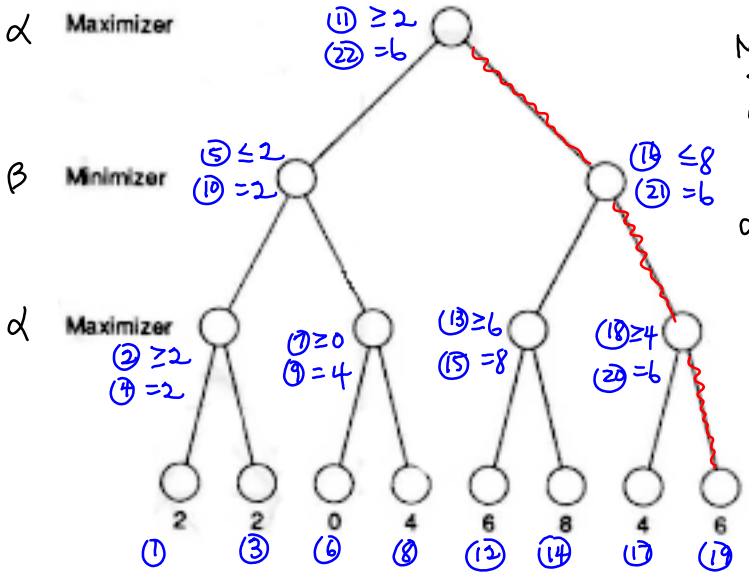
6.034: Game Search Exercise Solutions

Dr. Kimberle Koile

1. Consider the game tree shown below. Explore the tree using the alpha-beta procedure. Indicate all parts of the tree that are cut off, and indicate the winning path or paths. Strike out all static evaluation values that do not need to be computed. (Note: By cut off branches, we mean branches that do not enter into the game at all.)



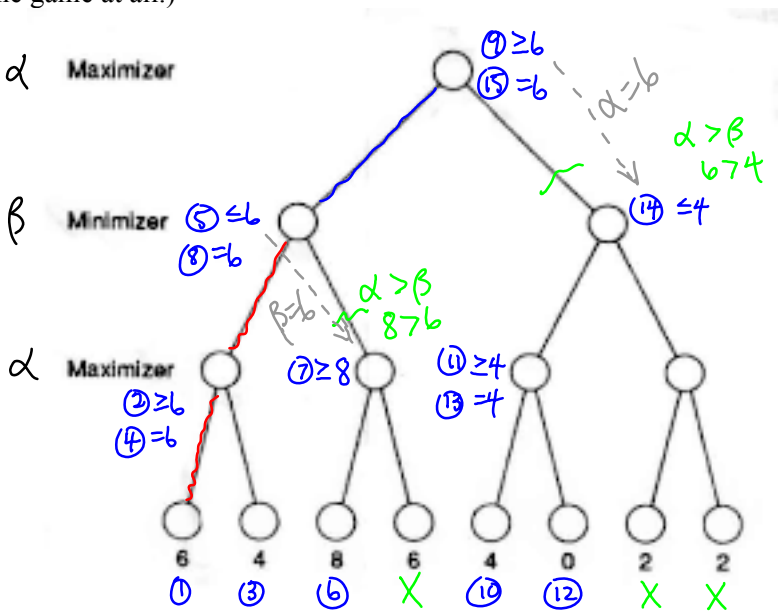
Maximizer and minimizer both stop when lower ≥ upper bound, or when all children have been evaluated.

α = lower, β = upper

Maximizer sets α, which becomes β for minimizer above it.

Minimizer sets β which becomes α for maximizer above it.

2. Now consider the tree shown below, which is a mirror image of the tree shown above. Explore the tree using the alpha-beta procedure. Indicate all parts of the tree that are cut off. Indicate the winning path or paths. Strike out all static evaluation values that do not need to be computed. (Note: By cut off branches, we mean branches that do not enter into the game at all.)



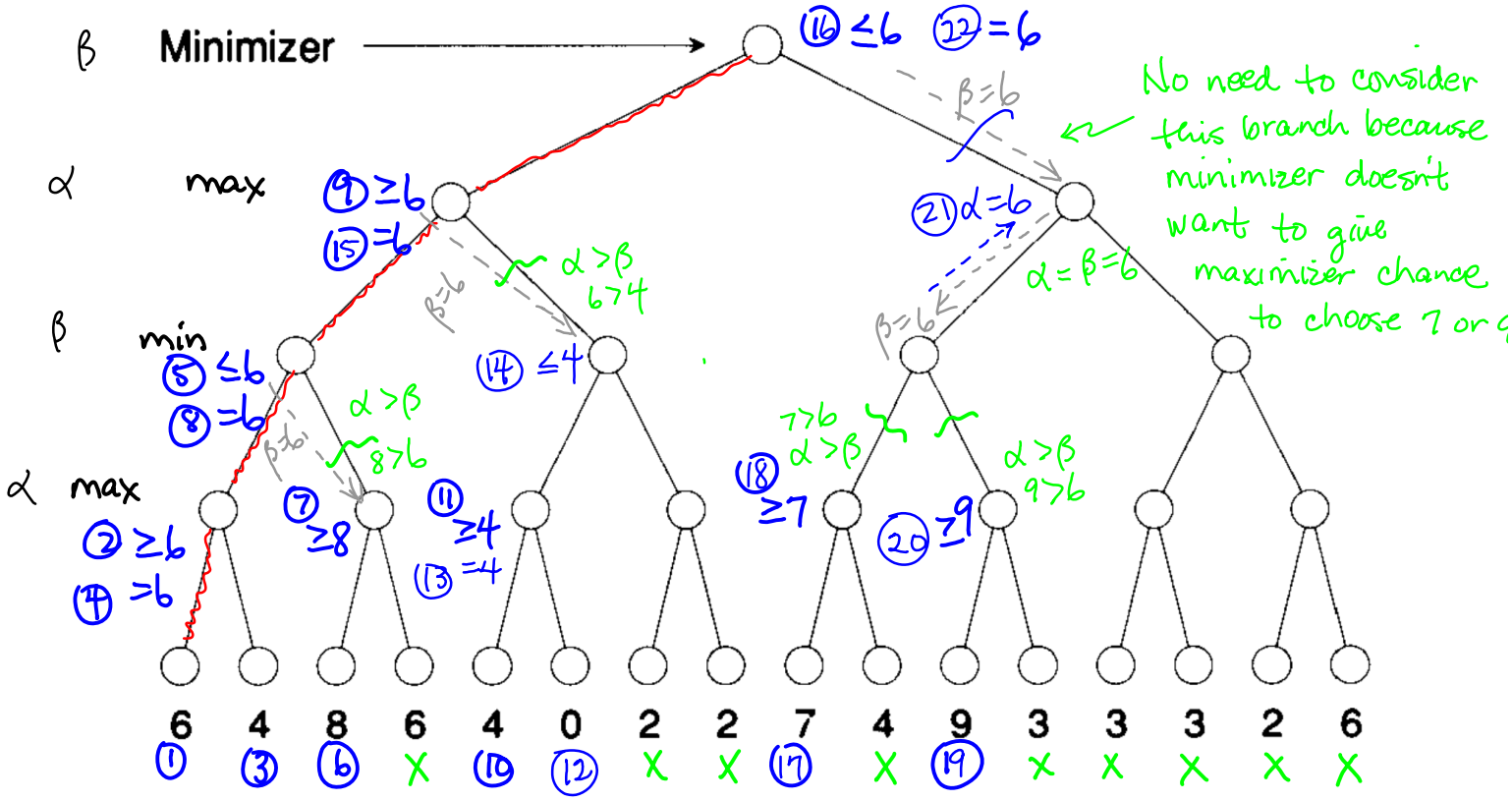
3. Compare the amount of cutoff in the above two trees. What do you notice about how the order of static evaluation nodes affects the amount of alpha-beta cutoff?

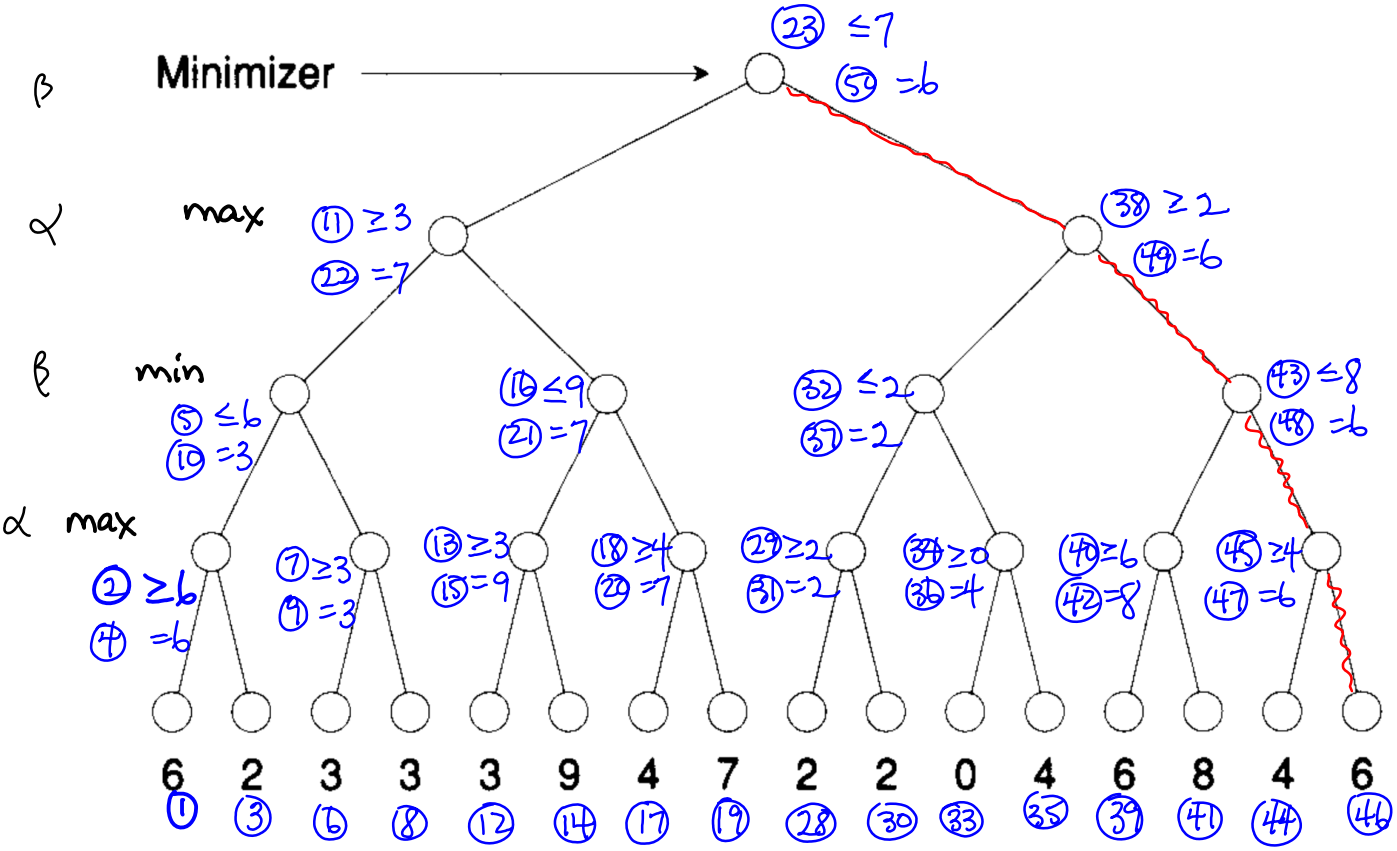
If the evaluated nodes are ordered in the manner described below, then you get maximal alpha-beta cutoff; the opposite order gets no alpha-beta cutoff. For the game trees we've been looking at (i.e., with the bottom row containing the evaluated nodes, and successively higher layers of nodes alternating between minimizer and maximizer, or maximizer and minimizer):

If penultimate level of tree is maximizer level, you get maximal cutoff if descendents of each node in that level are ordered from left to right, max value to min value.
If penultimate level of tree is minimizer level, you get maximal cutoff if descendents of each node in that level are ordered from left to right, min value to max value.

Note that since the goal of alpha-beta pruning is to cut down on the number of nodes that have to be evaluated, game programs don’t actually sort nodes. Two observations that can be made: (1) the performance of alpha-beta is variable and can't be counted on to always outperform minimax; (2) a move generator could attempt to produce new configurations in a sorted order.

4. Below are two game trees (as above, they are mirror images of one another). Perform alpha-beta search on both trees and comment on the effectiveness in each case.



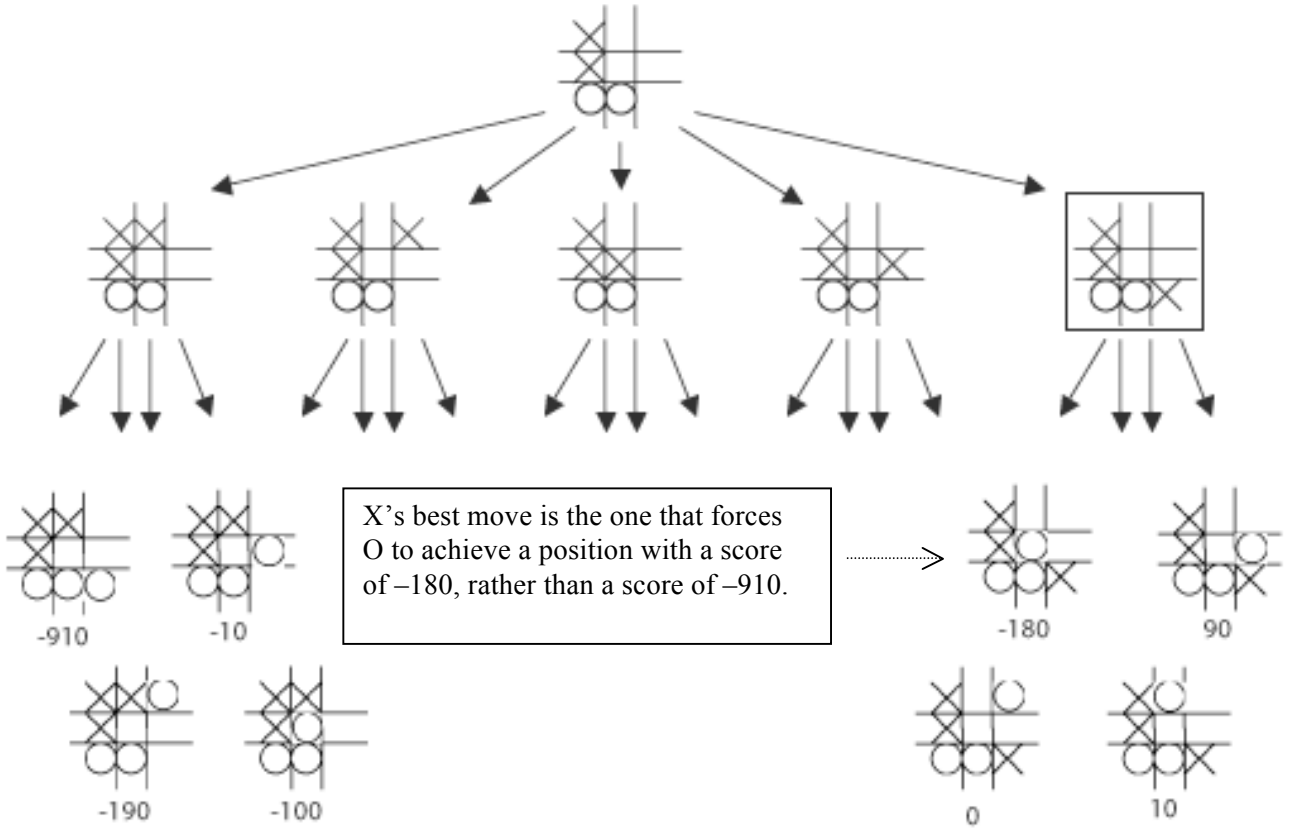


5. Tic-Tac-Toe

You are the X player, looking at the board shown below, with five possible moves. You want to look ahead to find your best move and decide to use the following evaluation function for rating board configurations:

```
value V = 0
do over all rows, columns, diagonals R:
    if R contains three Xs, V = 1000
    else if R contains three Os, V = -1000
        else when R contains only two Xs, V = V + 100
        else when R contains only one X, V = V + 10
        else when R contains only two Os, V = V – 100
        else when R contains only one O, V = V – 10
  end do
return V
```

Draw the four configurations possible from the leftmost and rightmost board configurations below. Use the above static evaluation function to rate the 8 board configurations and choose X's best move. (A reminder: The board configurations that you draw will show possibilities for O's next move.)



6. Consider the game GO. The game is played on a grid with white and black stones placed at the grid intersections. White and black alternate moves, each placing one stone per move. The actual GO board is 19 x 19. Let's consider mini-GO played on a simpler board that is 9 x 9 (81 positions). Estimate the size of the game tree for mini-GO (ignoring the rules for capturing opponent stones). How deep is the search tree?

The game tree would contain on the order of 81! nodes (roughly $10^{120)}$ AND it's 81 ply deep.