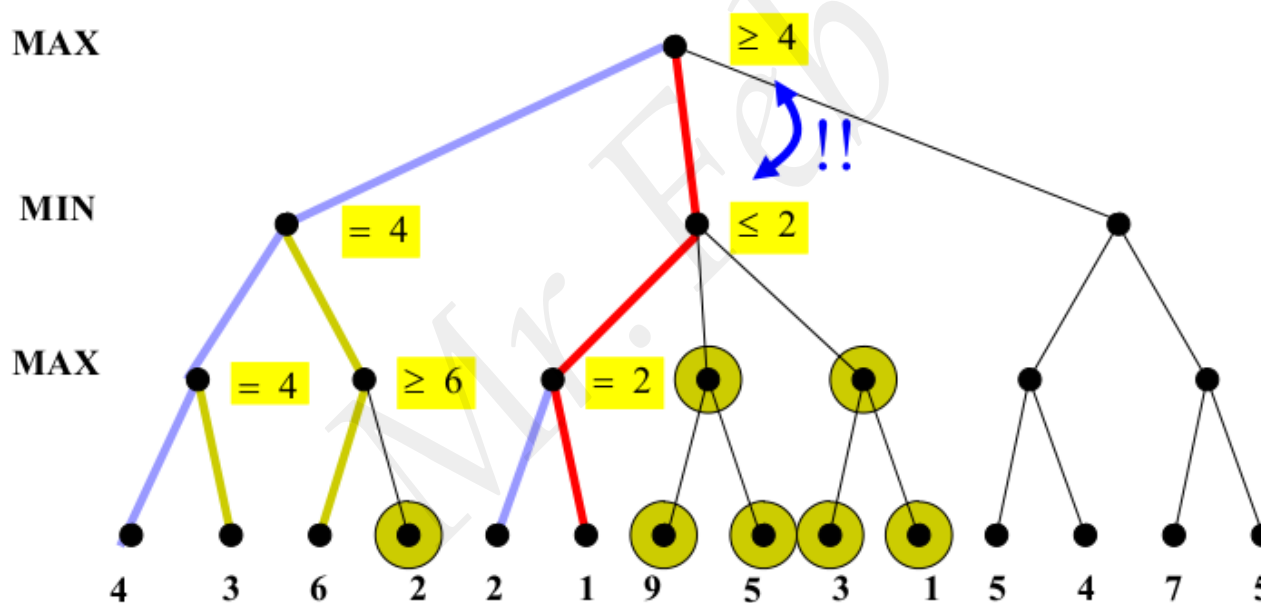


## Alpha beta pruning. Example

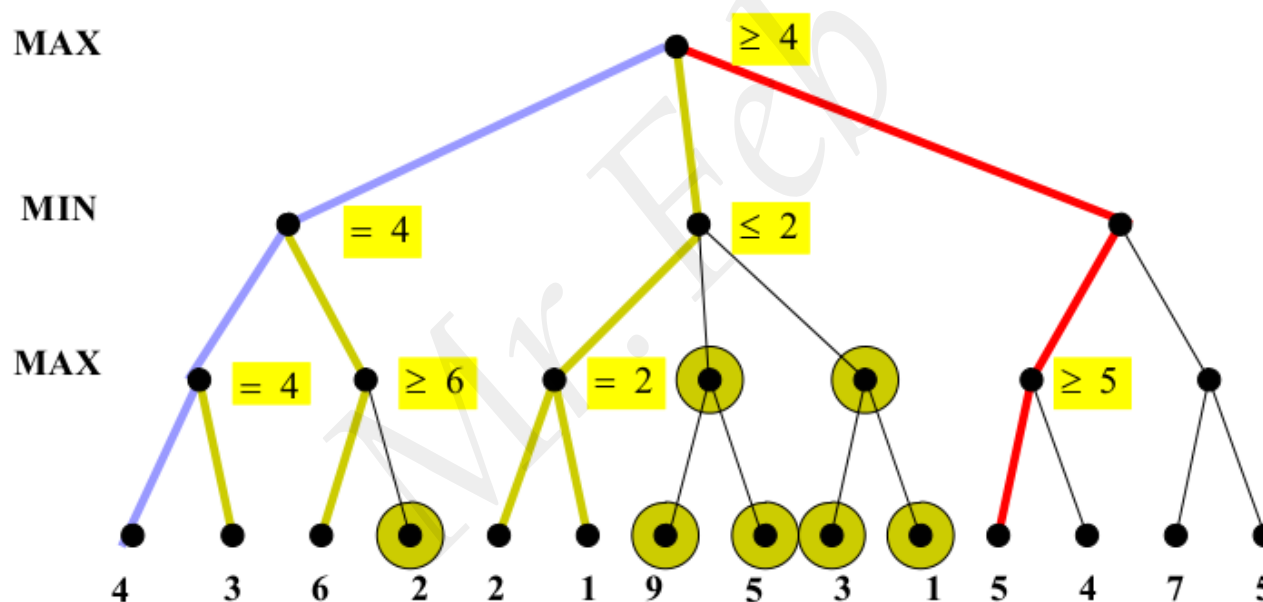




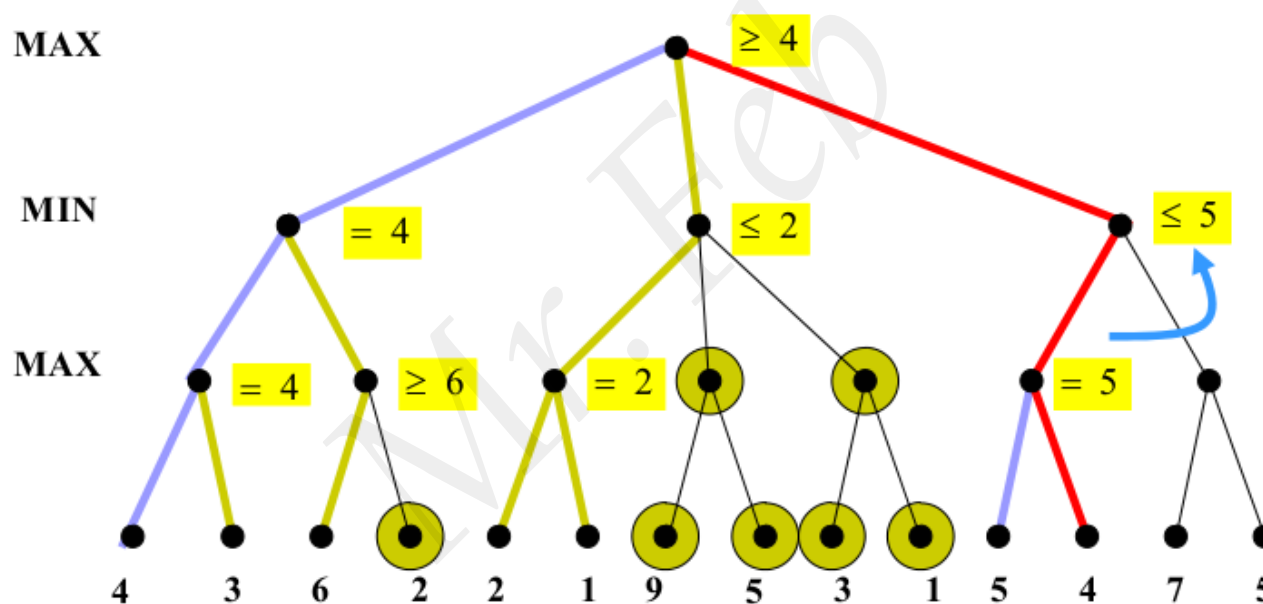
## Alpha beta pruning. Example



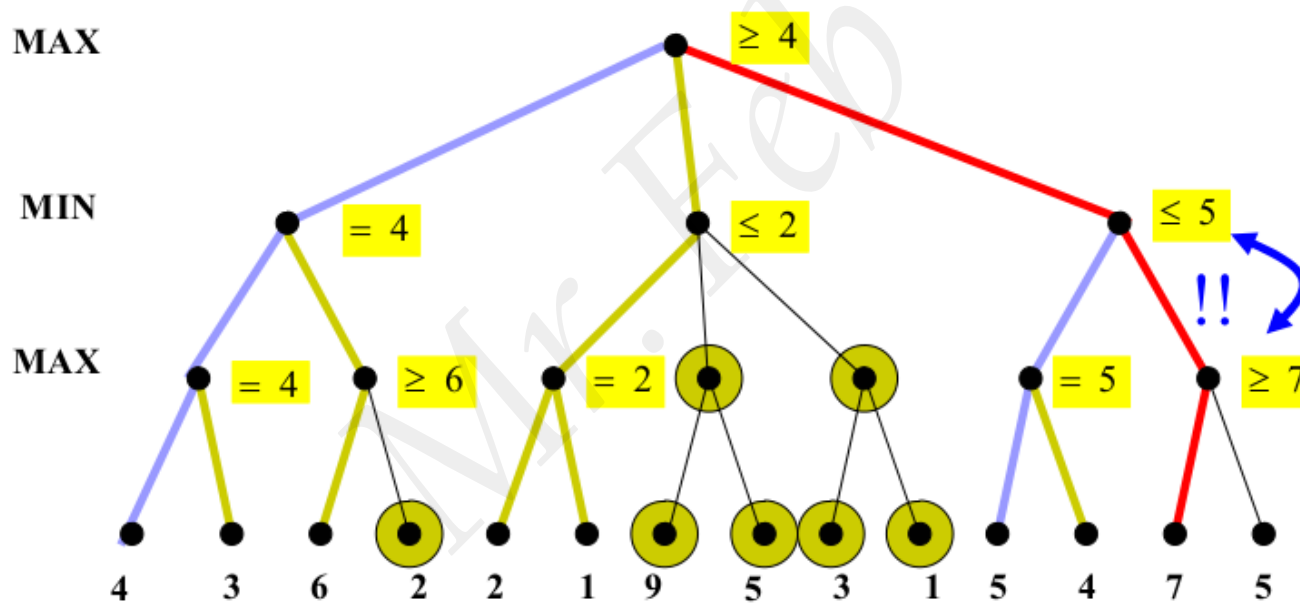
## Alpha beta pruning. Example



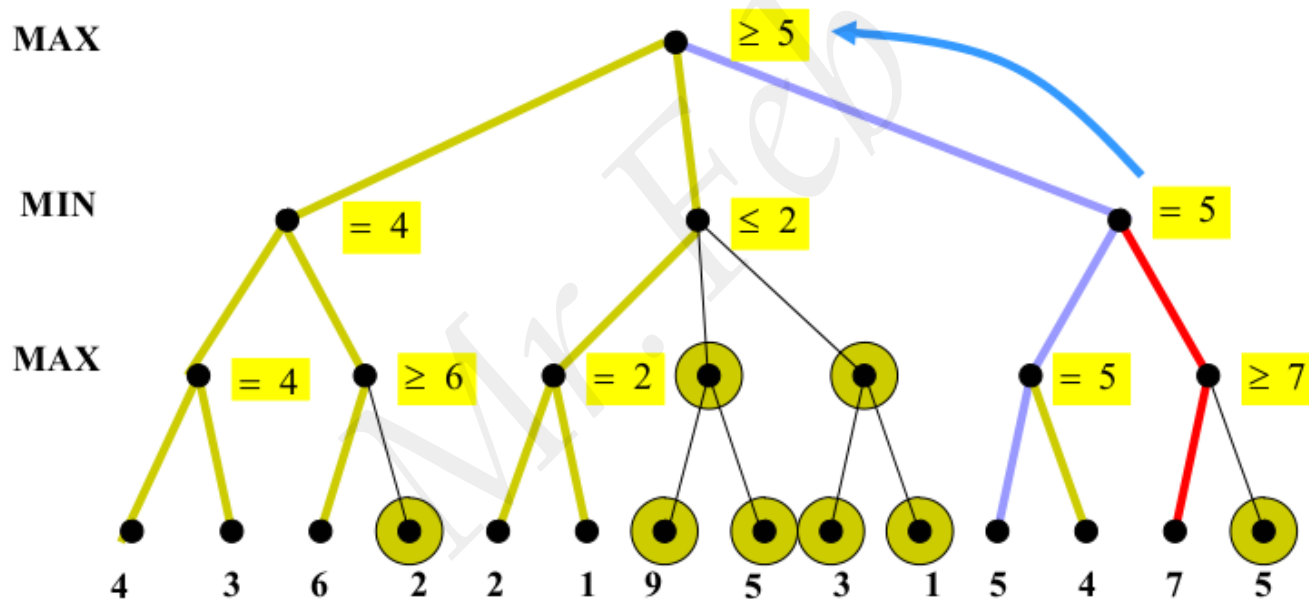
## Alpha beta pruning. Example



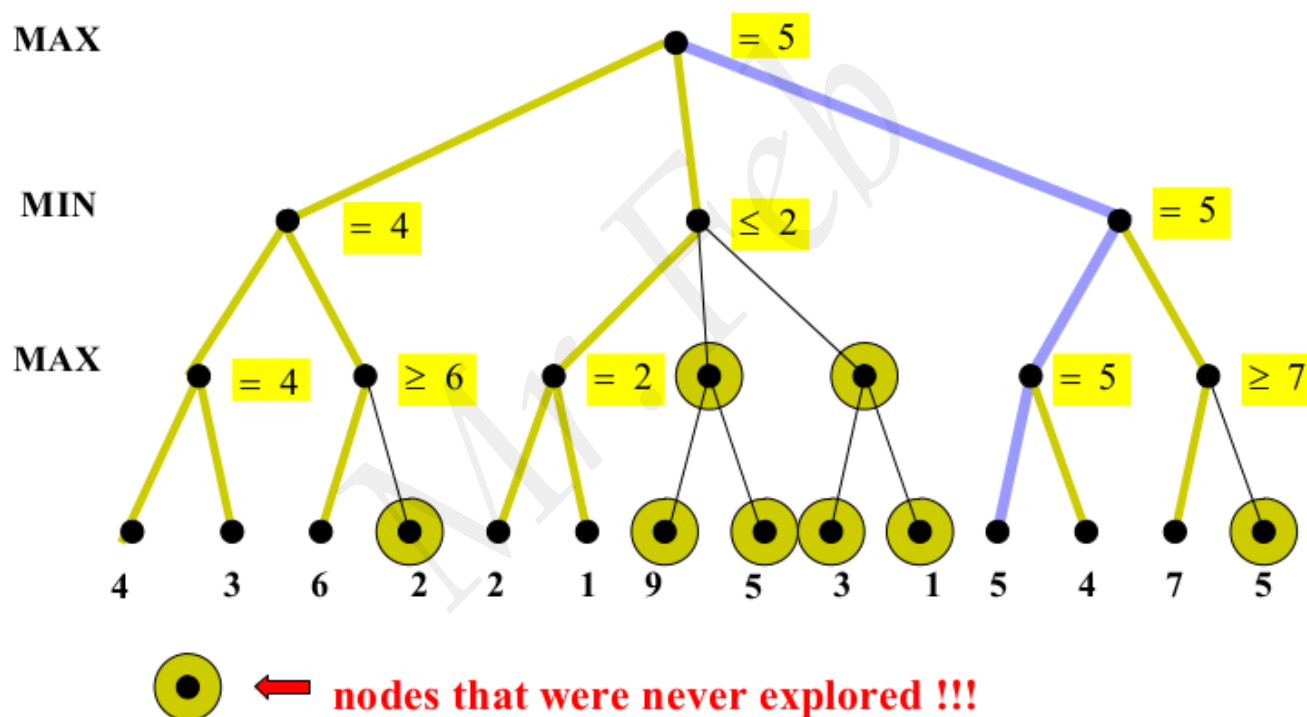
## Alpha beta pruning. Example



## Alpha beta pruning. Example



## Alpha beta pruning. Example

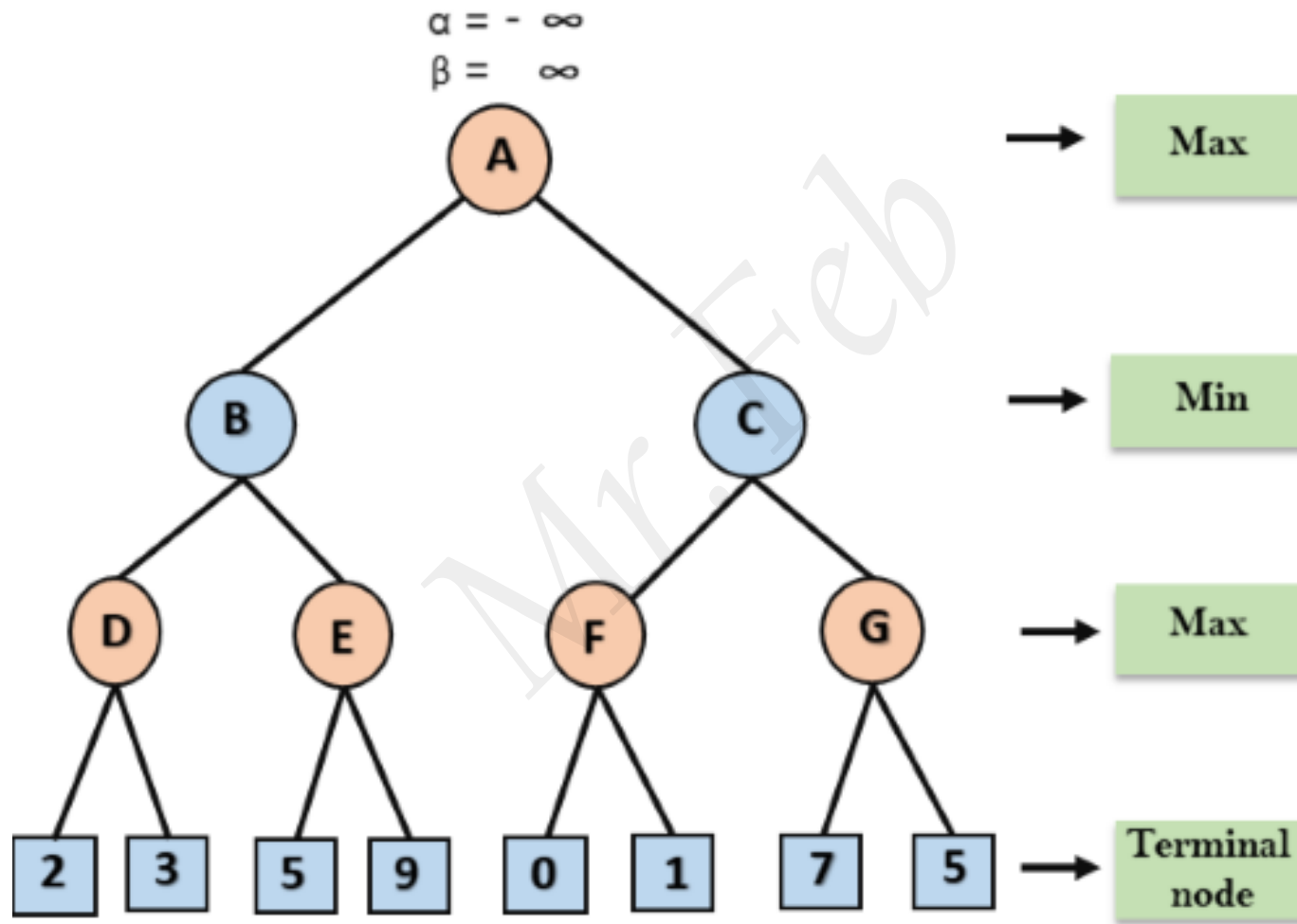


Where does the values of alpha-beta search get updated?

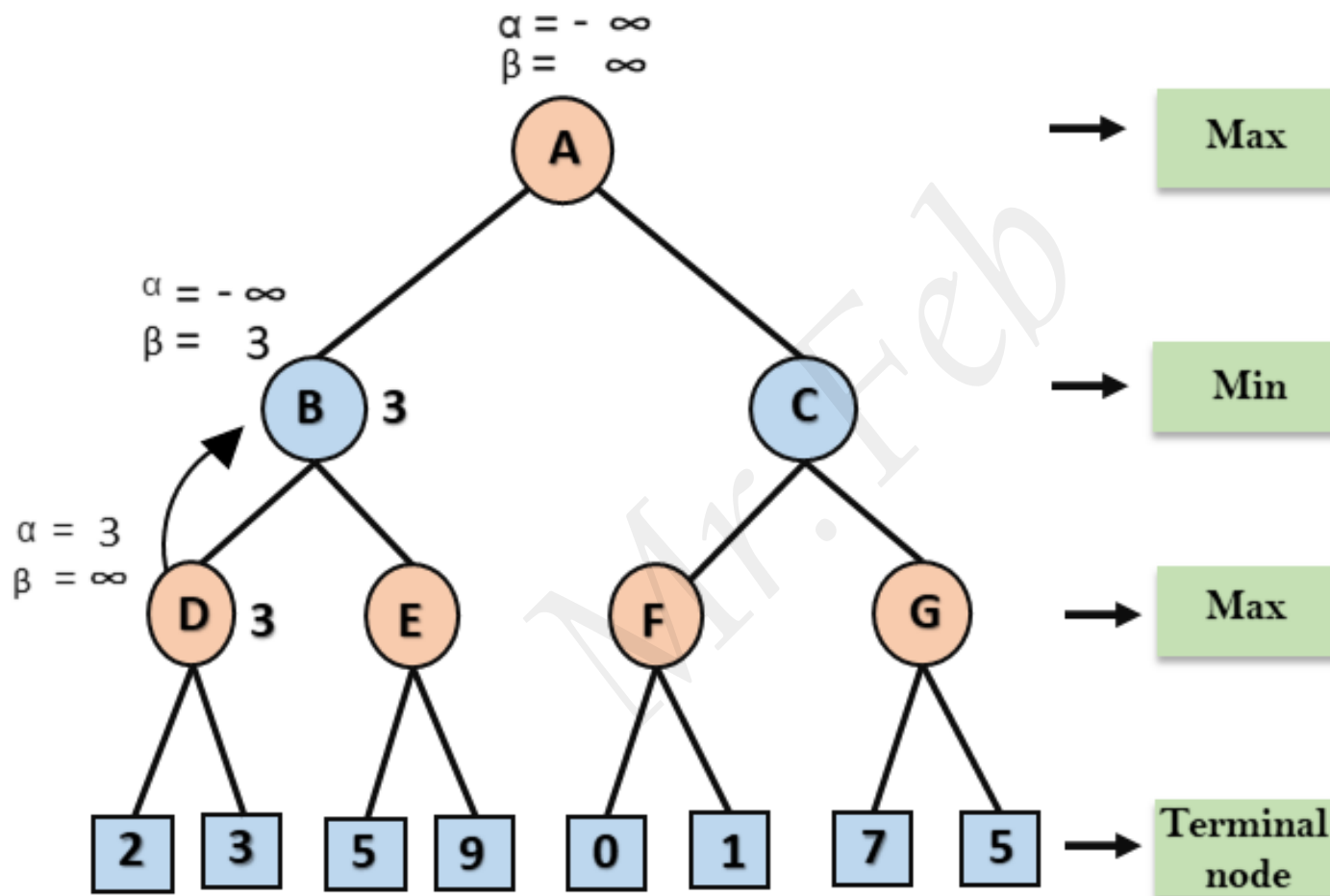
- a) Along the path of search
- b) Initial state itself
- c) At the end
- d) None of the mentioned

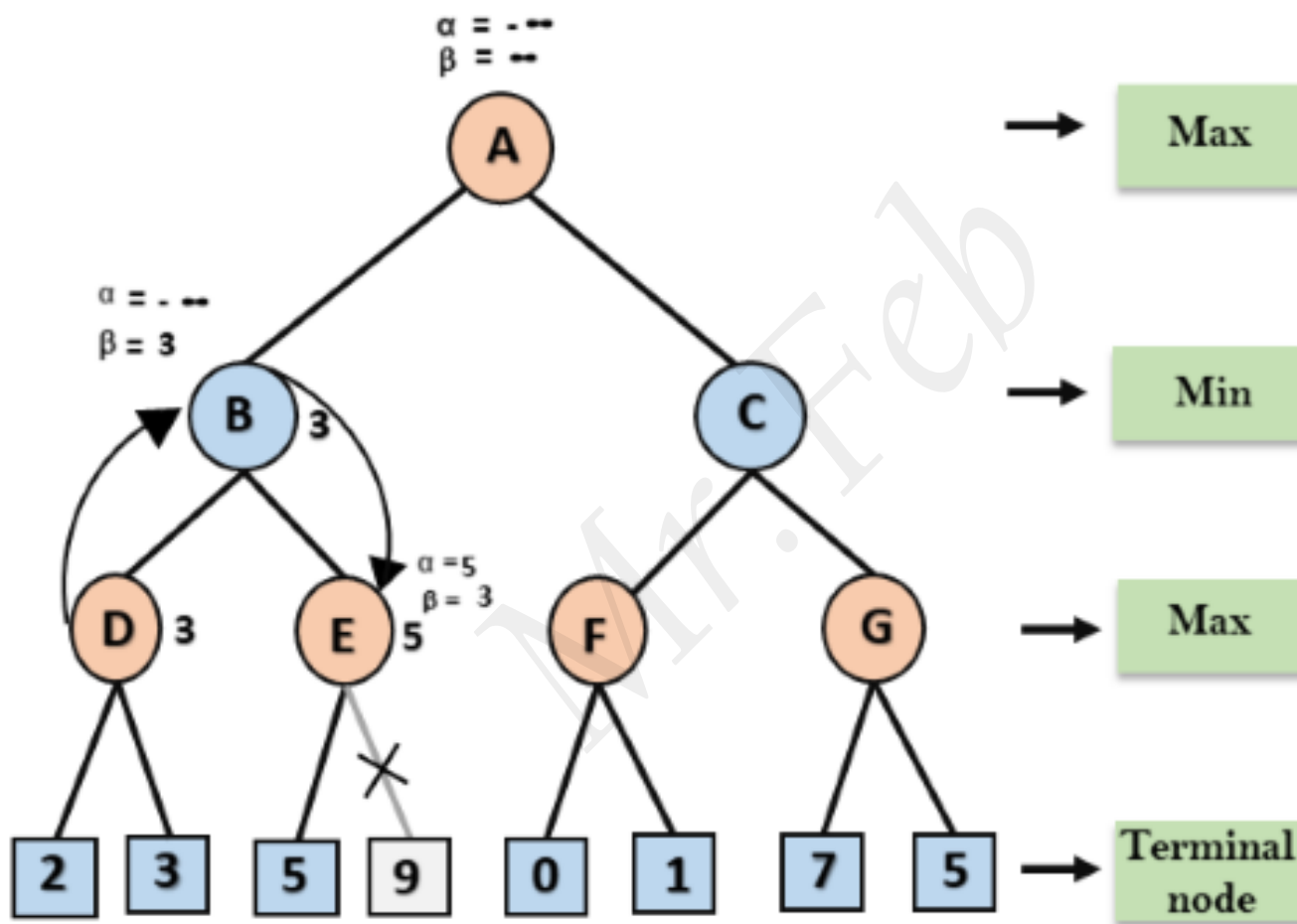
Mr. Feb

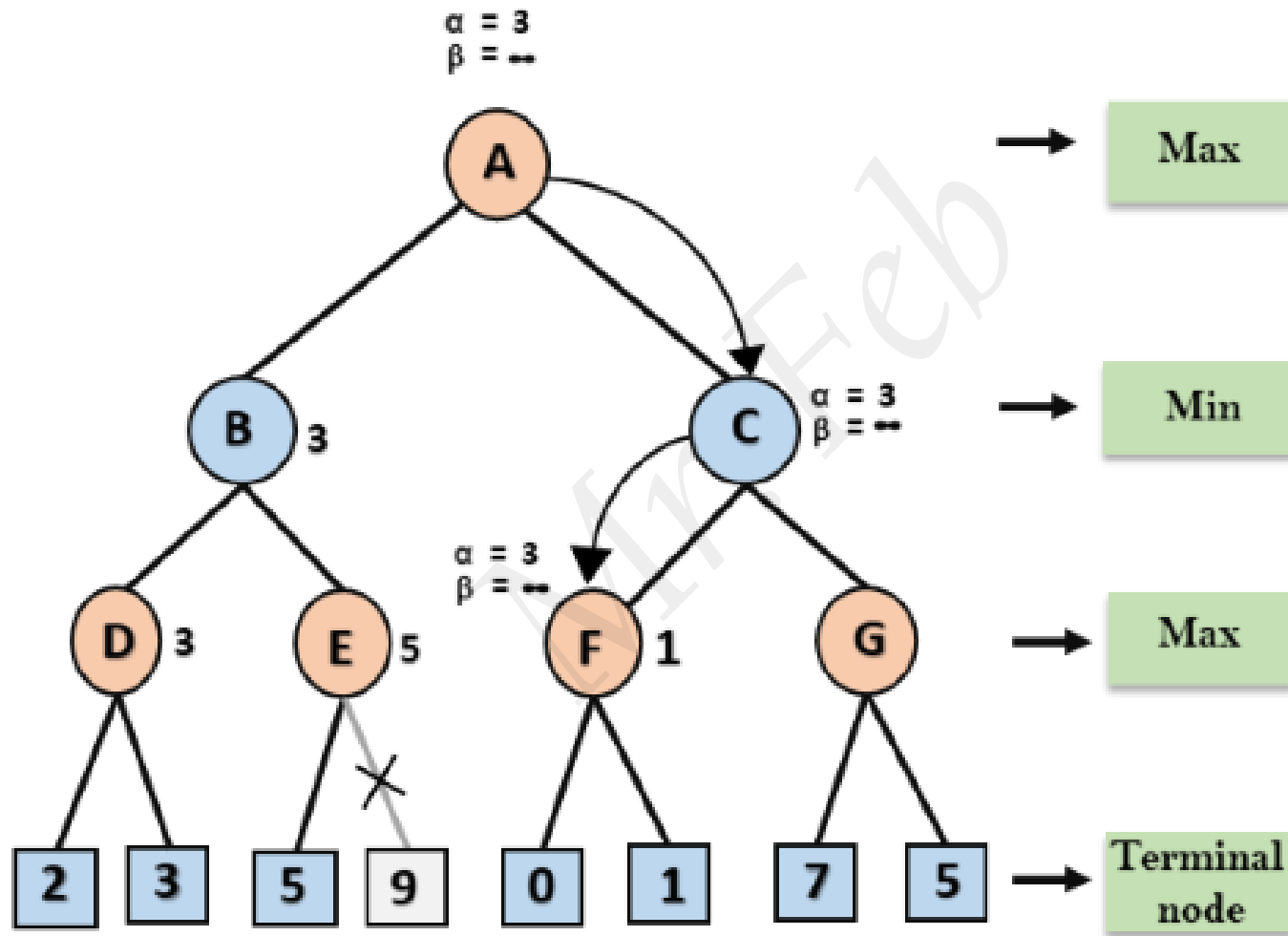
## Another Example:

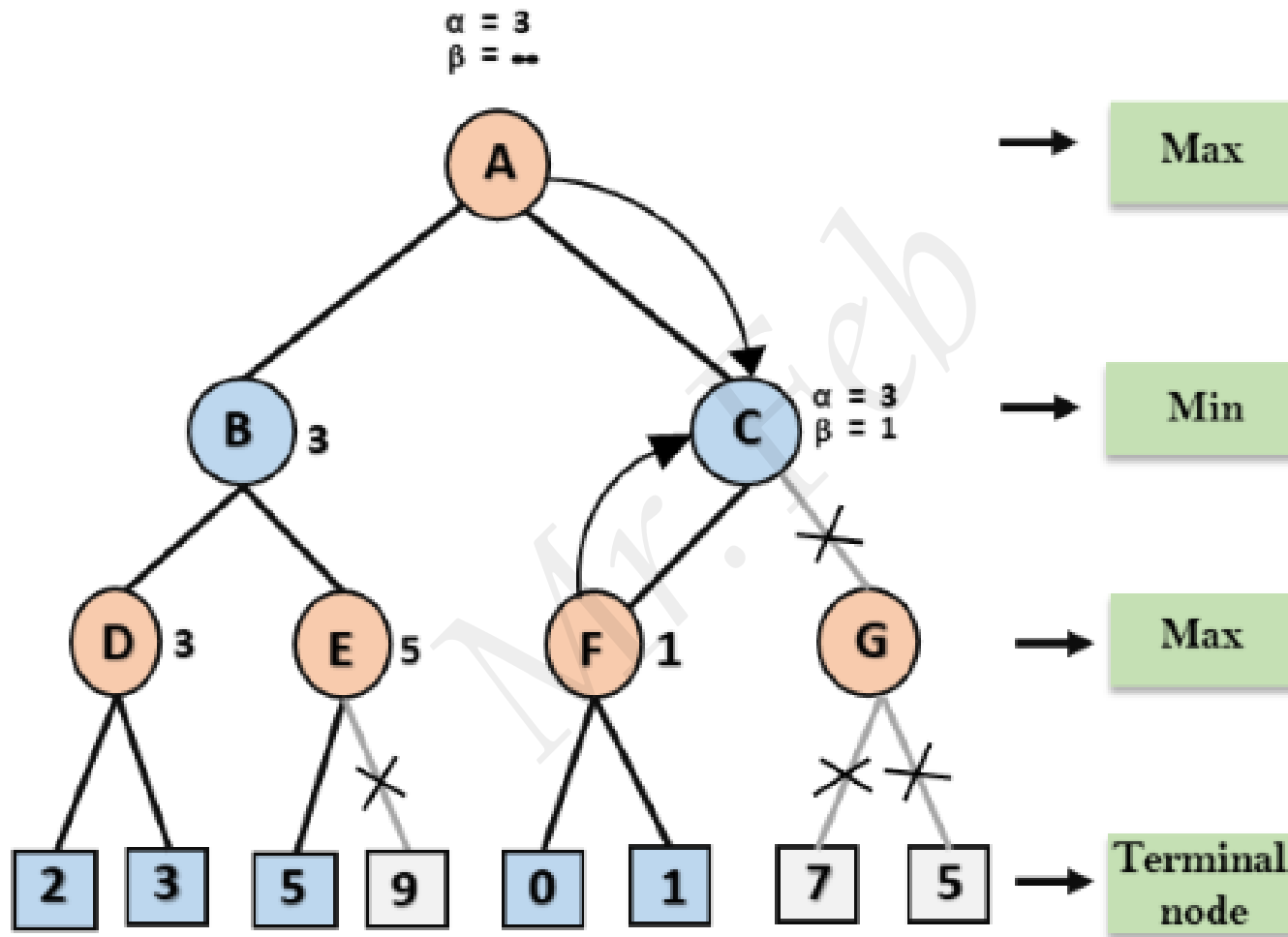


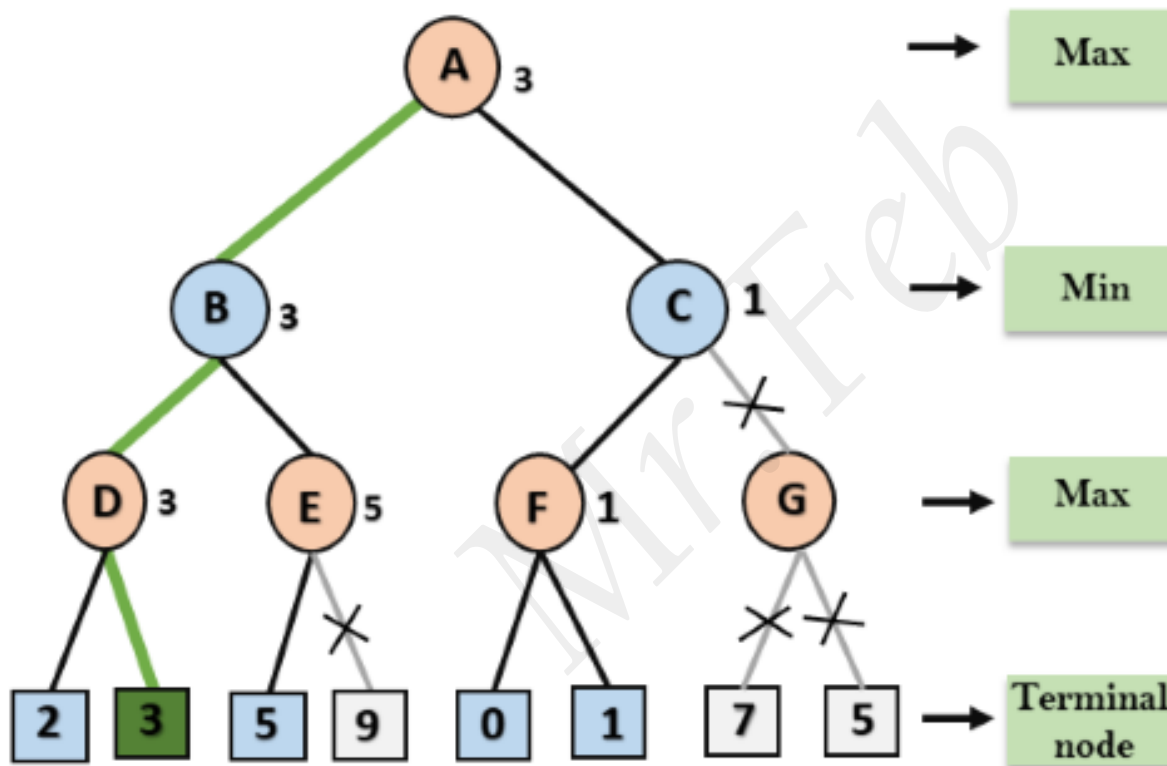


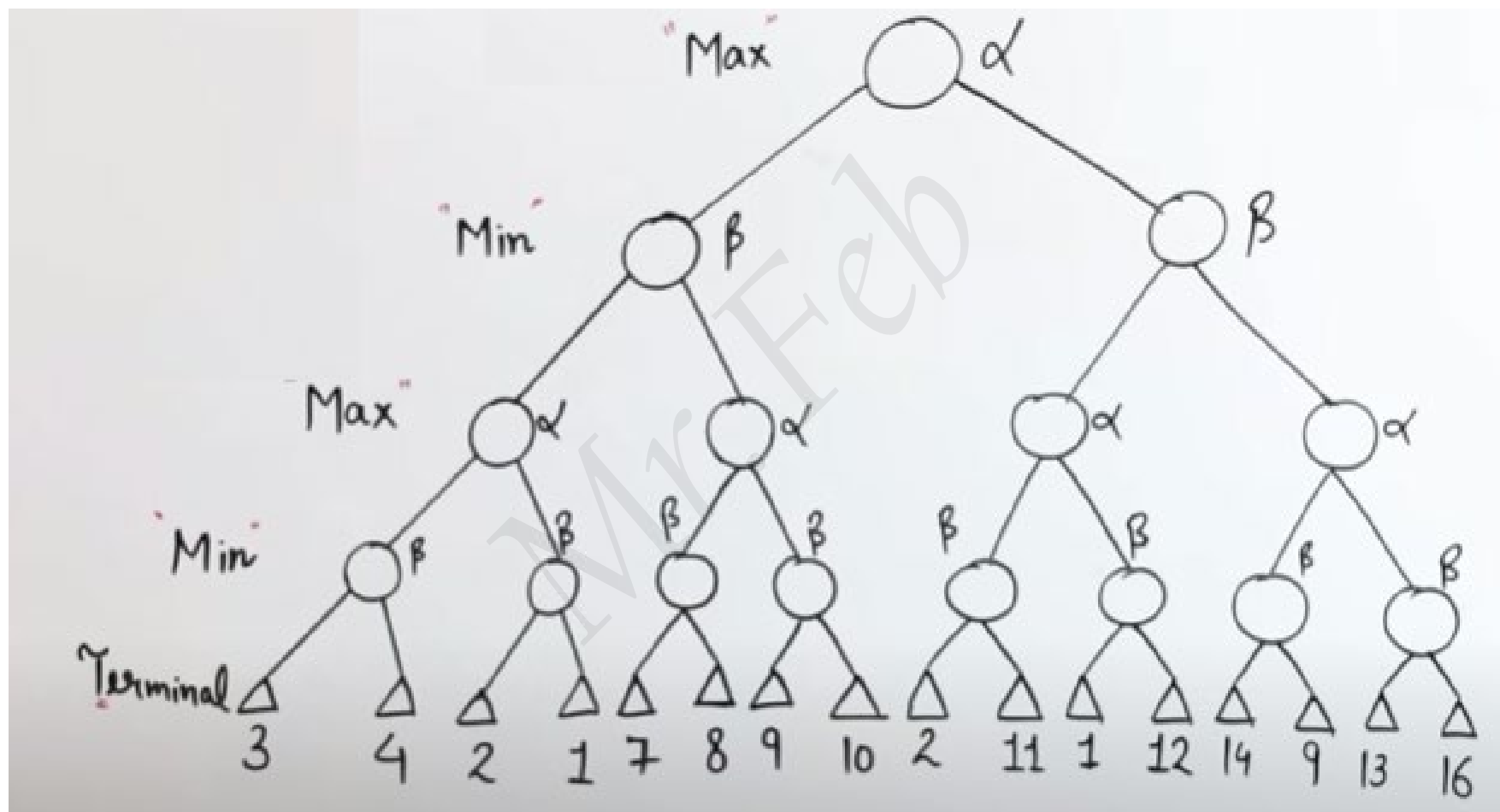


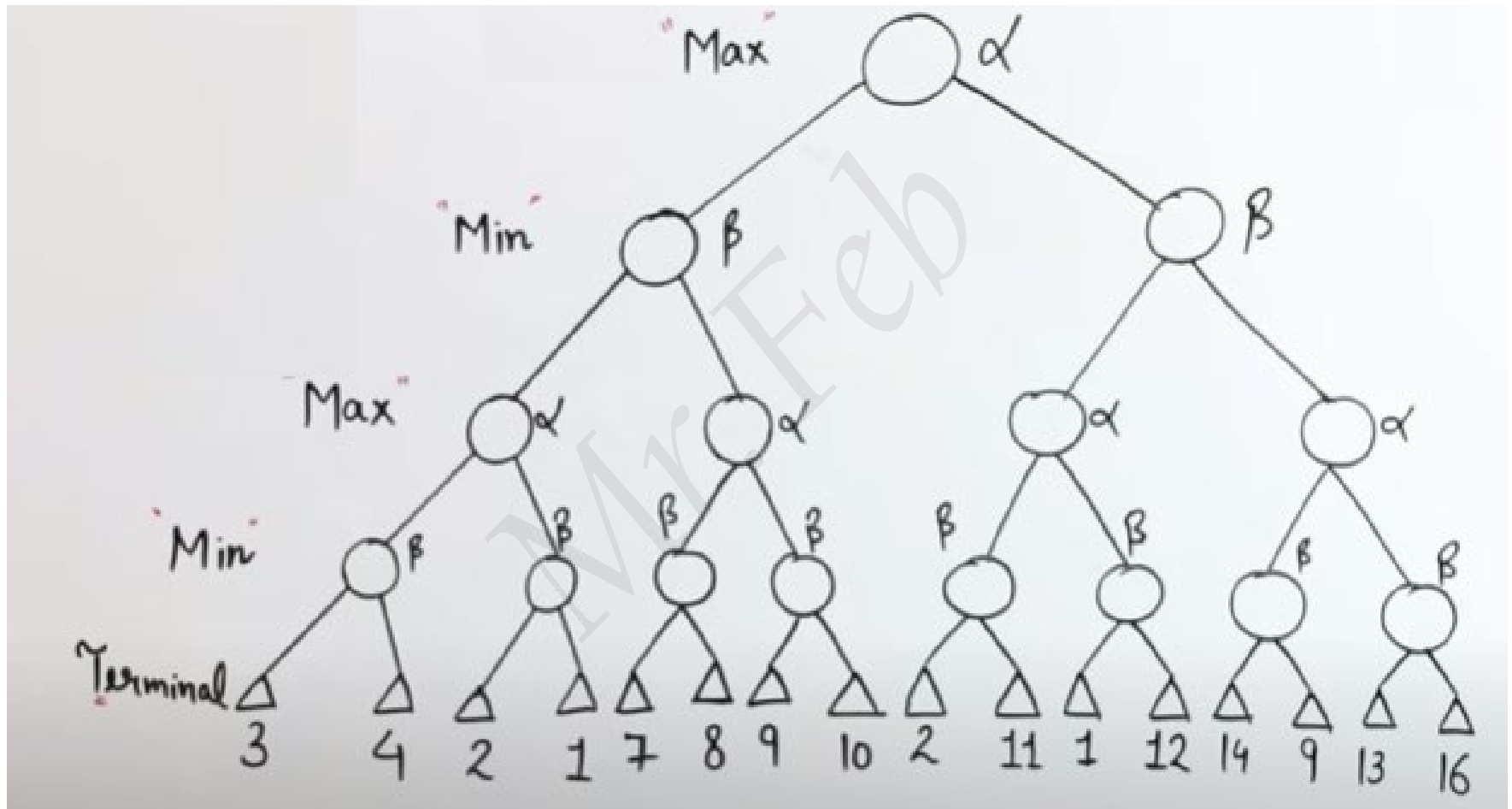


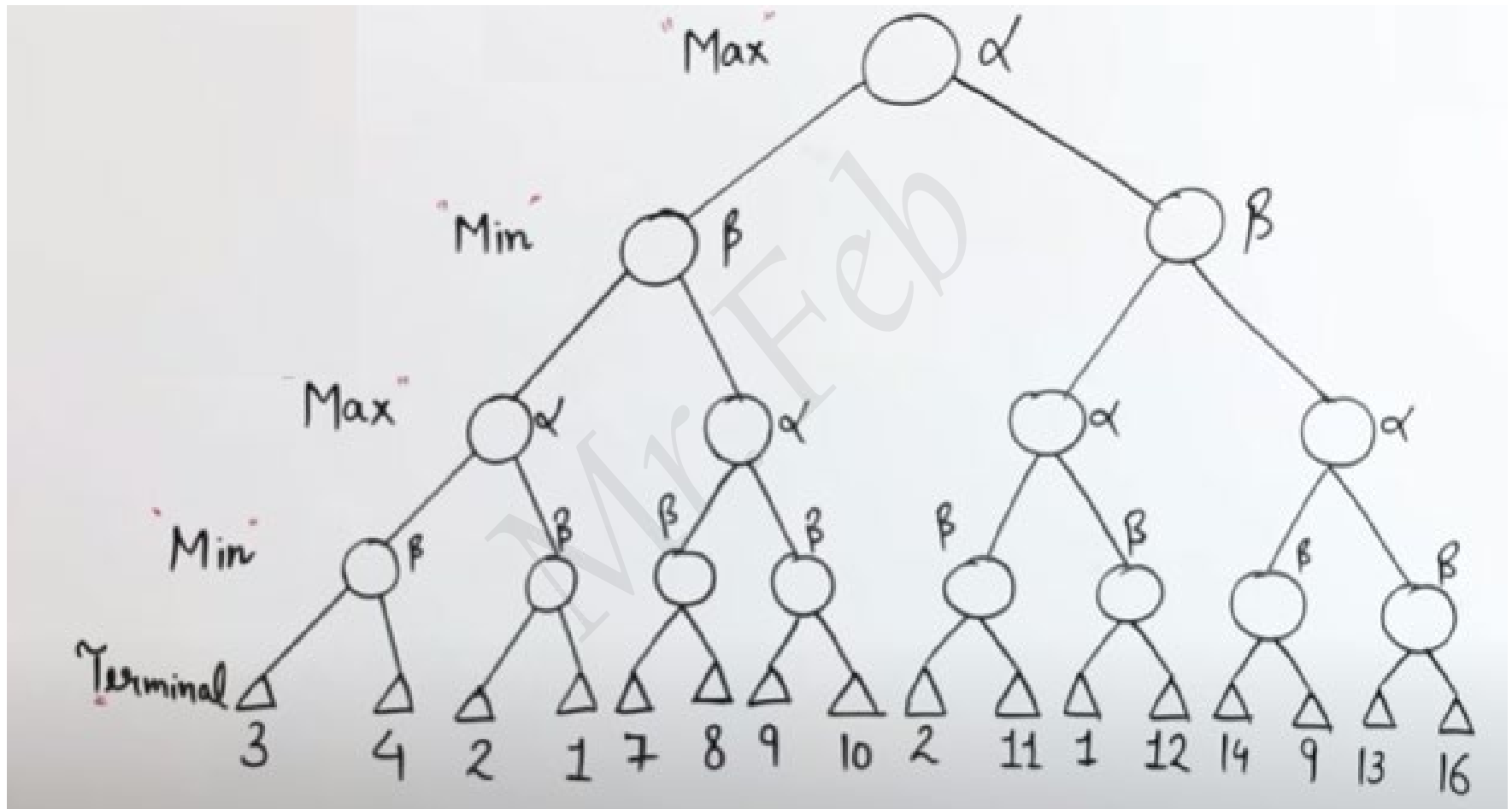




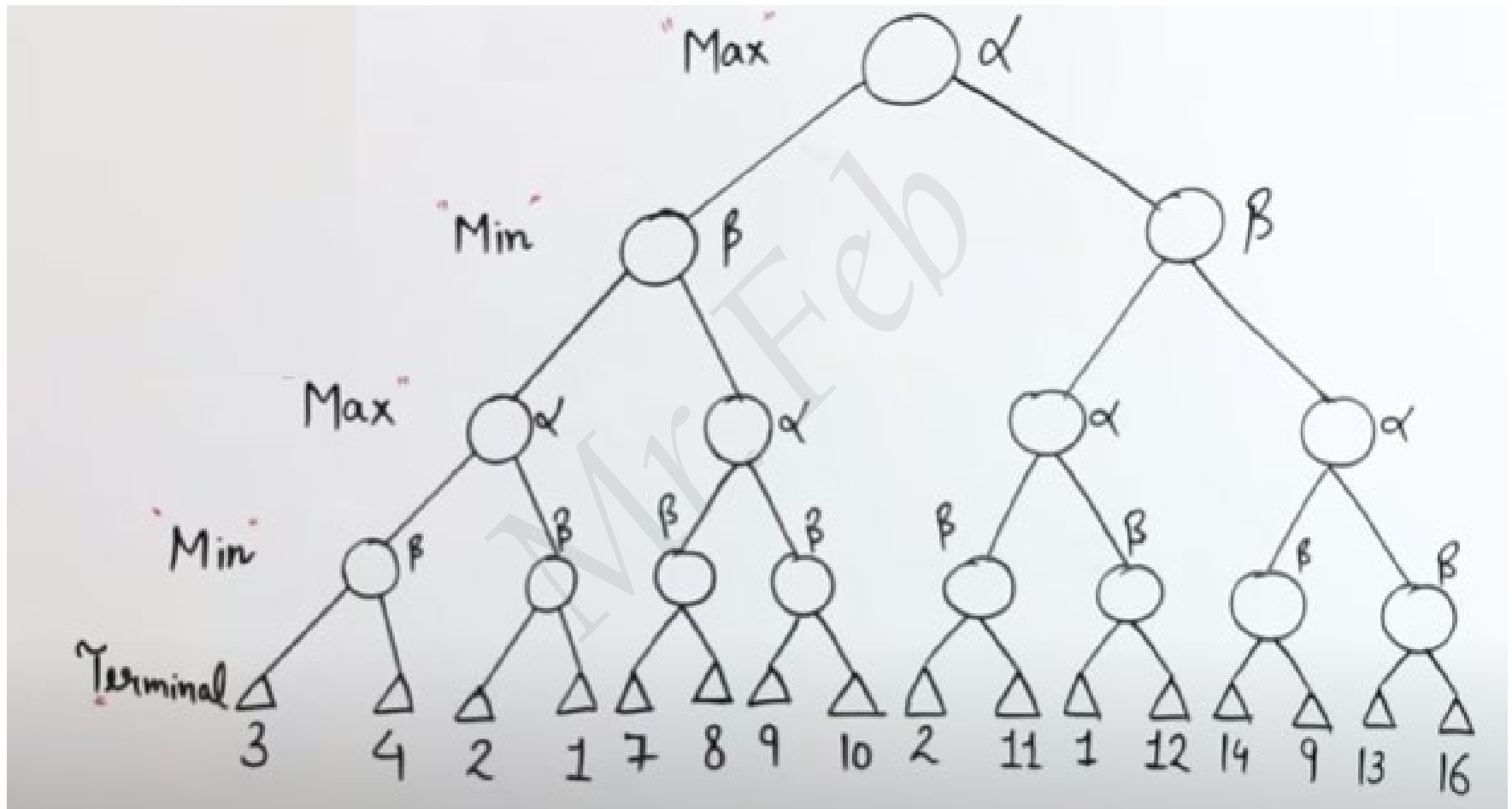


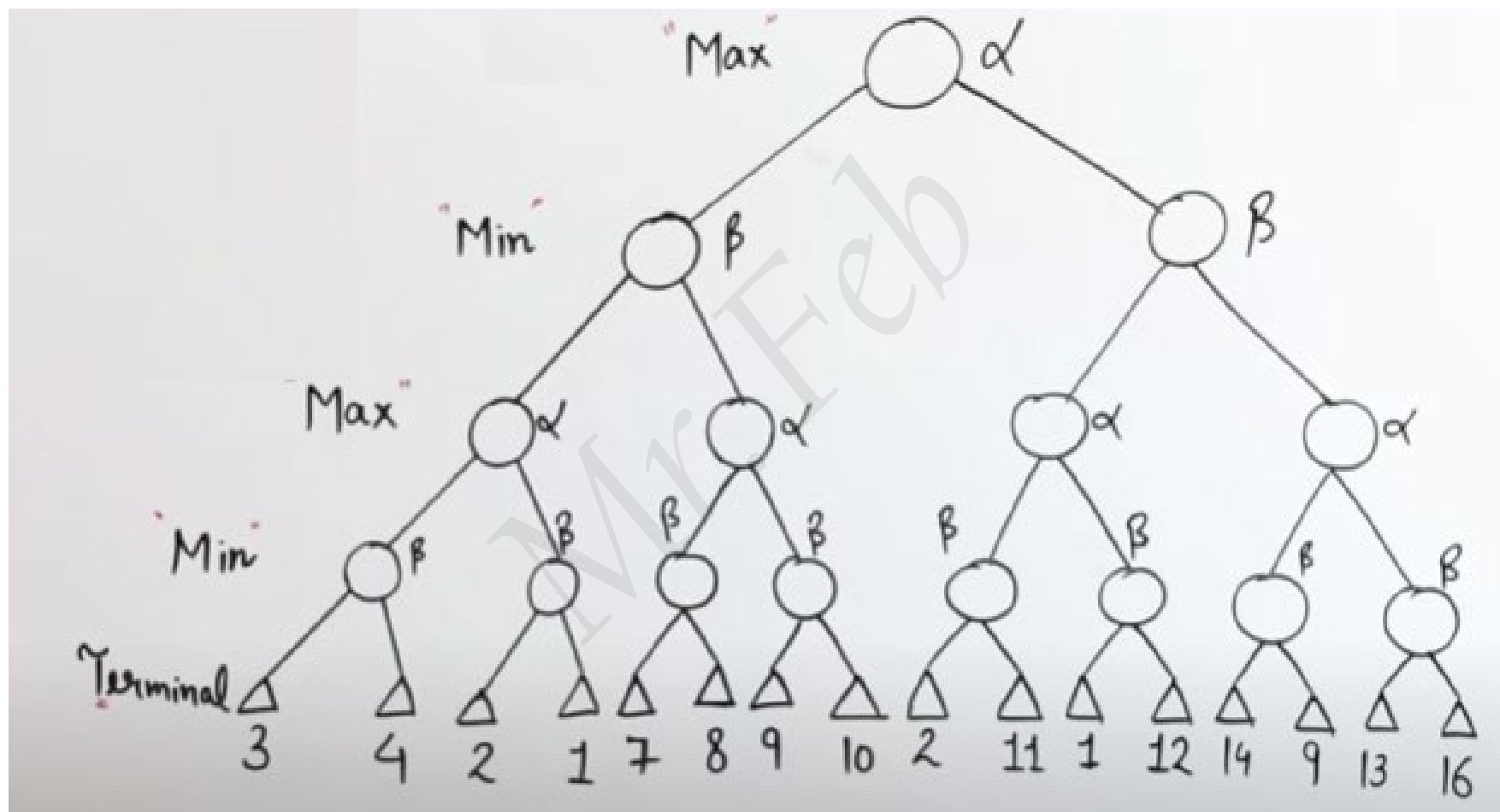












Which is used to compute the truth of any sentence?

- a) Semantics of propositional *logic*
- b) Alpha-beta pruning
- c) First-order logic
- d) Both Semantics of propositional logic & Alpha-beta pruning

Mr. Feb



**Thank You !!!**