Alpha-beta pruning

Alpha-beta pruning is an optimization technique for a minimax algorithm. It reduces computation time by a huge factor, allowing the user to traverse faster and deeper into the tree. It stops evaluating when at least one possibility has been found that typically proves the move to be worse than the previously examined move. The minimax search is based on depth-first search which considers the nodes along a single path in a tree. But Alph-Beta pruning bonds with two major parameters in MAX-VALUE(state, alpha, beta), representing Alpha plays a maximizer role, whereas Beta plays a minimizer role.

- **Alpha** – denotes the value of the best or highest value.
- **Beta** – denotes the value of the best or lowest value.

Why Alpha-beta pruning is considered as an effective algorithm in the adversarial search?

- **Branch elimination:** The Alpha-beta pruning algorithm relies on the branches of the game tree that can be eliminated which promotes the more promising subtree by limiting the search time.
- **Branch and bound:** Alpha-beta pruning is based on the branch and bound method which offers a way to solve the optimization problem by dividing them into sub-problems. It allows the algorithm to effectively remove the branches that did not yield the optimal solution.
- **Heuristic improvements:** To limit the computation time, the heuristic Alpha-beta tree search cuts off the search early and applies the heuristic evaluation function to states that treat the non-terminal nodes as if they were terminal nodes.

Understanding the Alpha-beta pruning: Decision-making in game trees

Game trees can hold abounding branches that can lead to different game states and implementing alpha-beta pruning in such game trees can help to prune away the branches if there already exists a better move. This makes the algorithm computationally efficient since there is no need to search the other branches if there already exists a better move in the game tree.

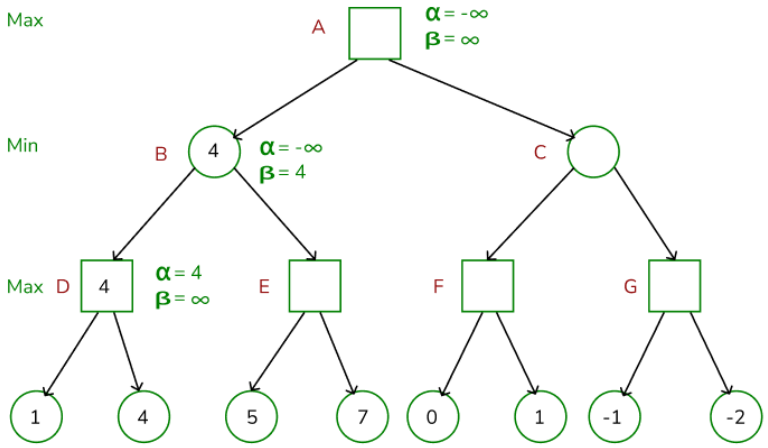
At any node in the game tree, the maximizing player is used to store the maximum score in the alpha which assures that the max node contains only the maximum scores. Similarly, the minimizing player is assured of a minimum value which is stored by the beta. These two values of alpha and beta are updated and maintained by the algorithm.

Take a look at the below example, initially both players begin with the worst possible values they can score, therefore the alpha is assigned with the value of minus infinity, whereas the

beta is assigned with the value of plus infinity. We have a condition to prune the branches, if the alpha is greater than or equal to the beta value then we can prune the branch.

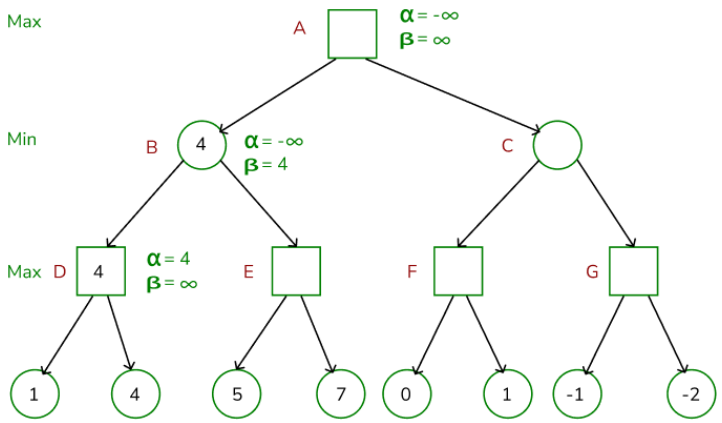
Note: alpha is the maximizer and beta is the minimizer.

Initially, the maximizer node A sets alpha to negative infinity and beta to positive infinity. Assuming that node A selects node B which inherits these values. Node B which is also a maximizer picks node D. At D, the maximizer chooses the max value (4) and then updates its alpha. Upon returning to B (minimizer), it compares its current beta with D's alpha value. If D's alpha is smaller, B's beta value gets updated. In our example, B's beta gets updated since D's alpha value is smaller.



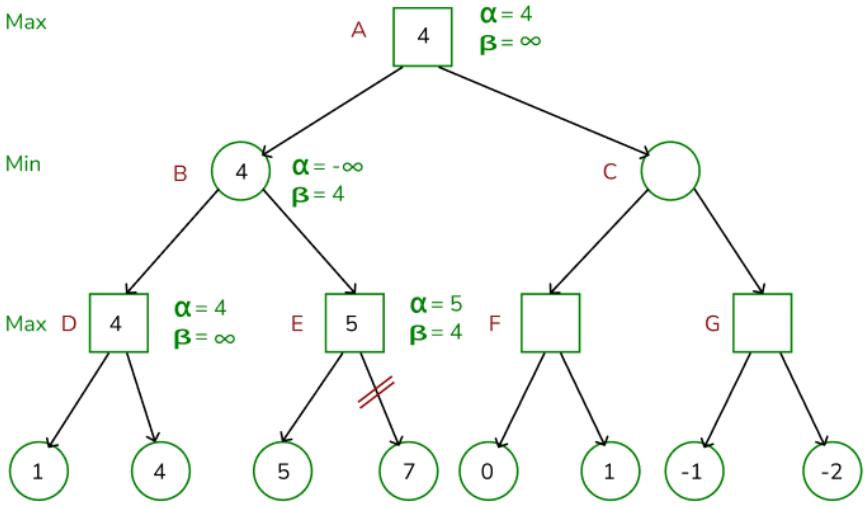
Initial game state

Now, node B selects node E and passes down its current alpha and beta values to node E. Node E inherits the alpha value of negative infinity and beta value of 4. As a maximizer, E explores its left leaf node that holds a value of 5. Since 5 is greater than negative infinity, E's alpha gets updated. Given that alpha is greater than beta, the other leaf node in E is unnecessary to explore so it gets pruned.

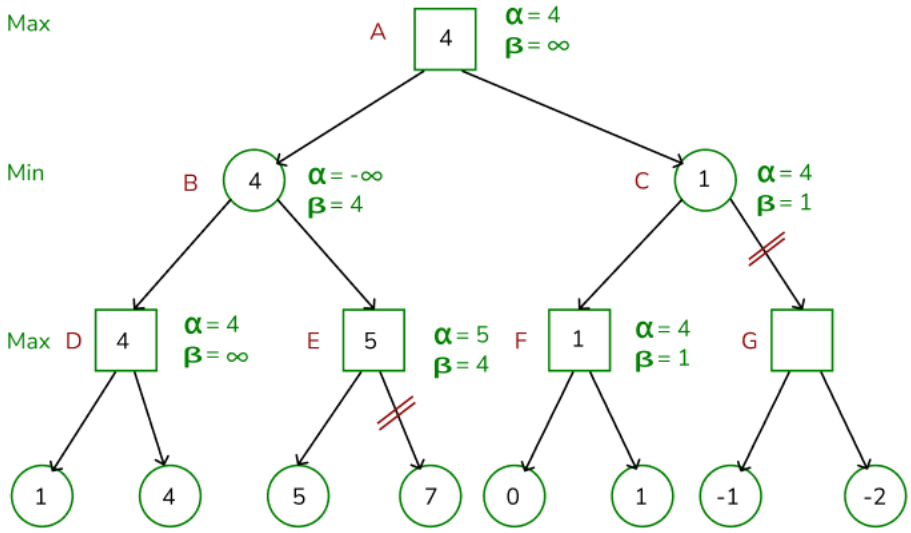


values of B and D were found

At node B, it compares its beta value with node E's alpha, if E's alpha value is lesser than B's beta updates with E's alpha. However, if E's alpha is greater than B's beta, B's beta remains unchanged. In our example, since B's beta is not greater than E's alpha, so B's beta value remains the same. Returning to node A, its alpha value updates with B's beta value because B's beta is greater than A's alpha.



The maximizer yields the value 4



The Maximizer still holds the max value 4 and branch G gets pruned.

At node A, the maximizer's alpha becomes 4 ensuring a value of 4 or higher. A then explores its right child which is node R. At C, alpha is 4 and beta is positive infinity. C then chooses its left child F. Node F inherits the alpha value 4 and beta value positive infinity from the parent node. F's left child yields 0 updating the F's alpha to 4 and also F's right child yields 1. Although F can achieve 1, alpha remains 4. C receives 1 from F which makes the beta 1. As alpha is greater than beta, the entire subtree of node G gets pruned. This pruning logic ensures that, as C is guaranteed 1 or less, it's irrational for A to choose C over B, where A is already guaranteed 4.