Game Playing

Applications of game playing, mini-max algorithm and alpha-beta pruning, perfect decision game and imperfect decision game

Adversarial Search

- Adversarial search is a problem-solving technique used in artificial intelligence (AI) to find the best move or strategy in a competitive, two-player game.
- The term "adversarial" refers to the fact that each player's goal is in direct conflict with the other player's goal, making it necessary to anticipate the opponent's moves and counter them effectively.
- In adversarial search, the two players are typically referred to as "max" and "min," representing the maximizing player (who seeks to maximize their own outcome) and the minimizing player (who seeks to minimize the maximizing player's outcome).
- The goal of the maximizing player is to find the move that leads to the best possible outcome for themselves, while considering the opponent's possible moves and trying to anticipate their response.

Adversarial Search

- There might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.

Types of Games in AI

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

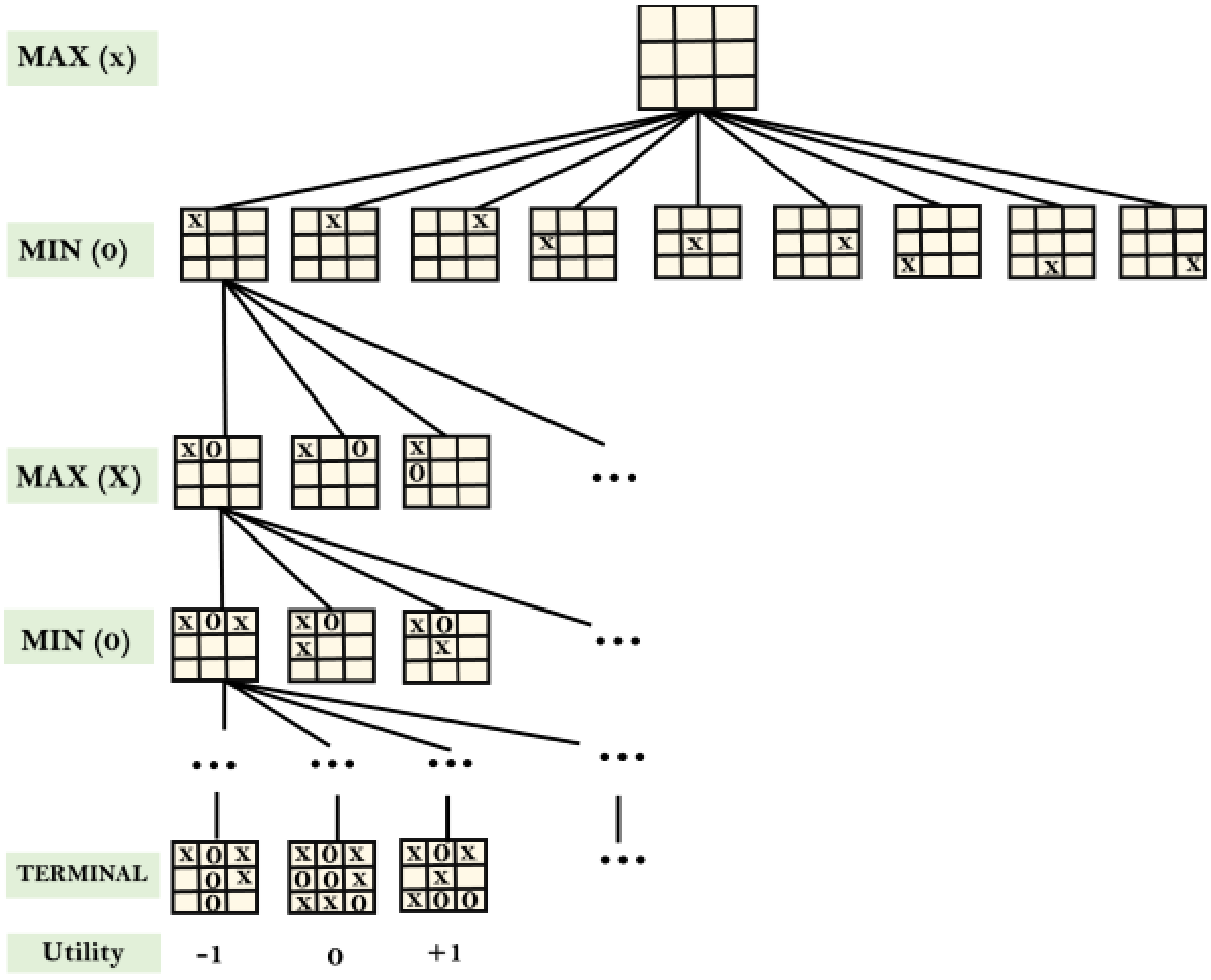
Formalization of the problem

- A game can be defined as a type of search in AI which can be formalized of the following elements:
  - **Initial state:** It specifies how the game is set up at the start.
  - **Player(s):** It specifies which player has moved in the state space.
  - **Action(s):** It returns the set of legal moves in state space.
  - **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
  - **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
  - **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

Game tree

- A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, action function, and result Function.
- Example: Tic-Tac-Toe game tree:

- The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:
- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.


MAX (x)
MIN (o)
MAX (X)
MIN (o)
TERMINAL
Utility
-1
0
+1

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.

- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.

- In this algorithm two players play the game, one is called MAX and other is called MIN.

Cont..

- 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 minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

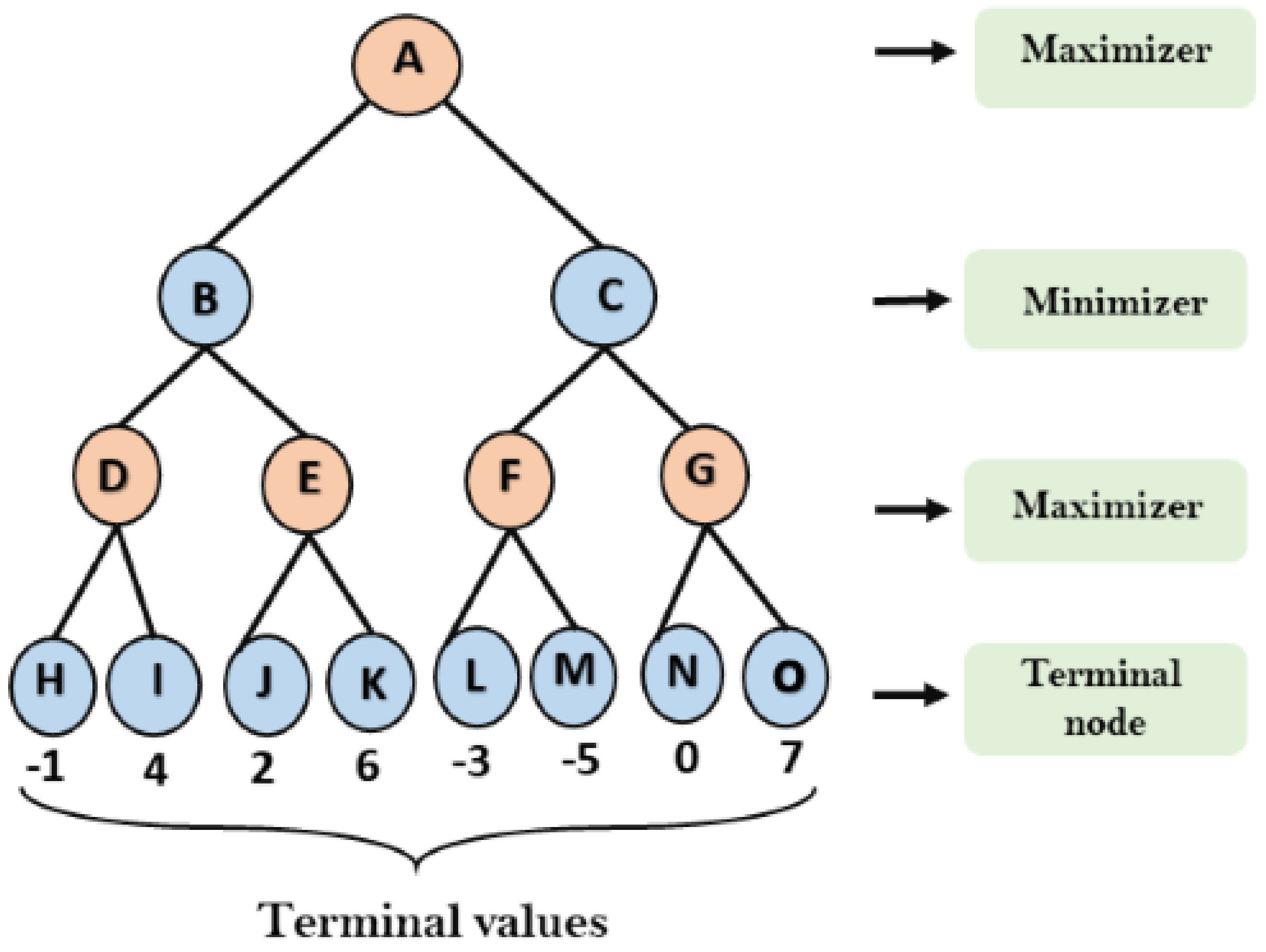
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players 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.
- Following are the main steps involved in solving the two-player game tree:

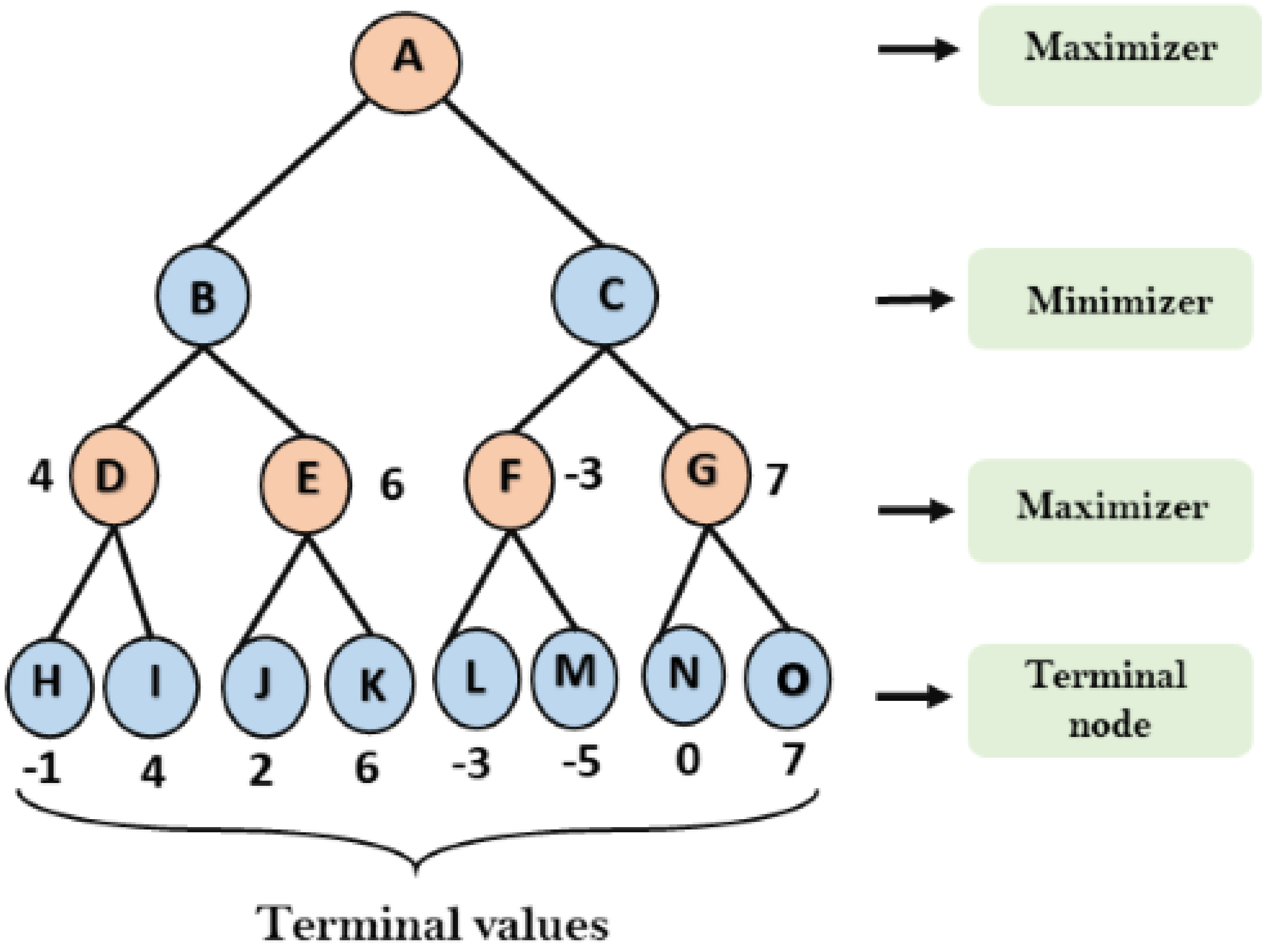
Step 1

- 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.
- In the below tree diagram, let's take A is the initial state of the tree.
- 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.


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

Step 2

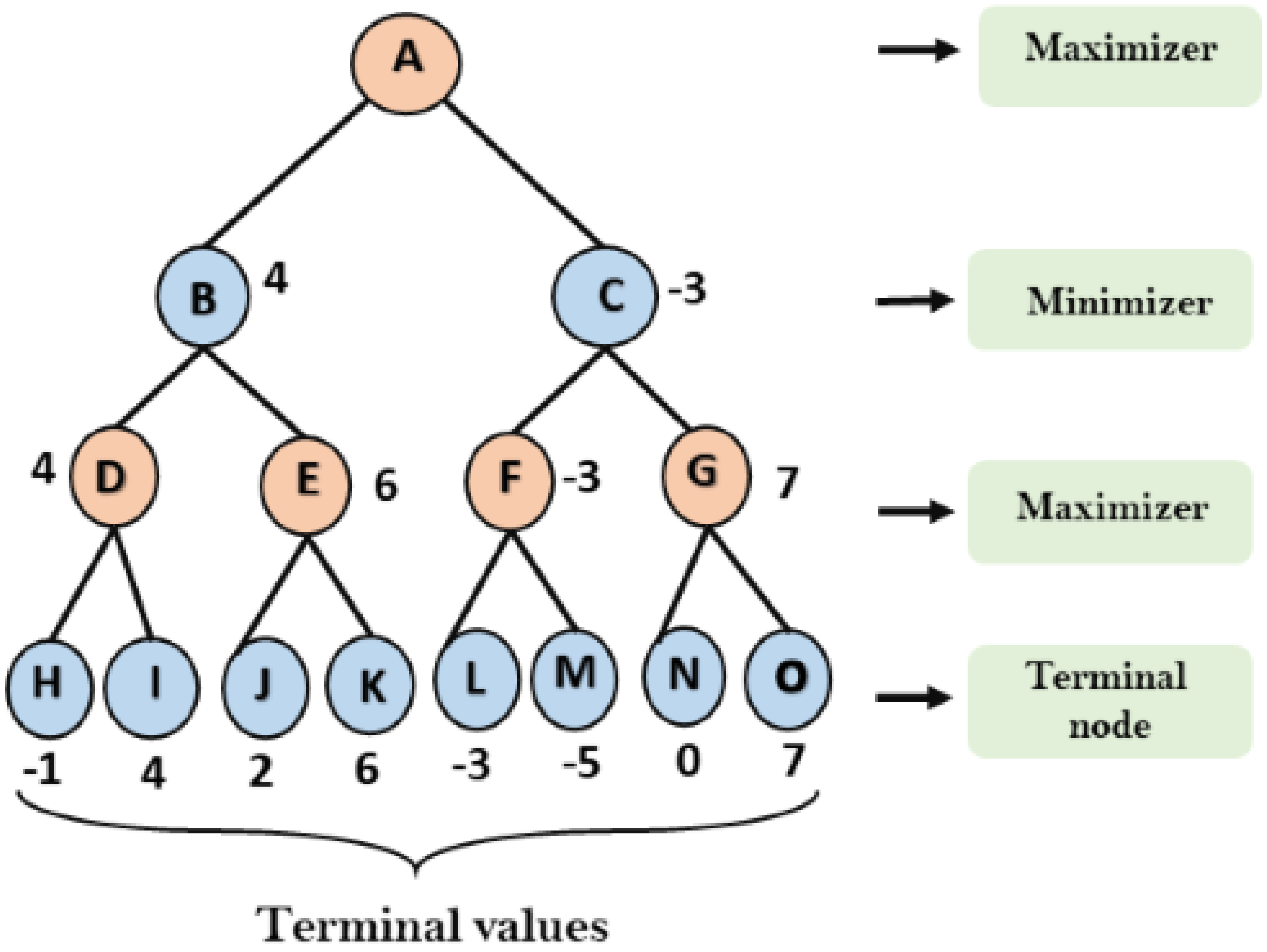
- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.
- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



A
B
C
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

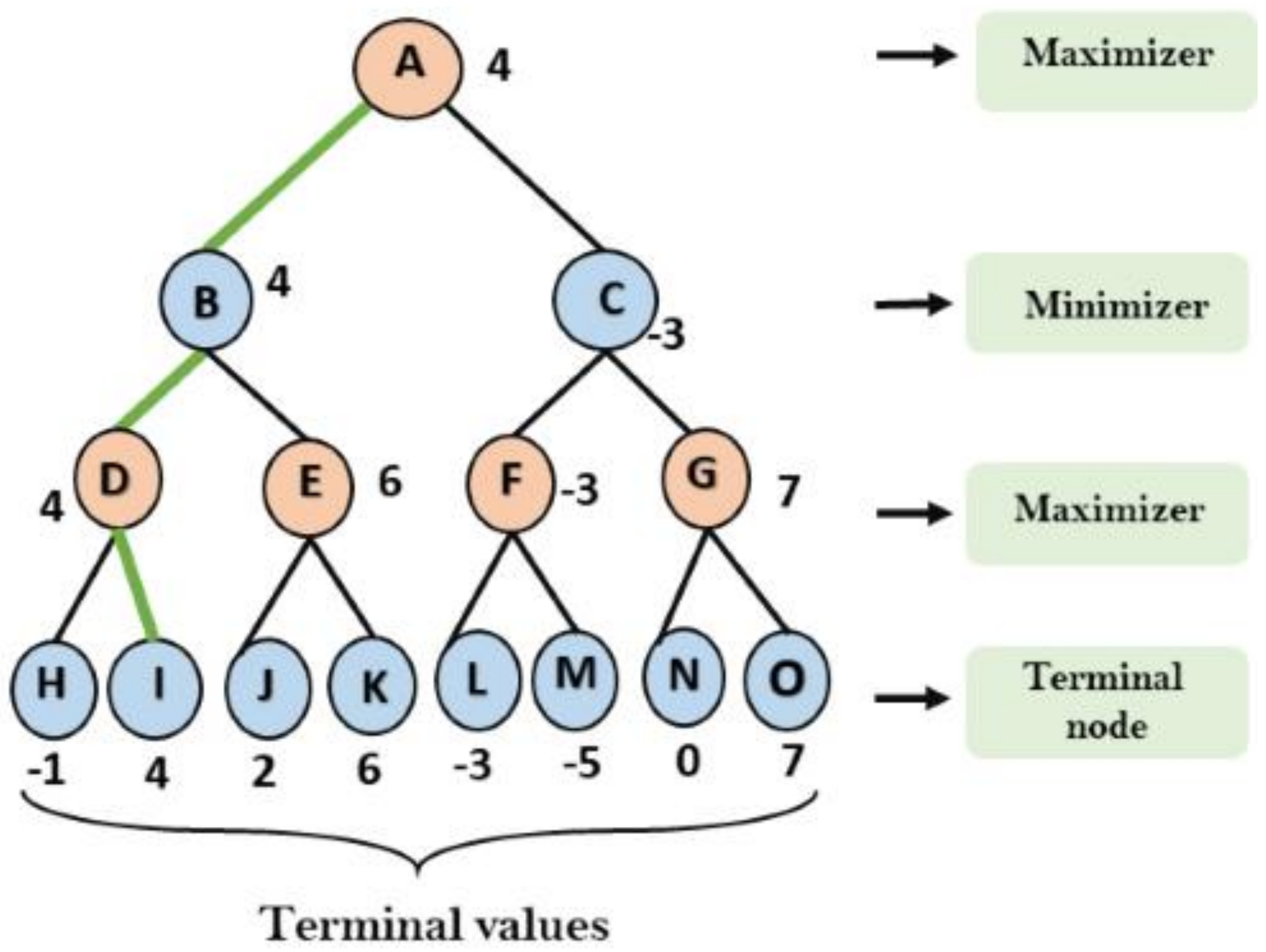
Step 3

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


A
Maximizer
B
4
C
-3
Minimizer
4
D
E
6
F
-3
G
7
Maximizer
H
I
J
K
L
M
N
O
Terminal node
-1
4
2
6
-3
-5
0
7
Terminal values

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



A 4
B 4
C -3
D 4
E 6
F -3
G 7
H I J K L M N O
-1 4 2 6 -3 -5 0 7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

Properties of Mini-Max algorithm

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

Limitation of the Minimax Algorithm

- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc.
- This type of games has a huge branching factor, and the player has lots of choices to decide.

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm.
- It is an optimization technique for the minimax algorithm.
- Pruning is a technique by which without checking each node of the game tree we can compute the correct minimax decision.
- This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. or Alpha-Beta Algorithm.
- Alpha-beta pruning can be applied at any depth of a 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

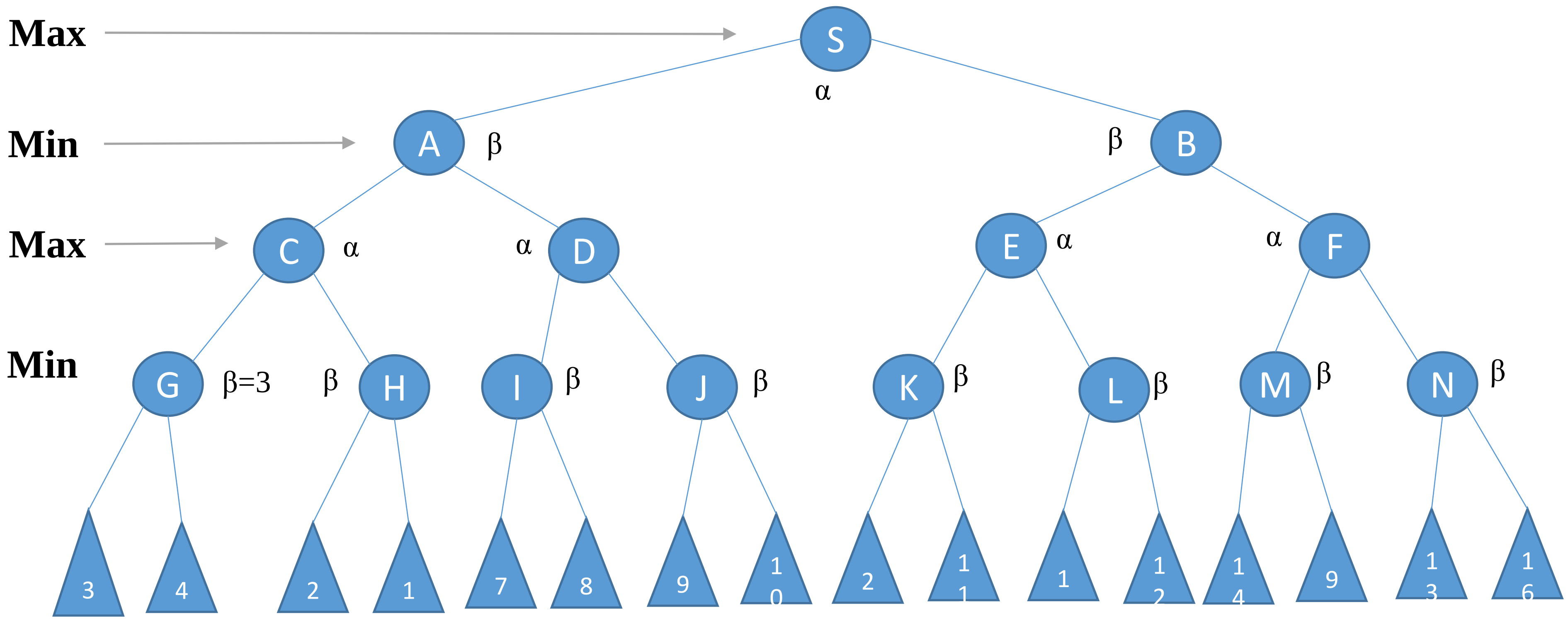
- The main condition which required for alpha-beta pruning is:
- $\alpha >= \beta$

Points To Remember 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.

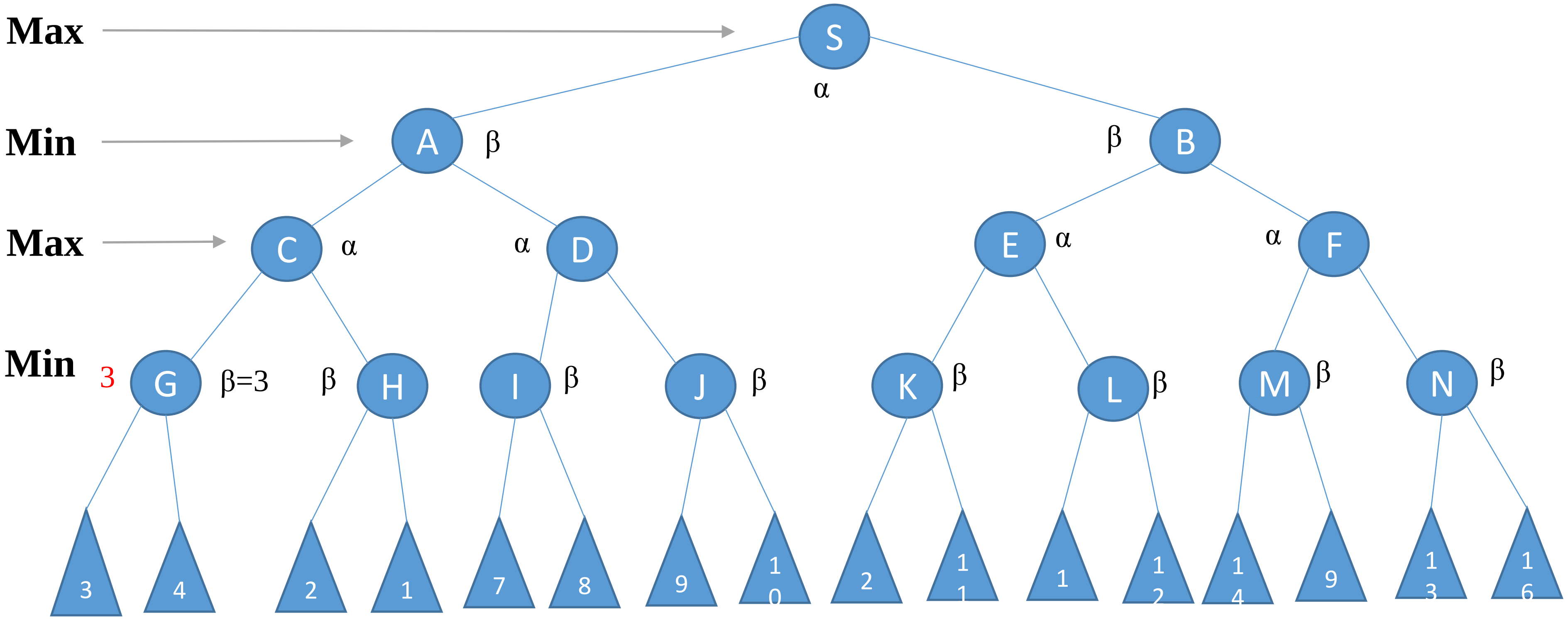
Working of Alpha-Beta Pruning

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



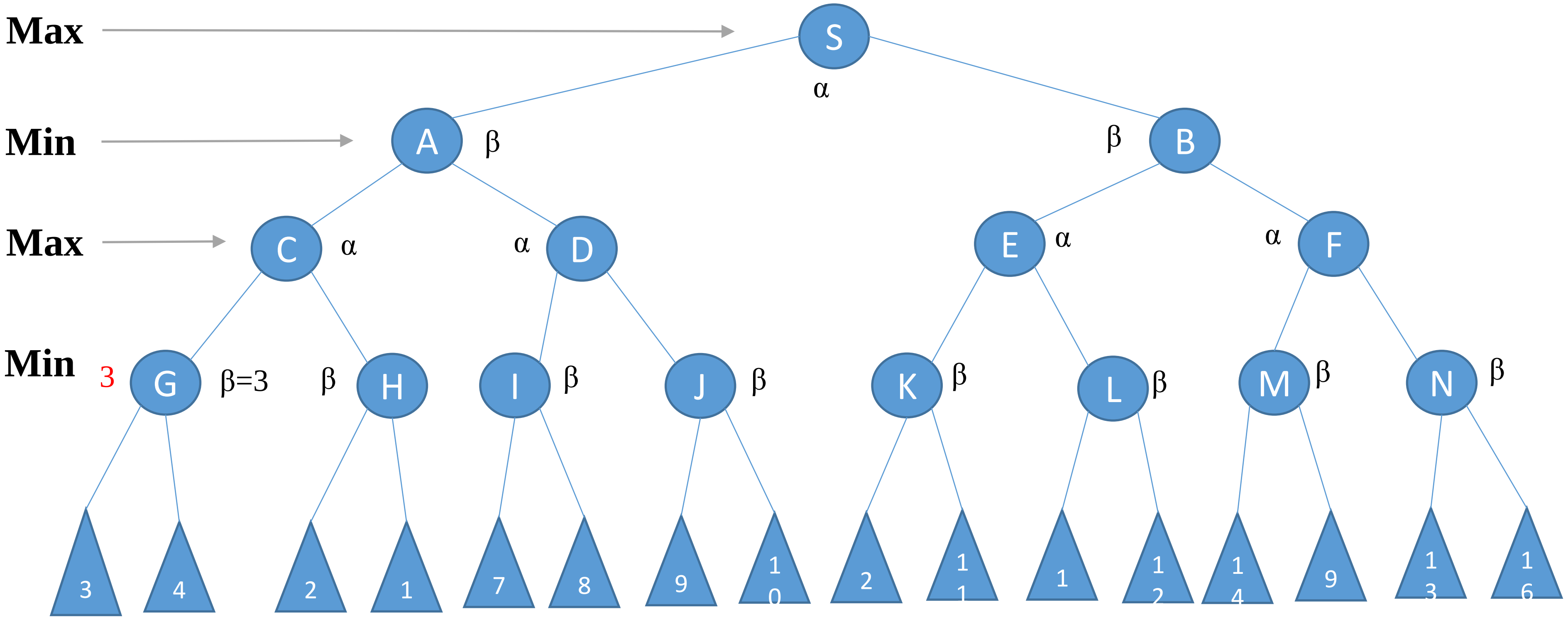
Working of Alpha-Beta Pruning

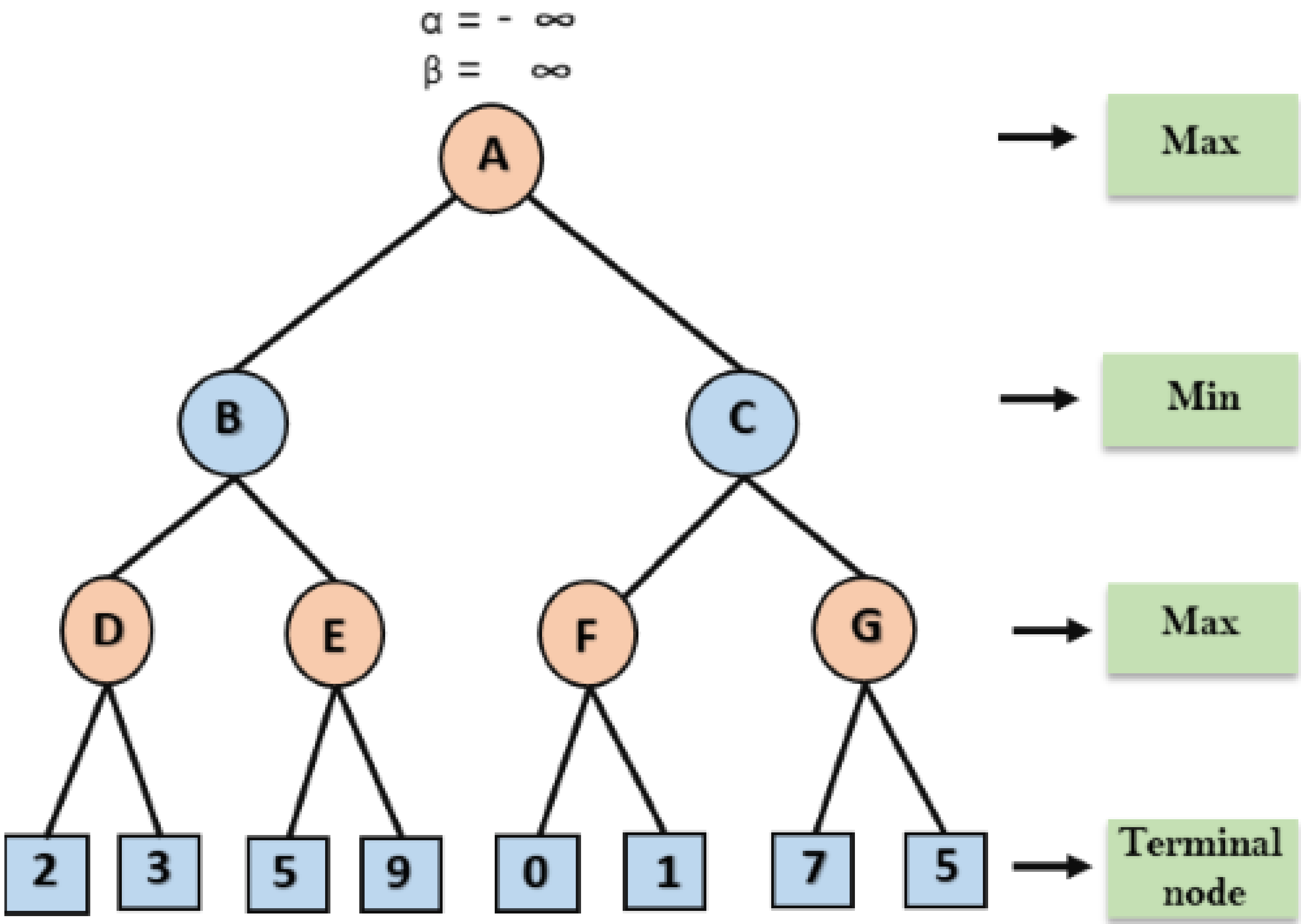
- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning



Working of Alpha-Beta Pruning

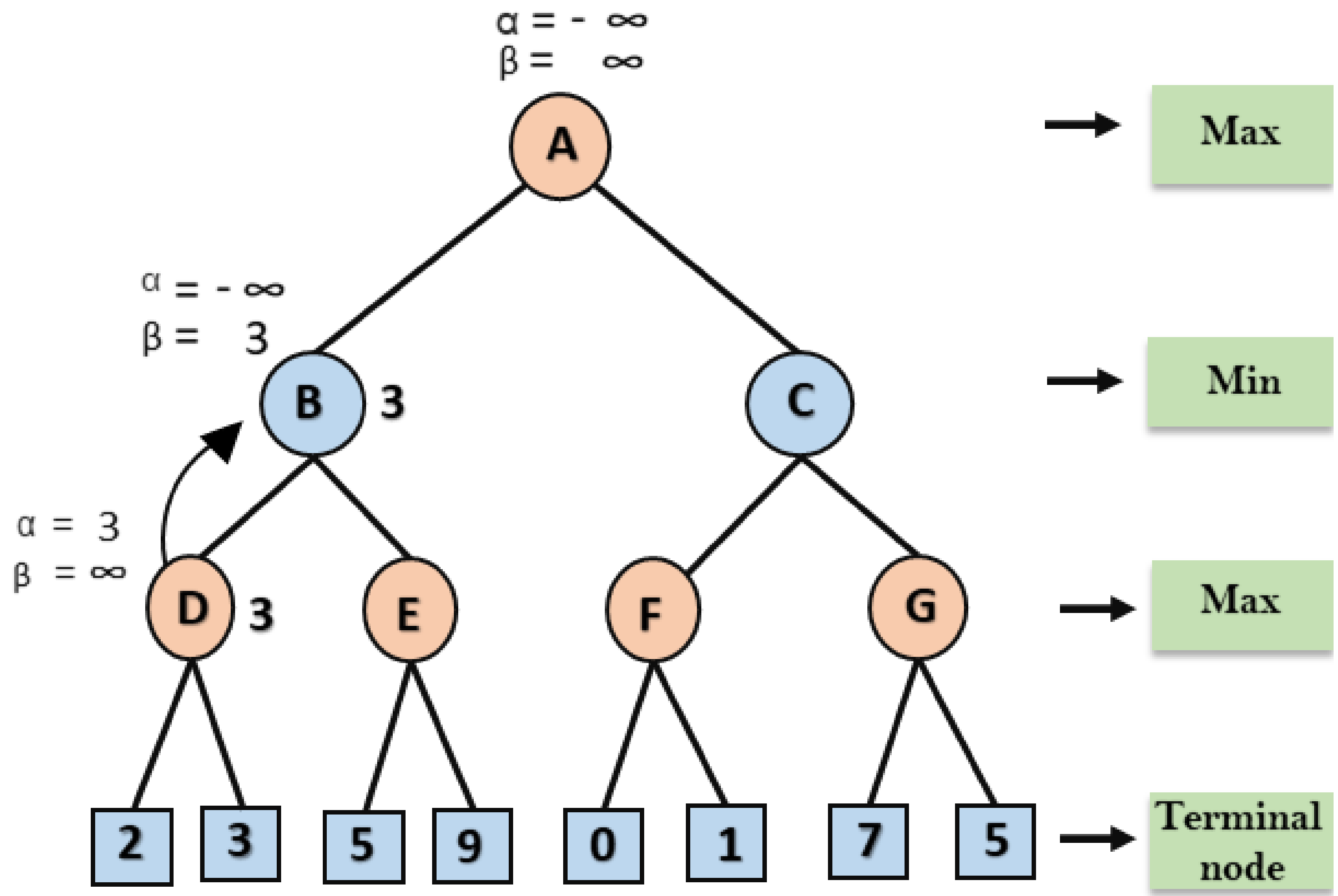
- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning



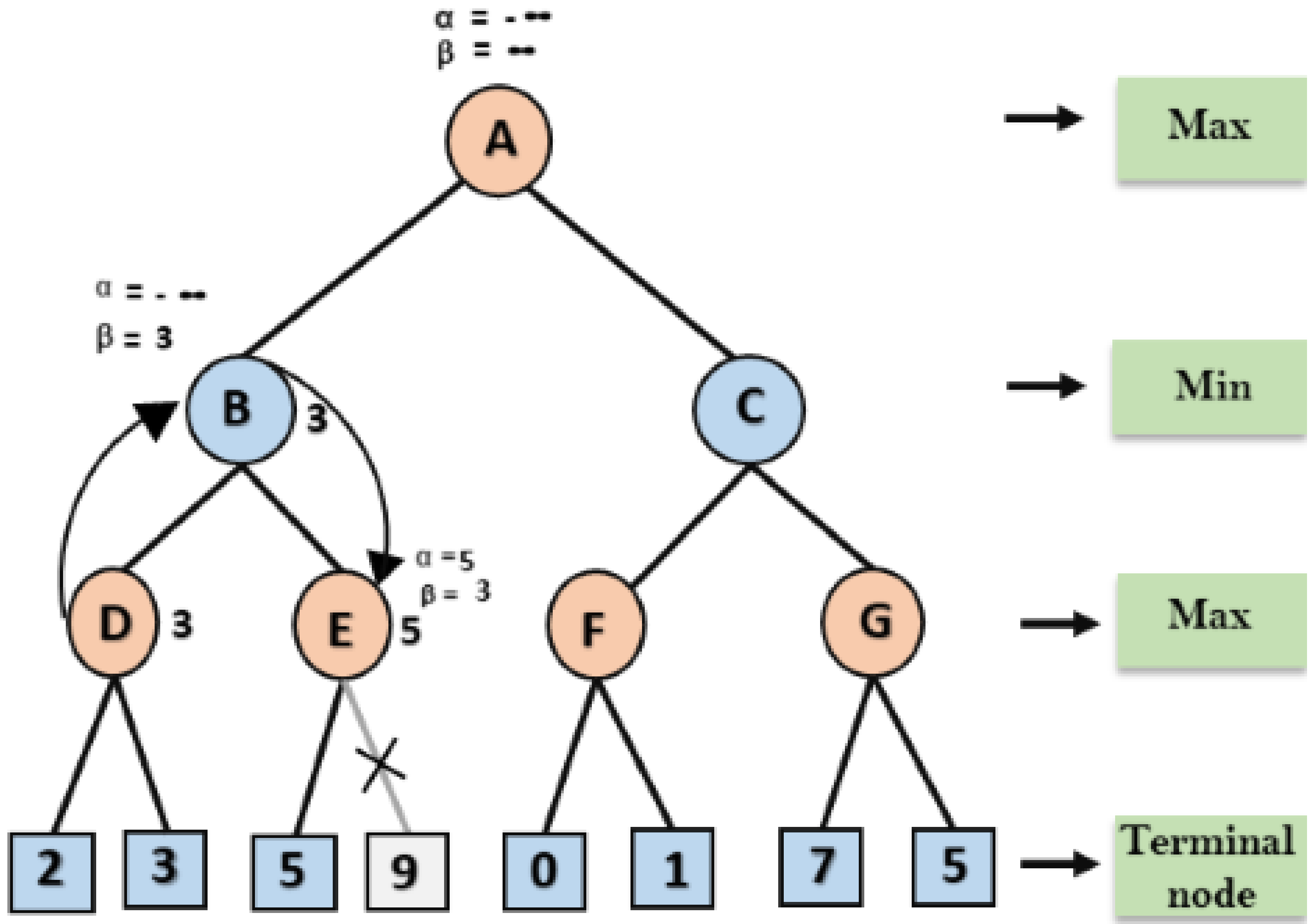


α = - ∞
β = ∞
A
B
C
D
E
F
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

- **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.



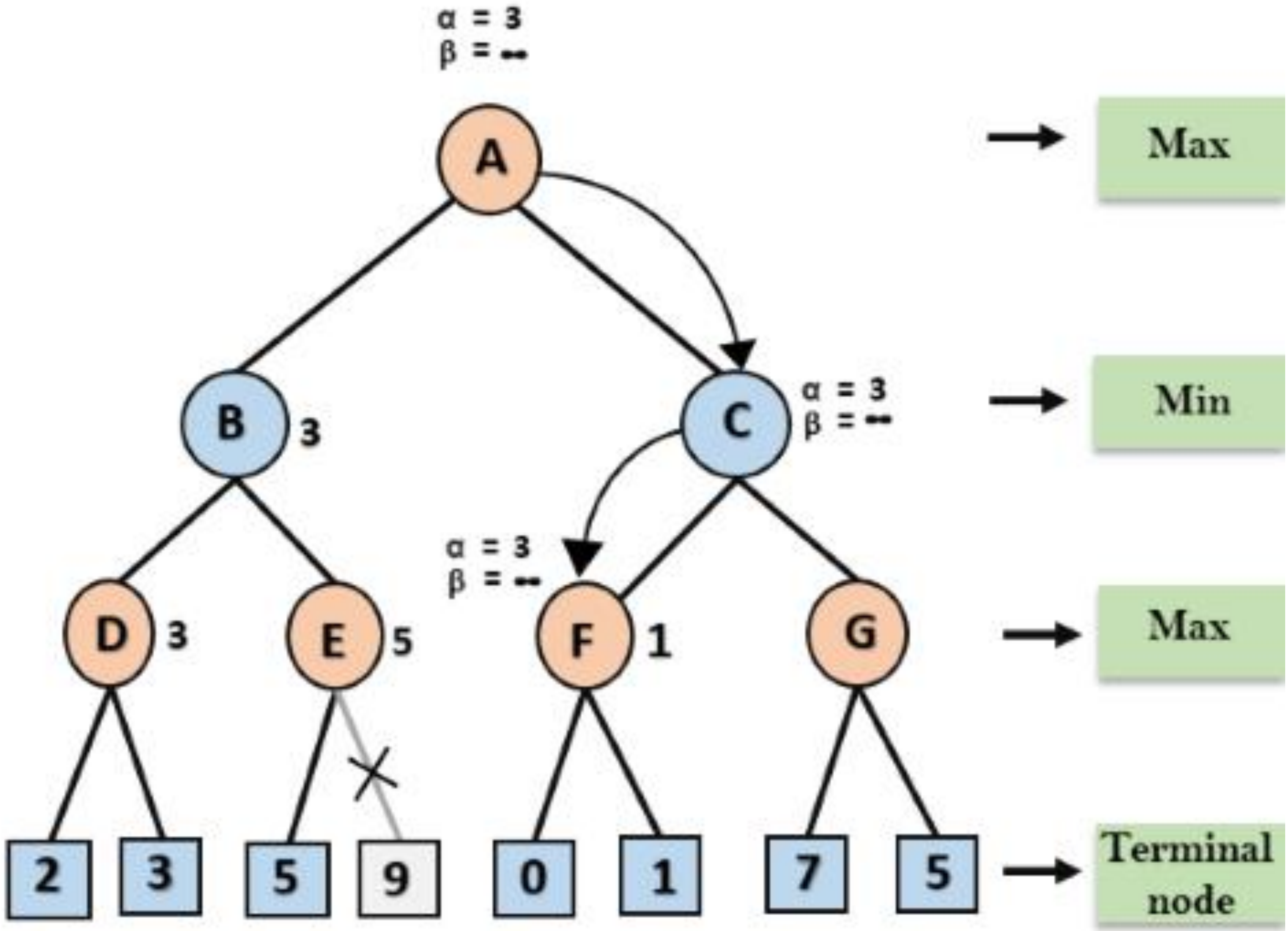
α = - ∞
β = ∞
A
Max
α = - ∞
β = 3
B
3
C
Min
α = 3
β = ∞
D
3
E
F
G
Max
2
3
5
9
0
1
7
5
Terminal node



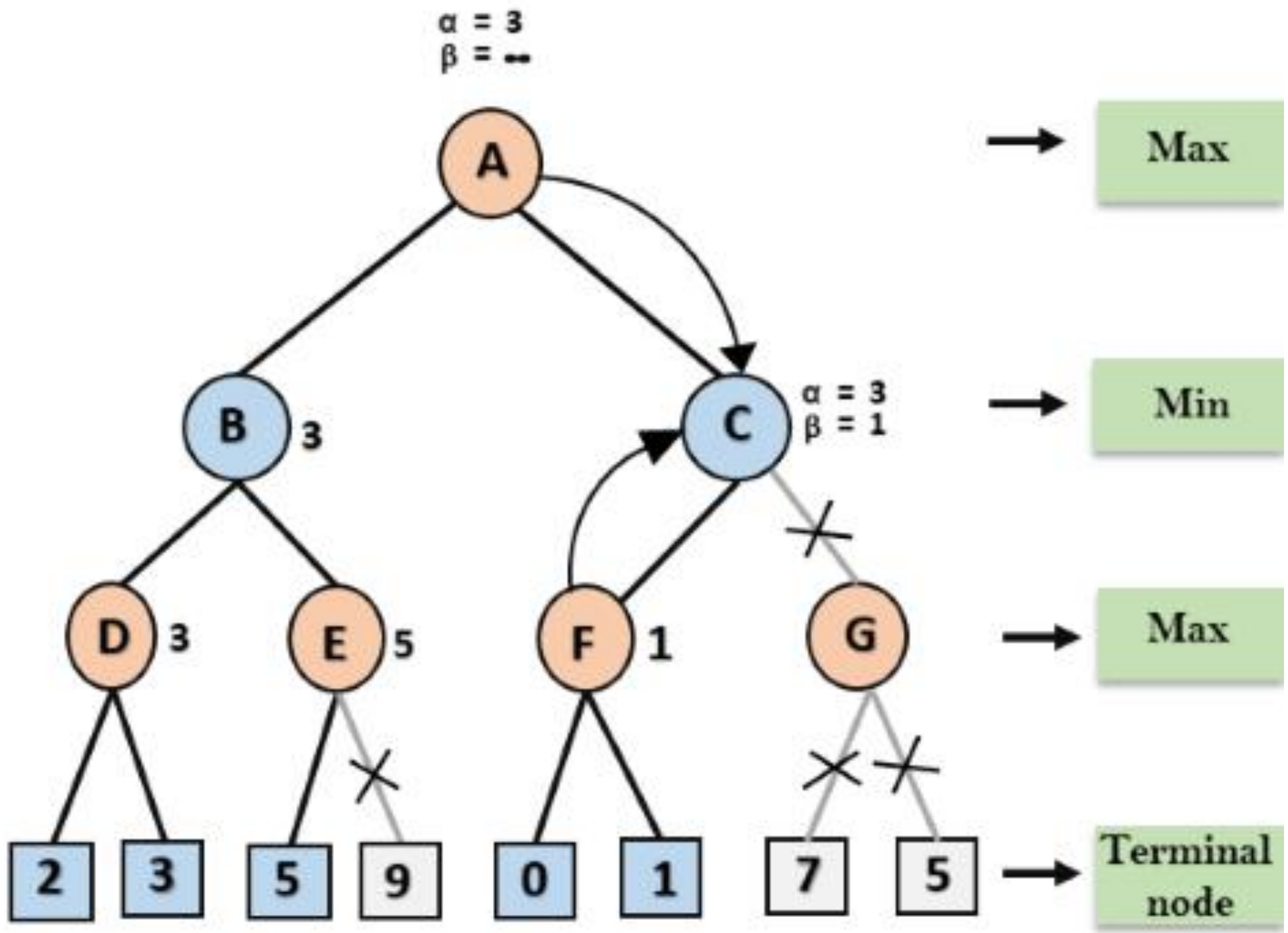
α = -∞
β = ∞
α = -∞
β = 3
A
B
C
D
E
F
G
3
3
5
α =5
β = 3
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

- **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.



α = 3
β = +∞
A
Max
B
3
C
α = 3
β = +∞
Min
α = 3
β = +∞
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node



α = 3
β = ∞
A
B
3
C
α = 3
β = 1
D
3
E
5
F
1
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

- **Step 8:** C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.

Move Ordering in Alpha-Beta pruning

- ***Worst ordering:*** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree.
- ***Ideal ordering:*** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time.

Rules to find good ordering

- Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

Perfect Decision Game:

- In a perfect decision game, players have complete and perfect information about the game state, available actions, and potential outcomes.
- Players can make decisions based on this perfect information, knowing the consequences of each possible action.
- Examples of perfect decision games include games like Tic-Tac-Toe or Chess, where players can see the entire board and all possible moves.

Imperfect Decision Game:

- In contrast, imperfect decision games involve uncertainty and incomplete information about the game state or the actions of other players.
- Players must make decisions based on incomplete or imperfect information, leading to uncertainty about the outcomes of their actions.
- Examples of imperfect decision games include card games like Poker, where players have limited information about the cards held by other players and must make decisions based on probabilities and bluffing.