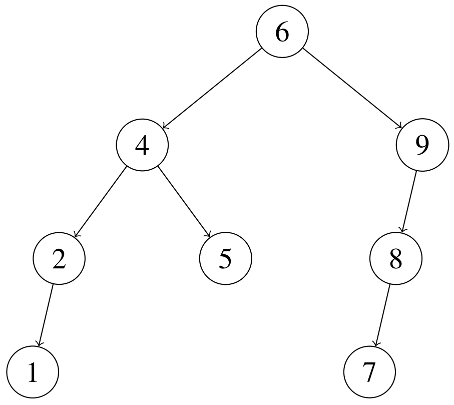
CS 61B
Spring 2020

Small Group Tutoring
Section 7: Trees

Worksheet 9

1 Tree-versal



a) What is the pre-order traversal of the tree? **Solution:** 6 4 2 1 5 9 8 7

b) What is the post-order traversal of the tree? **Solution:** 1 2 5 4 7 8 9 6

c) What is the in-order traversal of the tree? **Solution:** 1 2 4 5 6 7 8 9

d) What is the breadth-first traversal of the tree?
Solution: 6 4 9 2 5 8 1 7

2 Runtime

Provide the best case and worst case runtimes in theta notation in terms of N, and a brief justification for the following operations on a binary search tree. Assume N to be the number of nodes in the tree. Additionally, each node correctly maintains the size of the subtree rooted at it. [Taken from Final Summer 2016]

boolean contains(T o); //Returns true if the object is in the tree

Solution: Best: $\Theta(1)$ Why: If the object is at the root.

Worst: $\Theta(N)$ Why: If the object is at the leaf of a spindly tree.

void insert(T o); //Inserts the given object.

Solution: Best: $\Theta(1)$ Why: One example may be inserting to the left child of the root of a right leaning spindly tree.

Worst: $\Theta(N)$ Why: One example may be inserting to the leaf node of a right leaning spindly tree.

T getElement(int i); //Returns the ith smallest object in the tree.

Solution: Best: $\Theta(1)$ Why: One example may be if i = 1 and the tree is a very spindly right leaning tree.

Worst: $\Theta(N)$ Why: One example may be if i = N and the tree is a very spindly right leaning tree.

3 Pruning Trees

Assume we have some binary search tree, and we want to prune it so that all values in the tree are between *L* and *R*, inclusive. Fill out the method below that takes in a BST, as well as *L* and *R*, and returns the pruned tree. Note that the root of the original tree might not be between *L* and *R*, so make sure you return the root of the new pruned tree.

```
class BST {
    int label;
    BST left; // null if no left child
    BST right; // null if no right child
}

public BST pruneBST(BST root, int L, int R) {
    if (root == null) {
        return null;
    } else if (root.label < L) {
        return pruneBST(root.right, L, R);
    } else if (root.label > R) {
        return pruneBST(root.left, L, R);
    }
    root.left = pruneBST(root.left, L, R);
    root.right = pruneBST(root.right, L, R);
    return root;
}
```

4 Game Trees

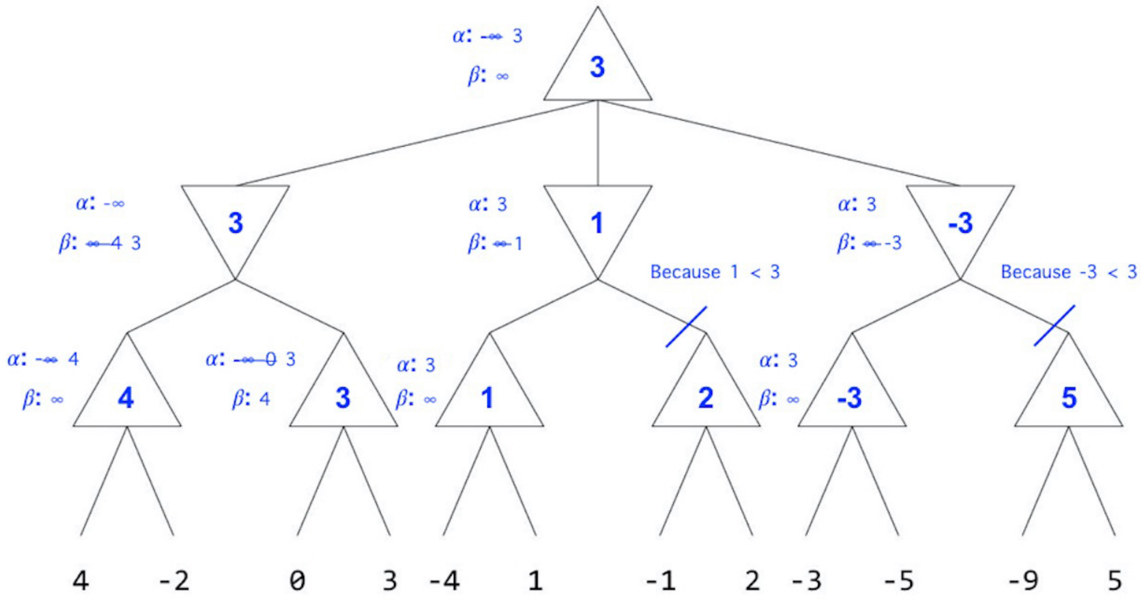
Minimax is an algorithm that allows a computer to calculate the best move to make in a game, assuming the opponent plays optimally. At the bottom of the game tree are integer scores, where a higher score indicates a more favorable outcome for the player of interest (who wants to win). Game trees could grow exponentially large in size if the algorithm were to consider the eventual outcome of every possible move that could be made. Thus, in addition to setting a depth limit, to simplify computation, we employ alpha-beta pruning.

Here's how it works. As the computer searches through the game tree via a depth-first traversal, it passes as arguments to each recursive call an alpha (α) and beta (β) value. α is the highest value found so far by a maximizing node along the path from root to leaf. β is the lowest value found so far by a minimizing node along the path from root to leaf. Initially, α is set to negative infinity and β set to positive infinity.

At a maximizing node, we update its value from each branch and then ask if the current value is greater than β. If so, we stop searching the other branches of this maximizing node (in other words, prune the other branches) and move back up in the tree. Since the parent of the current node is a minimizer, the parent would choose the β value instead of any further updates to the maximizer node, as those updates would be even greater and therefore ignored.

Similarly, whenever we are at a minimizing node, we look at its value and ask if the value is smaller than α. If so, we stop searching the other branches of this minimizing node (prune them) and move back up in the tree. Since the parent of the current node is a maximizer, the parent would choose the α value instead of any further updates to the minimizer node, as those updates would be even smaller and therefore ignored.

Consider the game tree below. The upside down triangles represent minimizer nodes and the normal triangles represent maximizer nodes.



1. Fill out the values in each maximizer and minimizer node for the above game tree after applying the Minimax algorithm to it.

2. Cross out (with an X) all branches that would be pruned by a Minimax implementation that utilizes alpha-beta pruning. **Solution:**

   α and β values are updated every time a number returned from a child is either greater than α (for maximizer nodes) or less than β (for minimizer nodes).
   Pruning occurs between 1 and 2 in the center minimizer node because once the minimizer's value is updated to 1, 1 is less than α which is 3, so we know that the maximizer at the root will not choose 1 or anything less than 1 that the minimizer might return so we can prune the right child of the minimizer. We also prune between -3 and 5 in the rightmost minimizer for the same reason because $-3 < 3$.

   Here is a very nice step-by-step walkthrough of traversing and updating the values of nodes in a game tree, prepared by CS188 at Berkeley: https://www.youtube.com/watch?v=xBXHtz4Gbdo

3. According to the Minimax algorithm, which move should we make for the above game?
   Solution: Go to the left sub-branch

4. Minimax assumes that the opponent is playing optimally as well. Is this always a good idea? Consider the case where we are playing a 3-year old who is not familiar with the game. Would it be possible to use this to our advantage and win in less moves?

Solution: Minimax is, assuming you can generate the full tree, guaranteed to be optimal. However, it is a pessimistic algorithm because of the assumptions it makes regarding its opponent. Since 3-year-olds are not masters of strategy, we might want to opt for a potentially riskier move that has a chance of a higher reward if our opponent makes a suboptimal move. So, it is definitely possible to win in less moves if the opponent makes suboptimal moves.