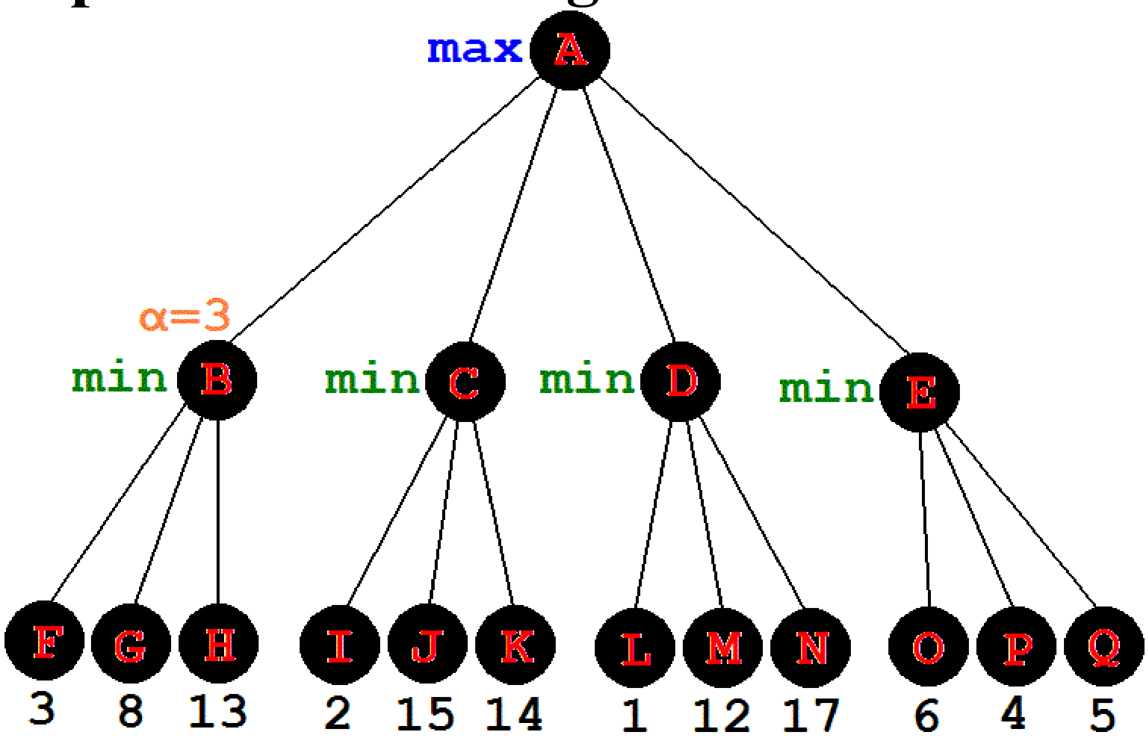
Alpha-Beta Pruning



To understand alpha-beta pruning, consider the situation above. The game is in state **A**. The computer is to make the next move and it has a choice of going to states **B**, **C**, **D**, or **E**. For the computer this means a maximization of state value (of course, because the computer is trying to make the best move).

First, the computer examines state **B**. In state **B** the computer must simulate, or anticipate, its opponent's next move. This is a minimization problem. Since it will look no deeper than **B**'s children in this example, the computer determines a value of **3** for state **B** (the min of **3**,**8**,**13**). Since α always tracks the best **max** value so far, it is set to **3**.

Next, the computer examines state **C**. The first thing it sees is that state **I** has value **2**. Consider for a moment what this means for the game. If the computer were to move to state **B** the opponent would move to state **F**, with value **3**. If the computer were to move to state **C**, the opponent could choose state **I** or worse (remember, we haven't seen states **J** or **K** yet, for all we know they may be worse than **I**). So, from the computer's point of view, there is no point in exploring subtrees **J** or **K** at all. State **B** is already better than **C**, so move on and see if **D** or **E** might be better still. The α value allows you to recall what you've seen before, and prune away parts of the tree that won't matter.

The situation with ß is simply the mirror image of this.

With the above in mind, the pseudocode should become easier to understand:

```
function MIN_VALUE(state, alpha, beta)
    returns:            utility value for state
    inputs:     state   current state of game
                alpha   best max value along path to state
                beta    best min value along path to state
    If depth-limit-reached return utility(state)
    v := infinity
    for all successors, s, of state do
        v := min(v,MAX_VALUE(s,alpha,beta))
        If v <= alpha return v
        beta := min(beta, v)
    return v
```