Game Trees

Min-Max Trees

When we talk about game trees in cs61b, we mainly talk about min-max trees for zero-sum games.

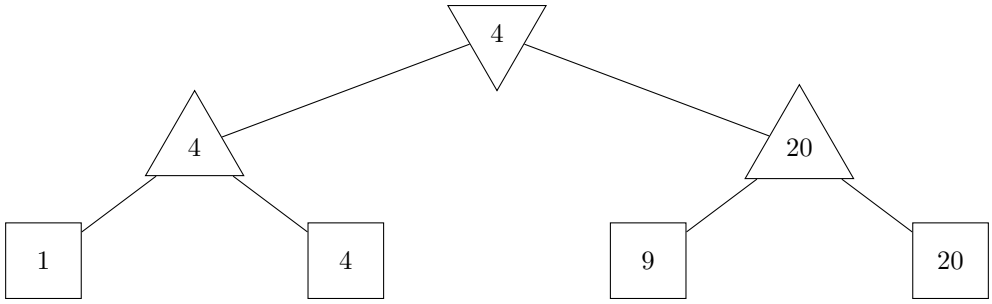
Zero Sum Game: A game in which the benefits for one player are a direct loss to another player. This will usually be a game in which player A is trying to maximize the number of points he gets and player B is trying to minimize the number of points that player A gets.

Terminology

A min-max tree consists of three kinds of nodes.

Right side up Triangle: These are maximizer nodes. They select the *max* value of all its children. These nodes will represent decisions for the maximizer player.
Upside down Triangle: These are minimizer nodes. They select the *min* value of all its children. These nodes will represent decisions for the minimizer player.
Square: These are static nodes. Their values are static and won't change. They will usually be the last level of the three.

Example Min-Max Tree



In this example, we see that the 4 static nodes at the bottom have values 1, 4, 9, and 20. The next level up is the maximizer's turn, so the entire level are maximizer nodes. The level above that is the minimizer's turn, so that entire level (which in this case is only 1 node) are minimizer nodes.
Notice how the values in the triangles are determined by their children. Left maximizer node takes on the value of 4 because 4 is the max value of it's children. Similarly, the right maximizer node takes on the value of 20 because 20 is the max value of it's children. The final minimizer node at the very top then takes the min of it's children (4 and 20) and selects 4.

Purpose

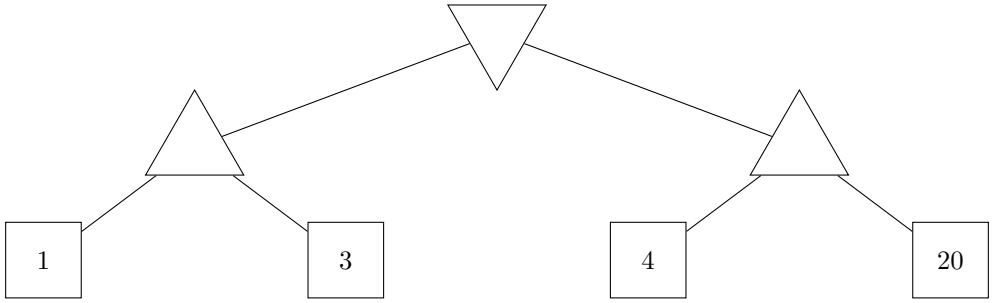
Min-max trees are used in AI because what a min-max tree allows the computer to do is make decisions based on the opponent's most optimal decision. If we make decisions based on the current state of a game, we are making greedy decisions. This is because we only consider our next immediate action. Take chess for example. A good chess player thinks about what the opponent will do if he make some decision and then thinks about how he will respond to the opponent's move. We emulate this in AI with min-max because the maximizer will choose from minimizer nodes, which is the optimal play for the minimizer, and similary for the minimizer. This means that the min-max tree will represent the **MOST OPTIMAL** actions that each side can take.

Alpha Beta Pruning

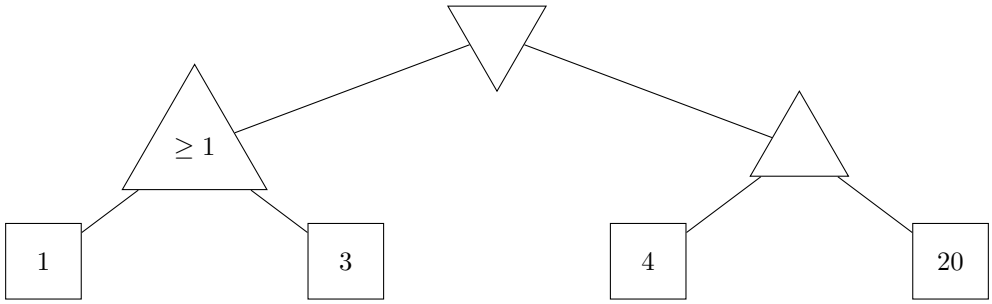
Min-max trees are **VERY** expensive to construct. They take up exponential space because each level has children equal to the number of decisions that player can make. It is very impractical to compute out all of the nodes in the min-max tree, so we introduce alpha beta pruning to prune out branches of nodes that we know we won't even need to check.

Pruning Process

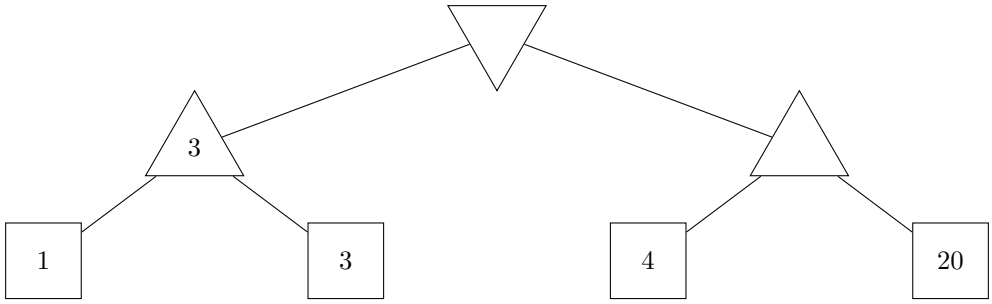
When we analyze our min-max tree, we analyze from left to right and we only have knowledge of the static nodes. So we could be given a tree that looks like this.



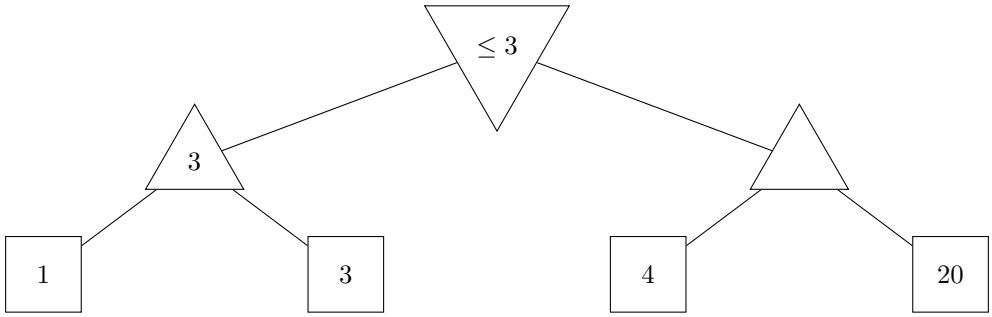
We start with the left most node and see that it is a 1. From this information, we know that the left maximizer node is atleast 1. I like to take note of this with ≥ 1



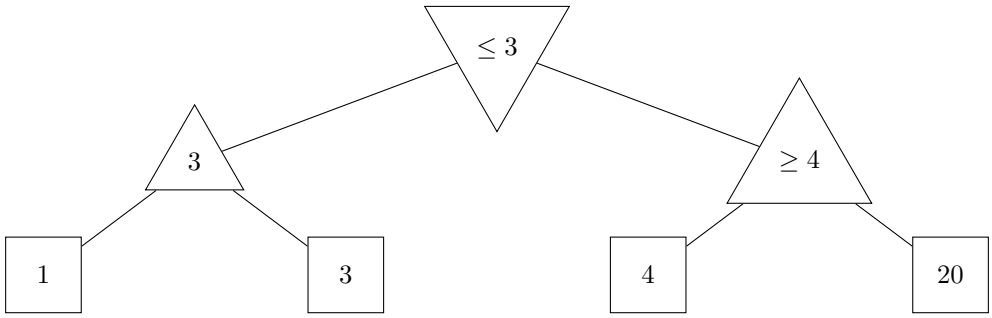
We then process the next static node and see that it is 3. We see that $3 > 1$, so we update the maximizer node to be 3. We set it's value because we know that maximizer node has no more children.



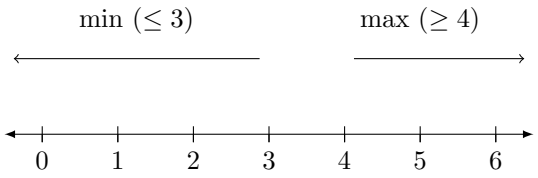
At this point, before moving onto the next static node, we know the left maximizer must have a value of 3, so this actually tells us information about the final minimizer node. From this information, we can say that the minimizer node is at most 3 (≤ 3).



Moving onto the next static node, we see that it has a value of 4, so we update the maximizer node with that information (≥ 4).

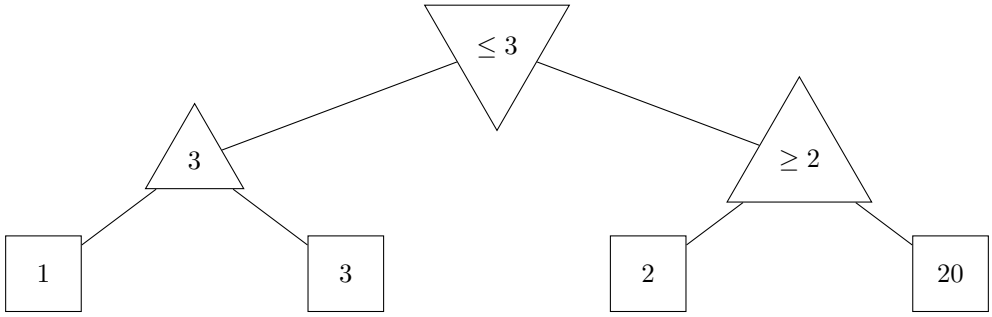


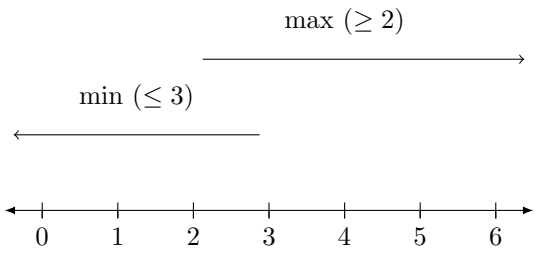
At this point, you should notice something interesting about the conclusions we have made about the minimizer and maximinzer nodes. We see that the minimizer node is at most 3, but the maximizer child is at least 4. From this information, we can conclude that the minimizer child will actually never select what is in the right maximizer node. Here is a visualization to help.



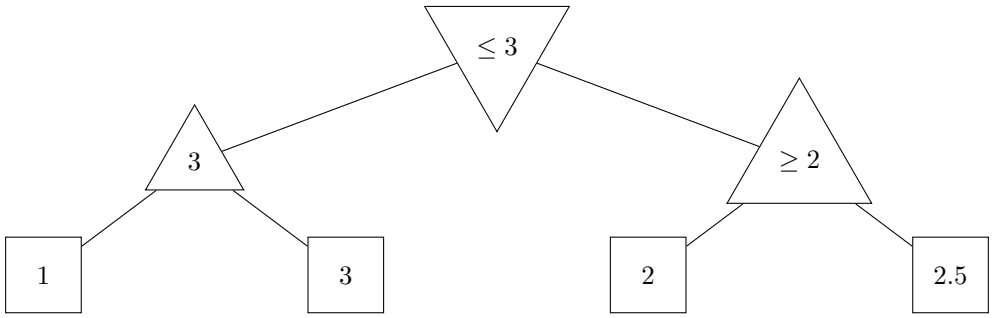
If we observe this number line, we see that the two ranges for the nodes do not cross or overlap. This means that whatever value the right maximizer node selects, it will never be within the range of possible values of the minimizer node, so we can then prune out all remaining children of the maximizer node. This means that we can prune the node of value 20 and not look at at it. (Pruning is done by "X"ing out the branch to the 20)

Now consider if the third node had a value of 2. Then our tree and number line would look like this.





What you will notice about this number line is that the two ranges actually overlap. This means the last node cannot be pruned because if the last node has a value in between the two ranges. That is what the minimizer will select. In the current example, the last node is 20, so the maximizer will choose 20, and the minimizer will ultimately choose 3. However, consider if the value of the last node is between 2 and 3, so like 2.5. The tree would look like this.



The maximizer will choose 2.5 and the minimizer will not choose the 3, but actually the newly selected 2.5. Since this is a possible case, we cannot prune the last node without checking it.

This is a small example of alpha beta pruning, but this process can be done for **ANY** min-max tree with any number of children and levels. Just apply these same concepts for a larger tree and you will be able to figure out which branches can be pruned.

TL;DR

1. Types of Nodes
   (a) **Right side up triangle:** maximizer node
   (b) **Upside down triangle:** minimizer node
   (c) **Square:** static node
2. Process
   (a) Start at bottom and work to top
   (b) Maximizer nodes select max of children
   (c) Minimizer nodes select min of children
3. Alpha Beta Pruning
   (a) Work left to right
   (b) Maximizer nodes are at least some value (≥)
   (c) Minimizer nodes are at most some value (≤)
   (d) If minimizer and maximizer ranges don't overlap, you can prune the next branches
   (e) If you processed all children of node, you can set that node to a specific value (max/min of children) instead of a range