GAME THEORY

Minmax & Alpha-Beta Pruning

INTRODUCTION TO THE MINIMAX ALGORITHM

- The Minimax algorithm is a fundamental decision-making algorithm used in two-player, zero-sum games to determine the best move for a player, assuming the opponent plays optimally.
- It's a recursive algorithm that builds a game tree, evaluating all possible moves and counter-moves to select the path that maximizes the player's chances of winning..

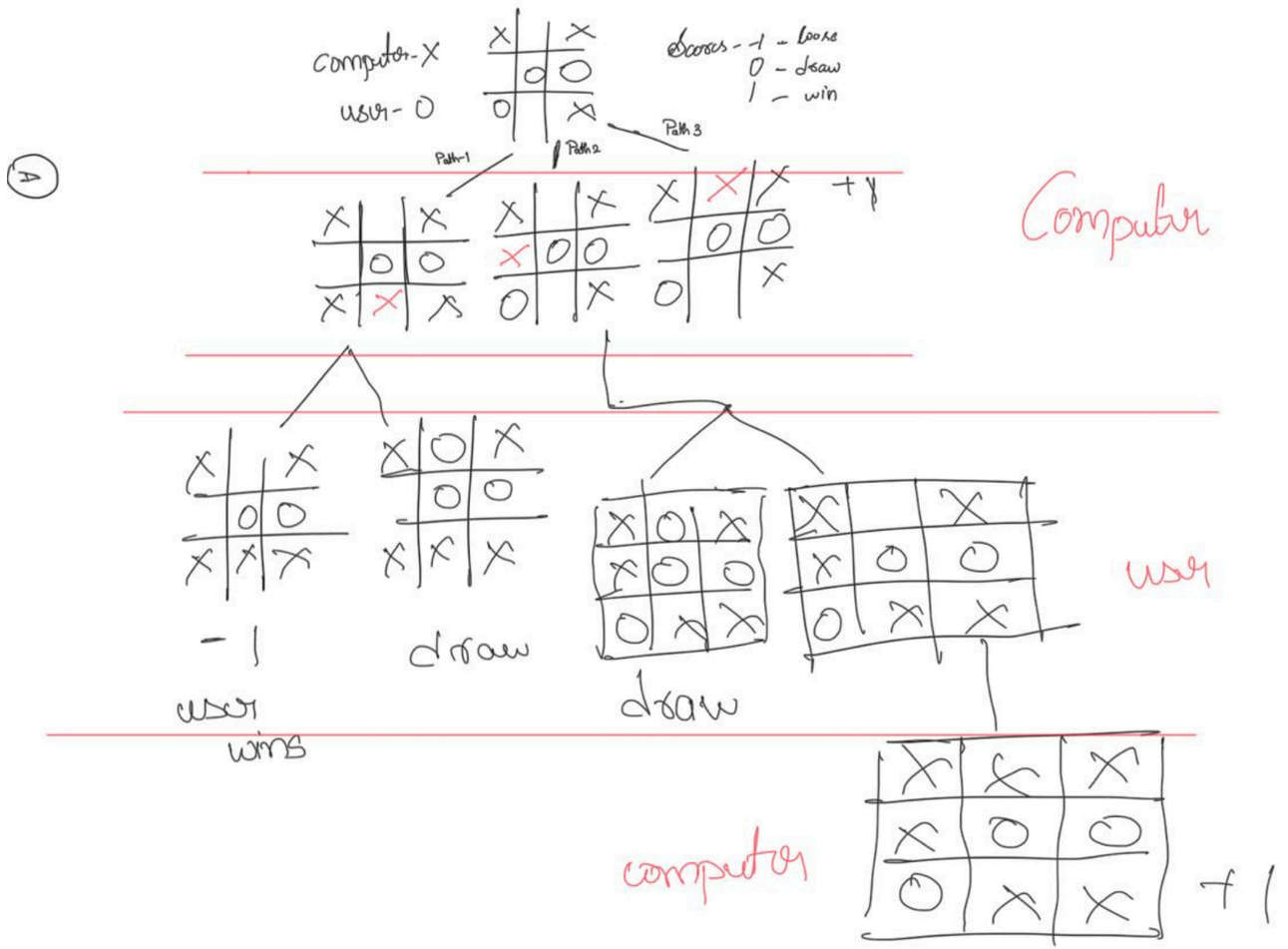
CORE CONCEPTS OF THE MINIMAX ALGORITHM

Maximizing player : tries to maximize the score by choosing best possible move

Minimizing player : tries to minimize the score

Game tree: Represnts all possible game states

MINIMAX IN ACTION: A SIMPLE EXAMPLE



MINIMAX IN ACTION: A SIMPLE EXAMPLE

- In the above image the computer is playing as maximizer X and user is playing as minimizer O
- The scores are assigned -1 for user win,0 for draw and 1 for computer win
- The computer tries to maximize the score and user tries to minimize the score
- After drawing the game tree the computer does not choose the first path which ca lead to user win since path 3 offers the score +1 the computer chooses path 3 and if not possible it chooses path 2 since it leads ot draw

COMPUTATIONAL COMPLEXITY AND LIMITATIONS

The Minimax algorithm faces computational challenges due to its exponential time complexity of O(b^d), where 'b' is the branching factor and 'd' is the depth. This means the computation time grows exponentially with the size of the game tree. Similarly, the space complexity is O(bd) for depth-first search.

ALPHA -BETA PRUNING

Alpha

Alpha: Best value (highest) for Maximizer. It's found along the path.

Beta

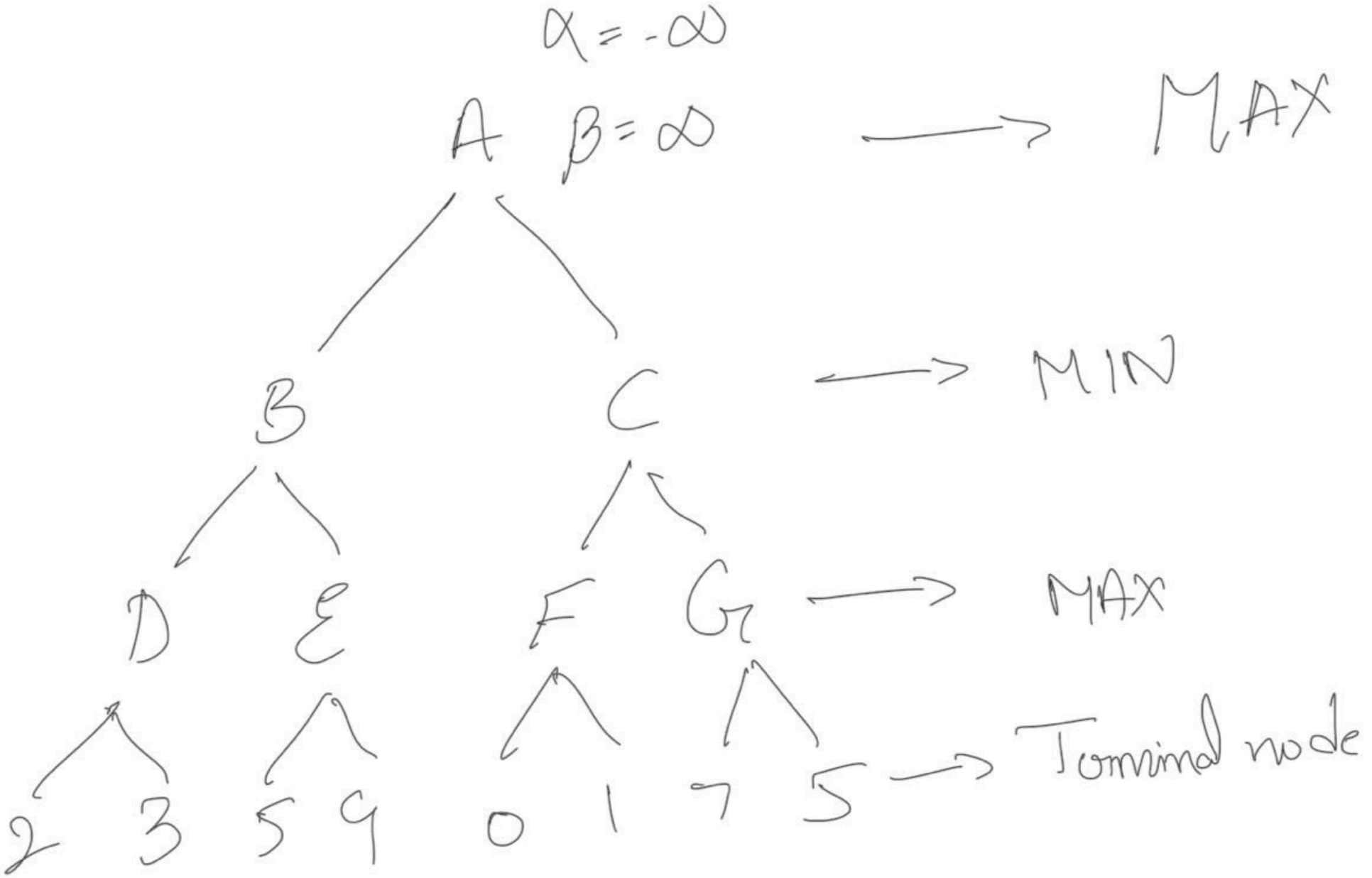
Beta: Best value (lowest) for Minimizer. It's found along the path.

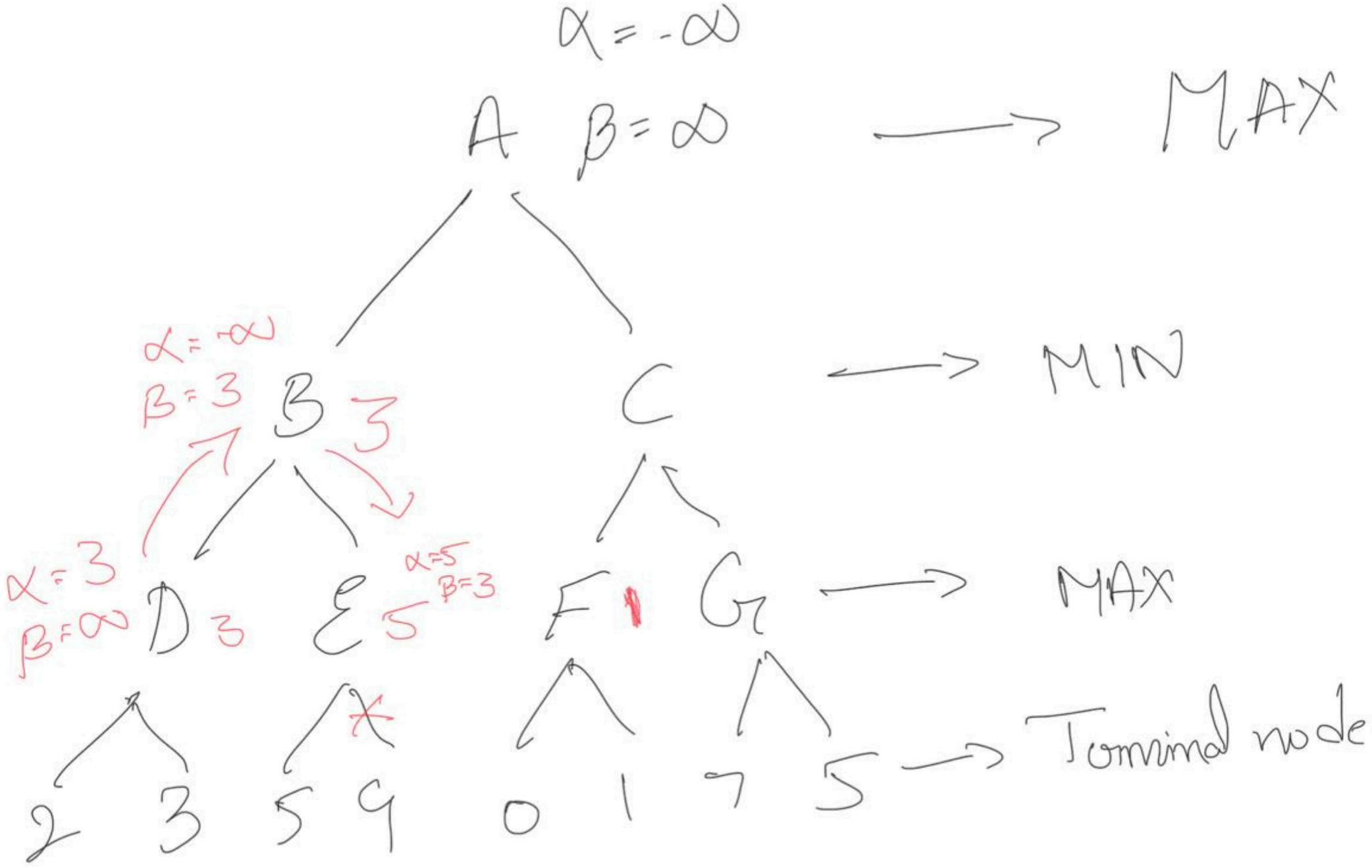
Pruning

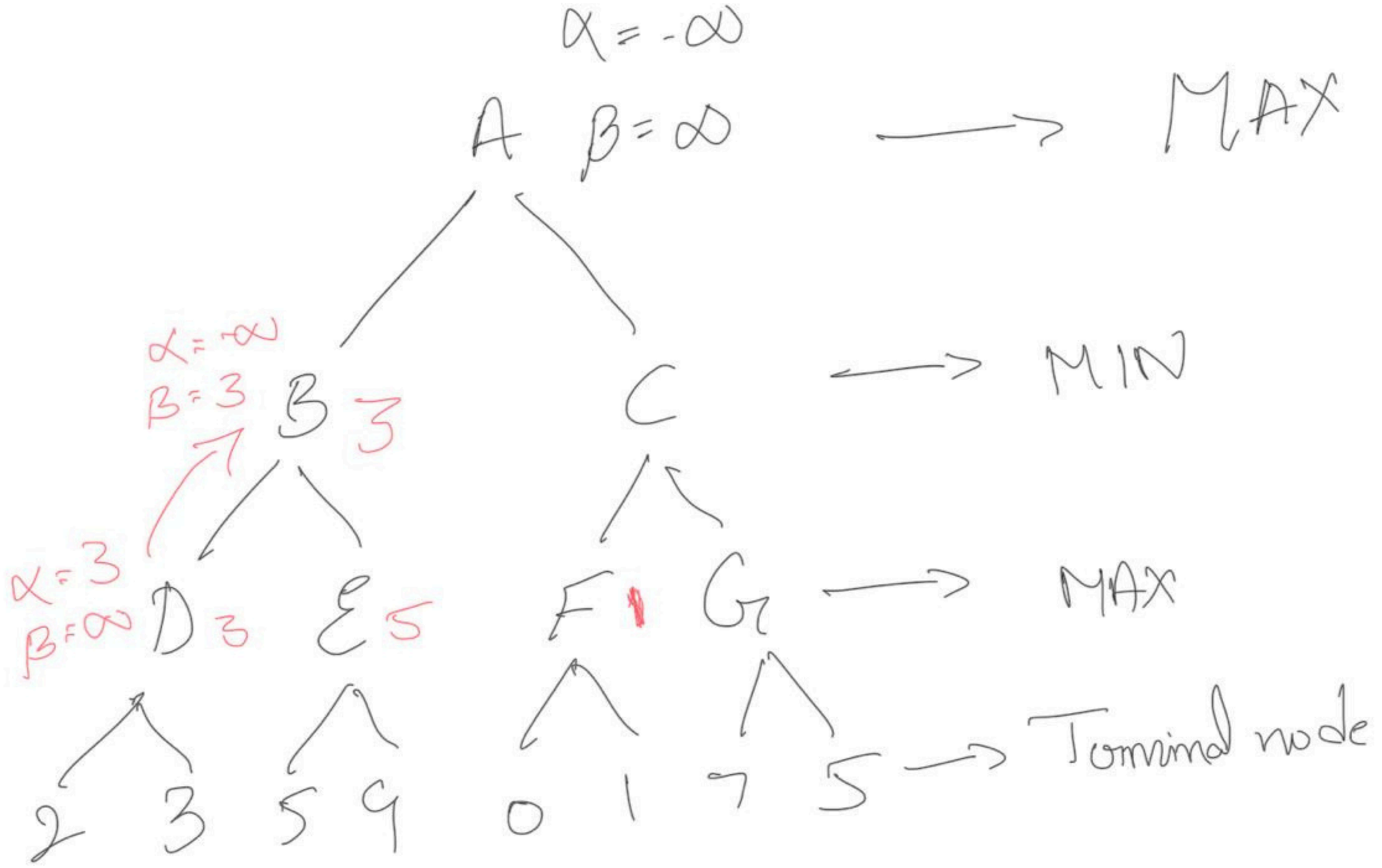
Pruning Condition: Prune a node if alpha is greater than or equal to beta.

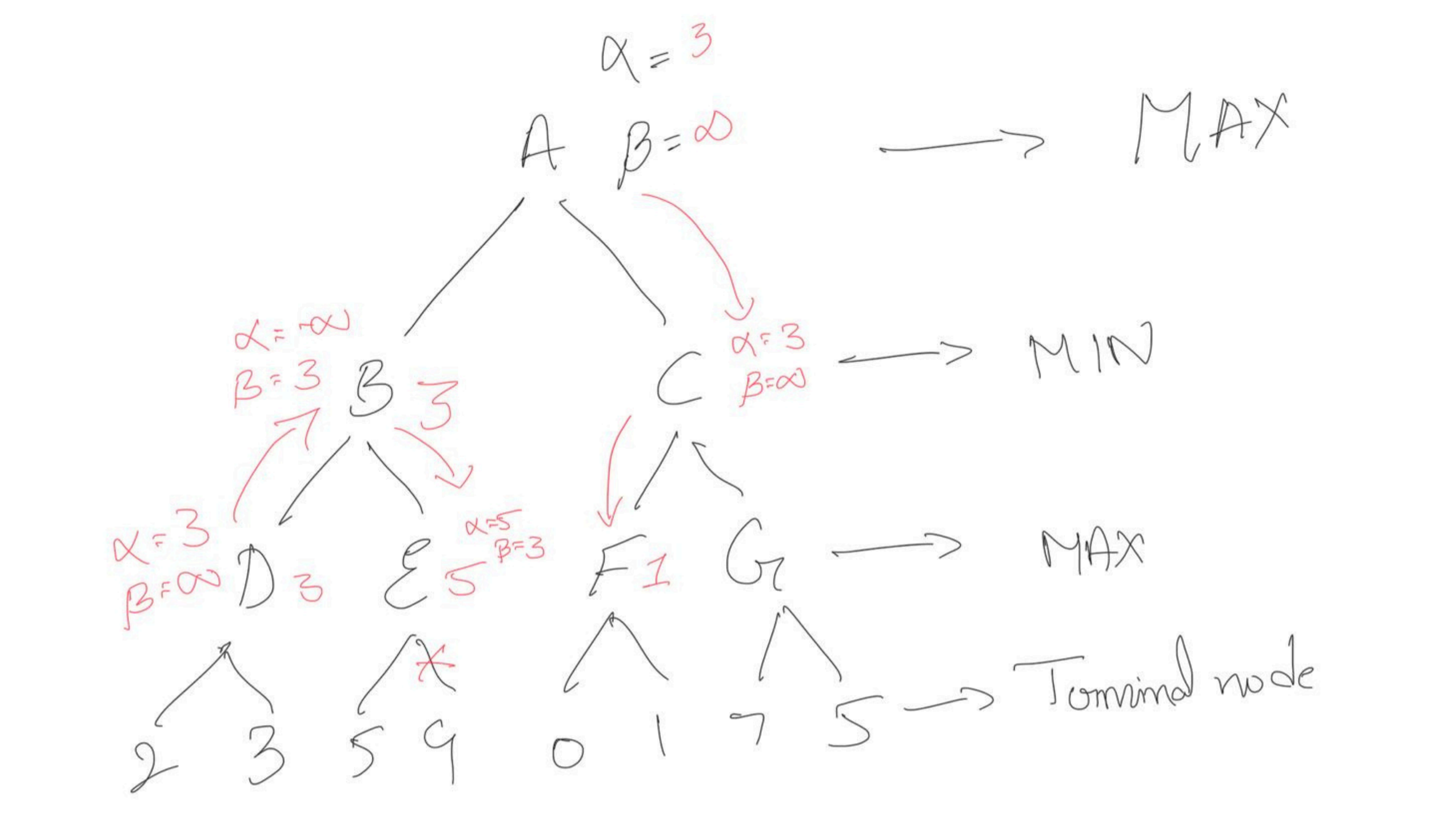
ALGORITHM STEPS

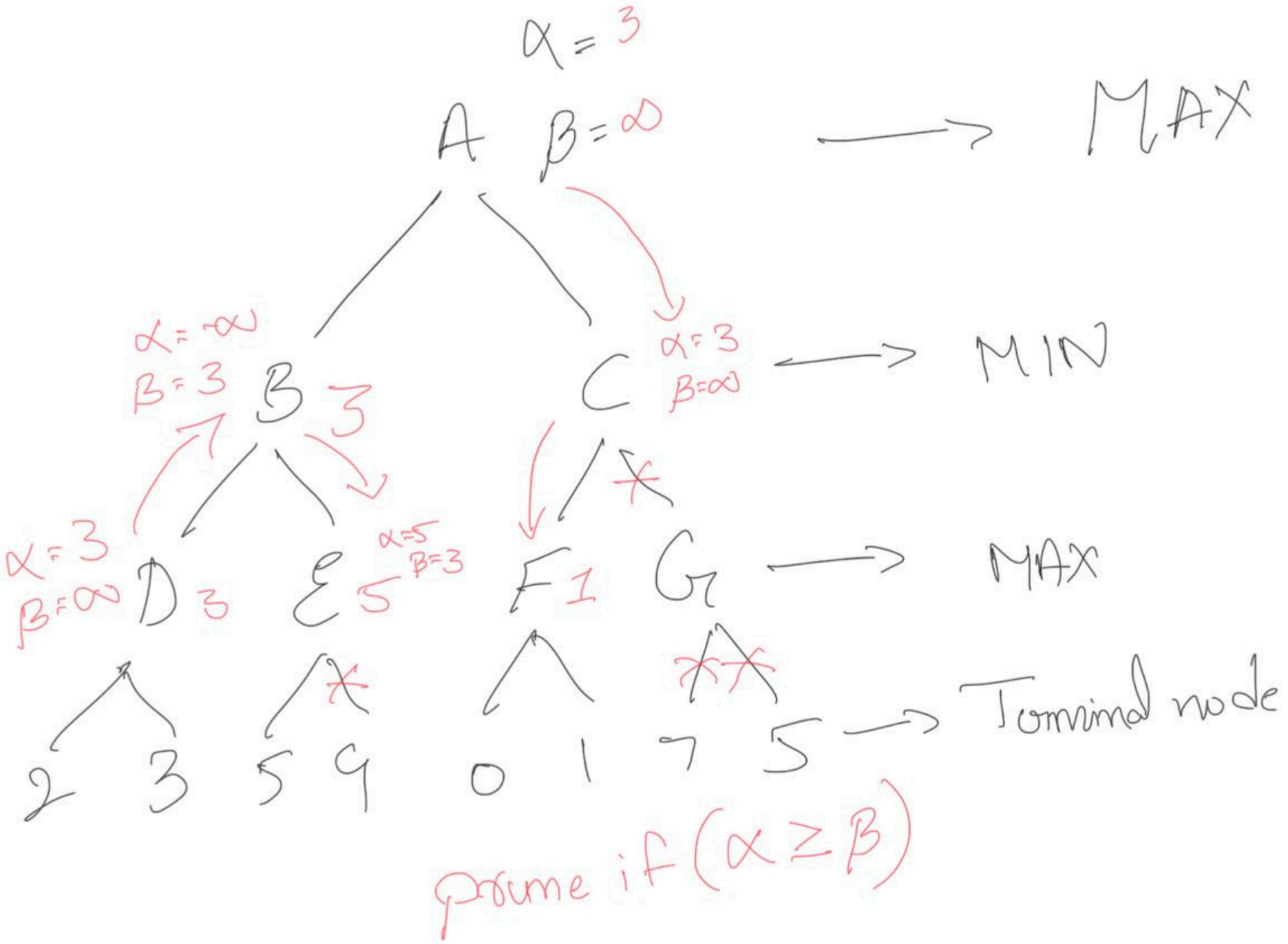
- Initialize the alpha and beta values to -infinity and + infinity
- Traverse the game tree recursively
- Update the alpha and beta values while traversing
- Prune branches when alpha is greater than or equal to beta


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


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


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


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


α = 3
A β = ∞
MAX
α = ∞
β = 3
B 3
C α = 3 β = ∞
MIN
α = 3
β = ∞
D 3
E 5
α = 5
β = 3
F 1
G
MAX
2 3 5 9 0 1 7 5
Terminal node
prune if (α ≥ β)

PROS AND CONS

Advantages

- Reduces search space significantly
- Improves efficiency for deep trees
- Maintains optimal moves

Disadvantages

- Performance is node-order dependent
- Worst-case: no pruning

THANK YOU