Binary Trees

Discussion 09

Announcements

- Weekly Survey due Tuesday 03/14
- Project 2 Checkpoint due Friday 03/18
- Lab 9 due Friday 03/18
- Next week is Spring Break!

Review

Trees

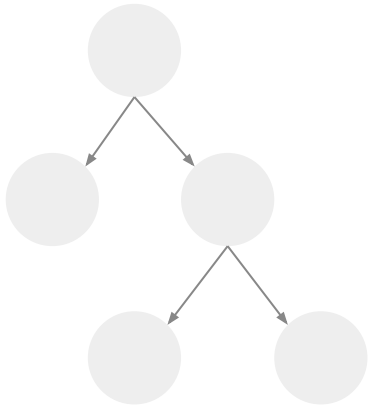
Trees are structures that follow a few basic rules:

1. If there are N nodes, there are N-1 edges
2. There is exactly 1 path from every node to every other node
3. The above two rules means that trees are fully connected and contain no cycles

A parent node points towards its child.

The root of a tree is a node with no parent nodes.

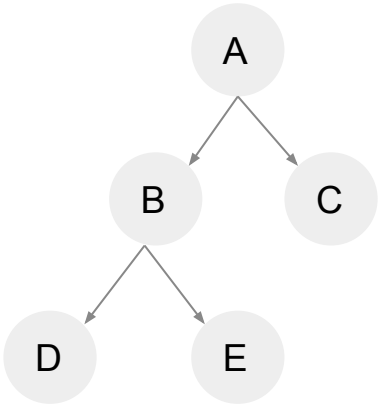
A leaf of a tree is a node with no child nodes.



Breadth First Traversal

In a **Breadth First Traversal (BFS)** we visit nodes based off of their distance to the source, or starting point. For trees, this means visiting the nodes of a tree level by level.

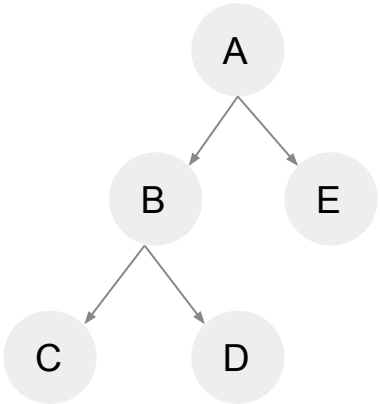
BFS is usually done using a queue.



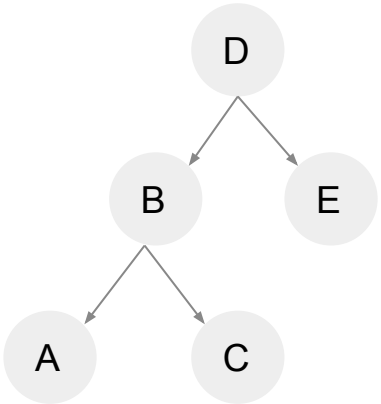
Depth First Traversal

In a **Depth First Traversal (DFS)** we visit each subtree in some order recursively.

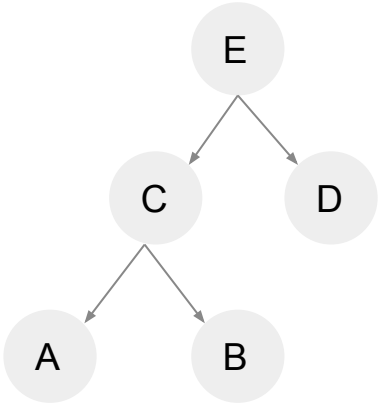
DFS is usually done using a stack.



Pre-order traversals visit the parent node before visiting child nodes.



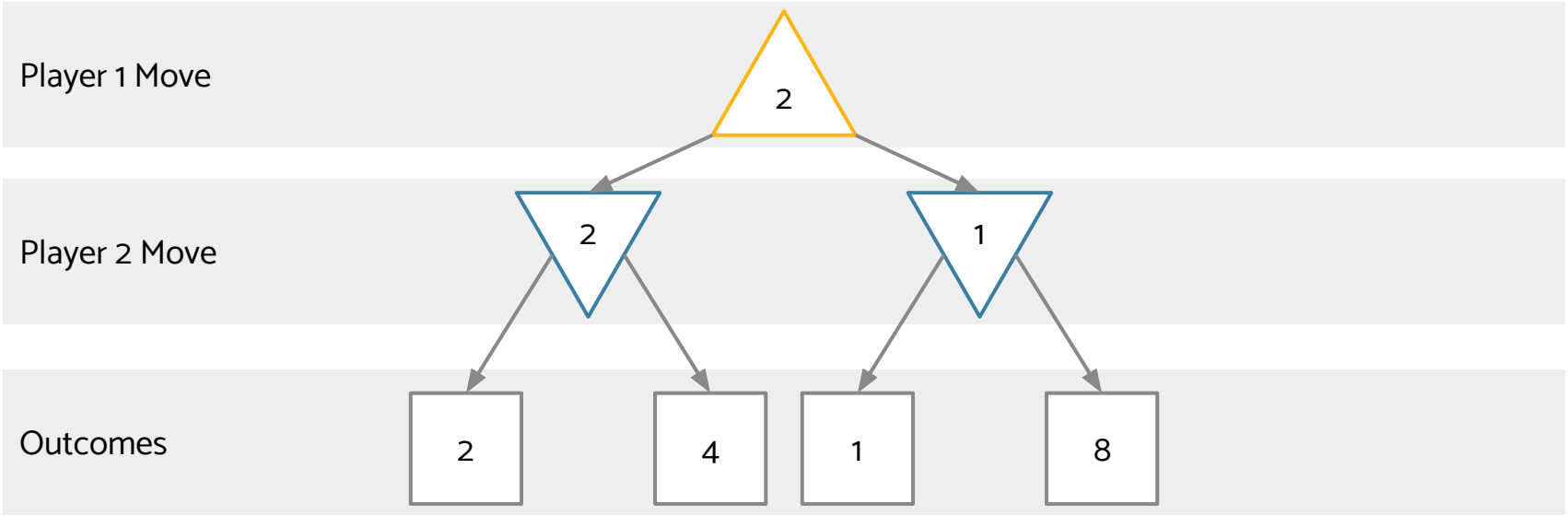
In-order traversals visit the left child, then the parent, then the right child.



Post-order traversals visit the child nodes before visiting the parent nodes

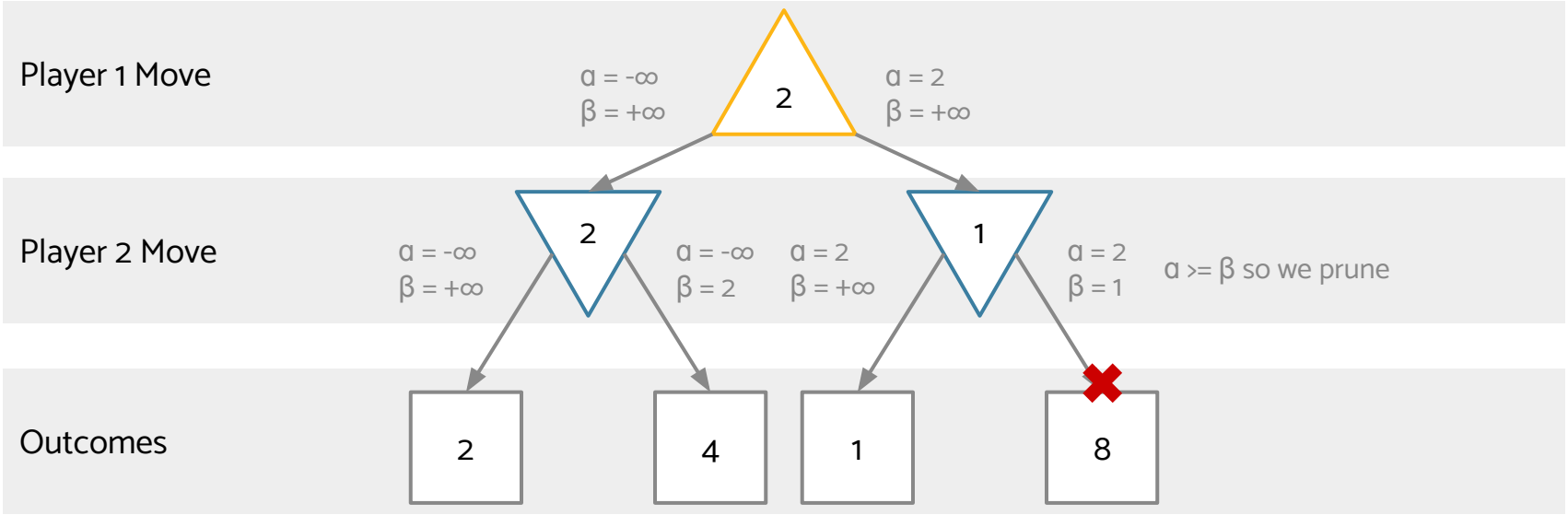
Game Trees

Game Trees or min-max trees allow us to showcase the outcomes of a two-player game where one person is trying to maximize the total score and one person is trying to minimize the total score.



Alpha-Beta Pruning

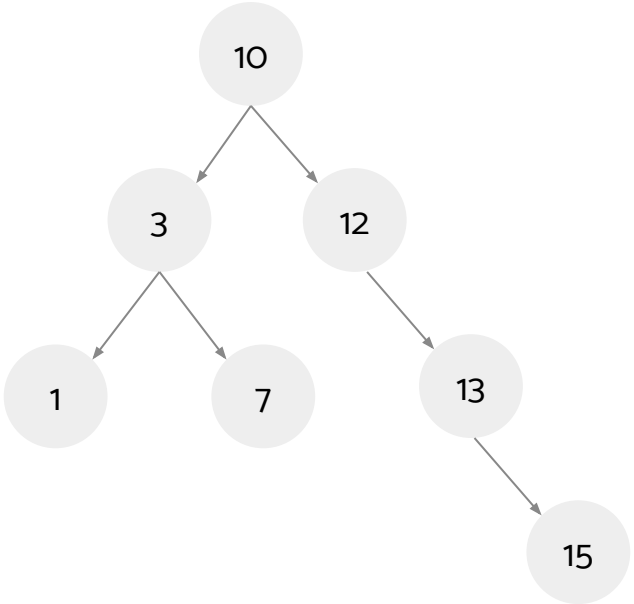
Alpha-Beta Pruning allows us to reduce the number of nodes we need to visit to determine the best possible move for player 1, since game trees get combinatorially large.



Worksheet

1 Law and Order

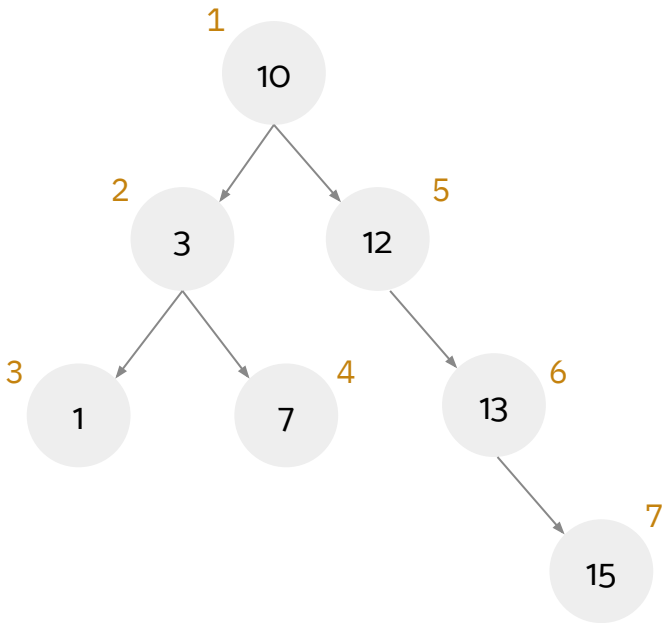
Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals for this BST.



1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals for this BST.

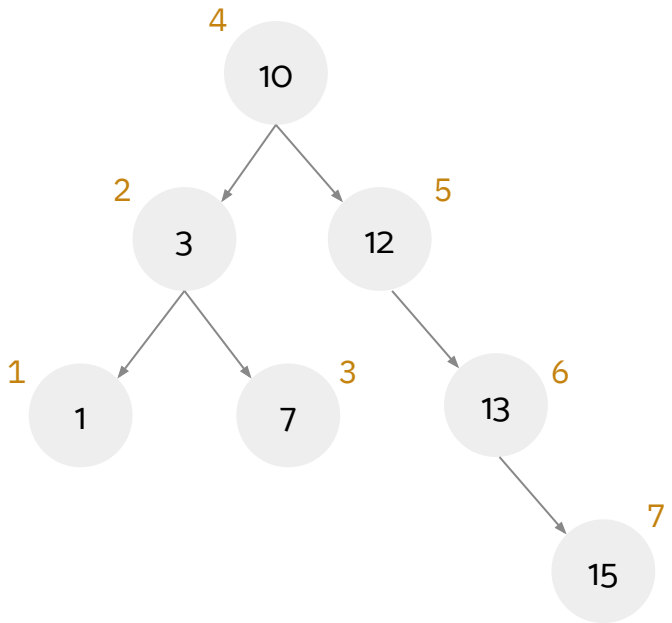
```
DFS Pre-Order:
10, 3, 1, 7, 12, 13, 15
```



1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals for this BST.

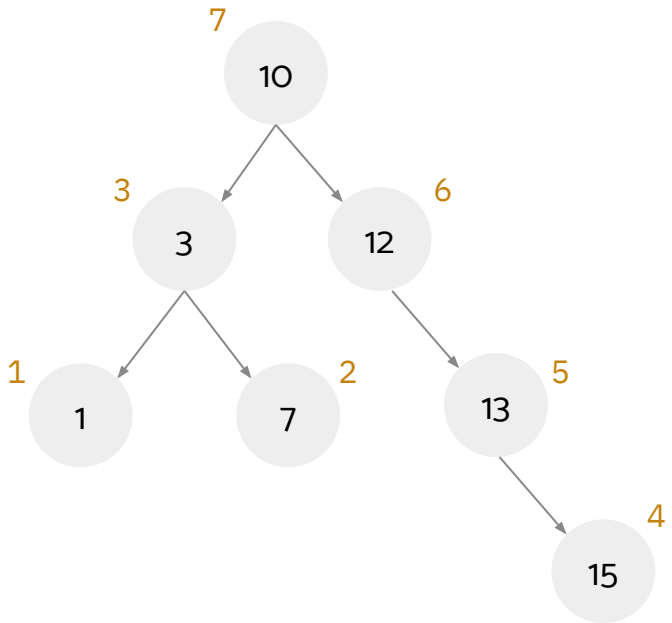
```
DFS In-Order:
1, 3, 7, 10, 12, 13, 15
```



1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals for this BST.

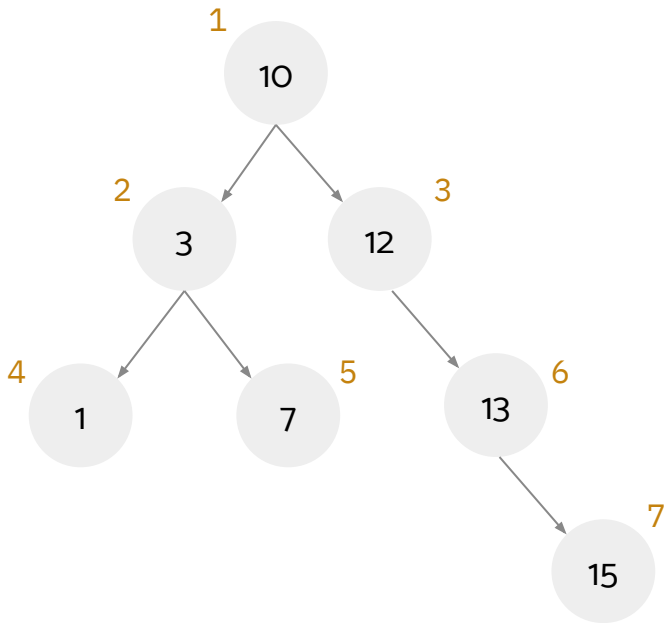
```
DFS Post-Order:
1, 7, 3, 15, 13, 12, 10
```



1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals for this BST.

```
BFS Level Order:
10, 3, 12, 1, 7, 13, 15
```



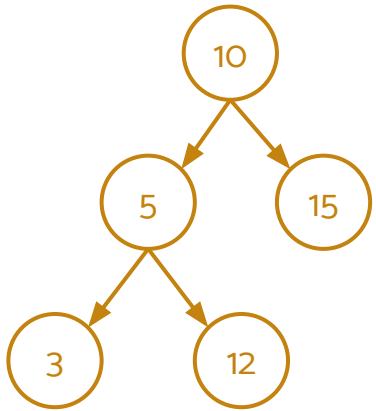
2A Is This A BST?

```
public static boolean brokenIsBST(TreeNode T) {
     if (T == null) {
          return true;
     } else if (T.left != null && T.left.val > T.val) {
          return false;
     } else if (T.right != null && T.right.val < T.val) {
          return false;
     } else {
          return brokenIsBST(T.left) && brokenIsBST(T.right);
     }
}
```

Give an example of a binary tree for which `brokenIsBST` fails.

2A Is This A BST?

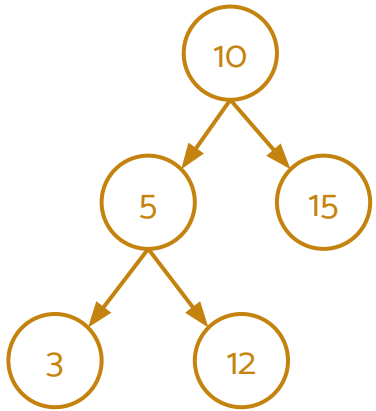
```
public static boolean brokenIsBST(TreeNode T) {
     if (T == null) {
          return true;
     } else if (T.left != null && T.left.val > T.val) {
          return false;
     } else if (T.right != null && T.right.val < T.val) {
          return false;
     } else {
          return brokenIsBST(T.left) && brokenIsBST(T.right);
     }
}
```



Give an example of a binary tree for which `brokenIsBST` fails.

2A Is This A BST?

```
public static boolean brokenIsBST(TreeNode T) {
     if (T == null) {
          return true;
     } else if (T.left != null && T.left.val > T.val) {
          return false;
     } else if (T.right != null && T.right.val < T.val) {
          return false;
     } else {
          return brokenIsBST(T.left) && brokenIsBST(T.right);
     }
}
```



Give an example of a binary tree for which `brokenIsBST` fails.

This method fails whenever a "grandchild" has a value that is too high or low since the method only compares parents and children. brokenIsBST would return true for this given tree.

2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(_____________________________________);
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {









}
```

2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE); // Start with root
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {









}
```

2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {
     if (T == null) { // If T is null, then its a BST (also handy as a base case)
          return true;
     }




}
```

2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {
     if (T == null) {
          return true;
     } else if (T.val < min || T.val > max) {
          return false;
     }



}
// We can use min and max to make sure that we don't run into the same problem as last time
```

2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {
     if (T == null) {
          return true;
     } else if (T.val < min || T.val > max) {
          return false;
     } else {
          return isBSTHelper(T.left, min, T.val) && isBSTHelper(T.right, T.val, max);
     }
}
// Update min and max appropriately for the left and right child
```

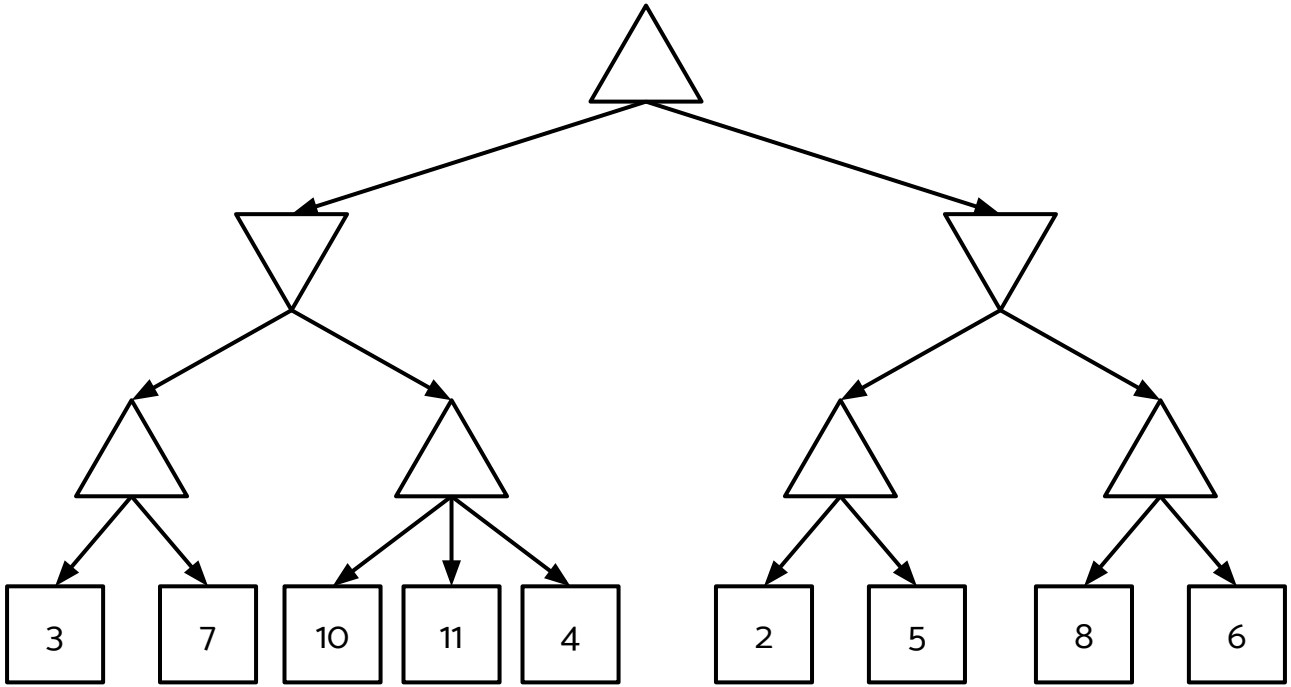
2B Is This A BST?

Write isBST such that it fixes the error from part A.

```
public static boolean isBST(TreeNode T) {
     return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}
public static boolean isBSTHelper(TreeNode T, int min, int max) {
     if (T == null) {
          return true;
     } else if (T.val < min || T.val > max) {
          return false;
     } else {
          return isBSTHelper(T.left, min, T.val) && isBSTHelper(T.right, T.val, max);
     }
}
```

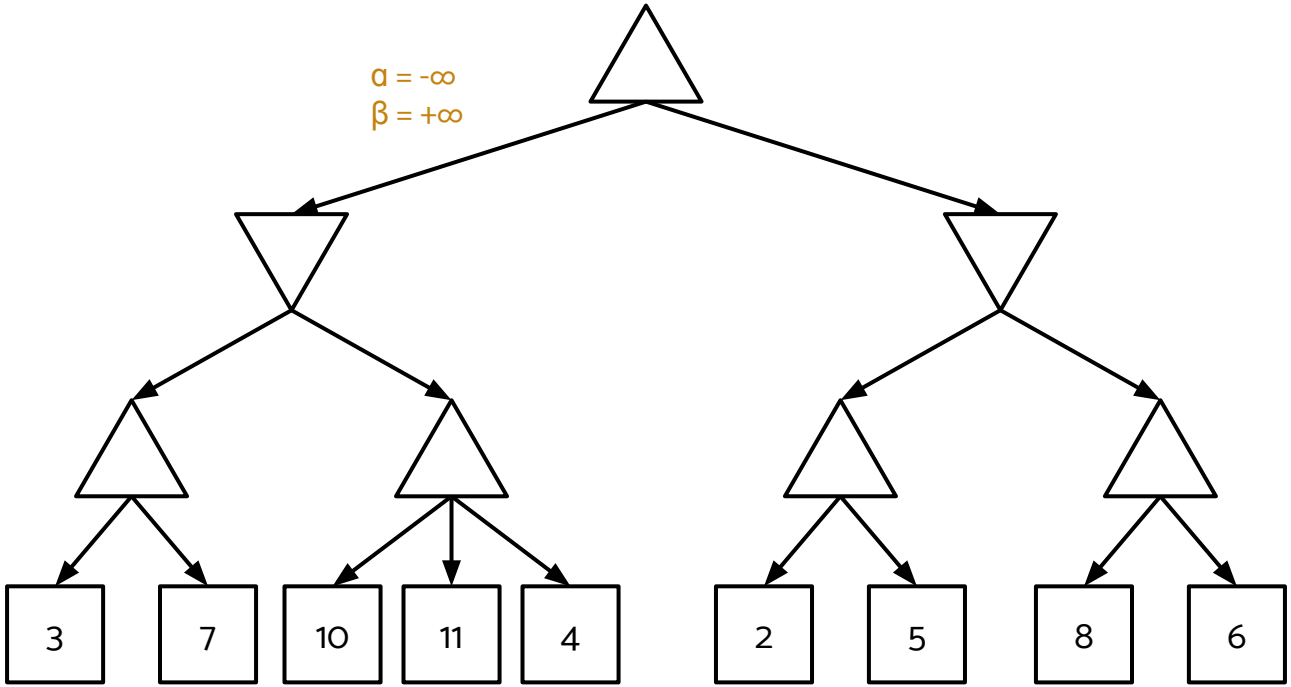
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



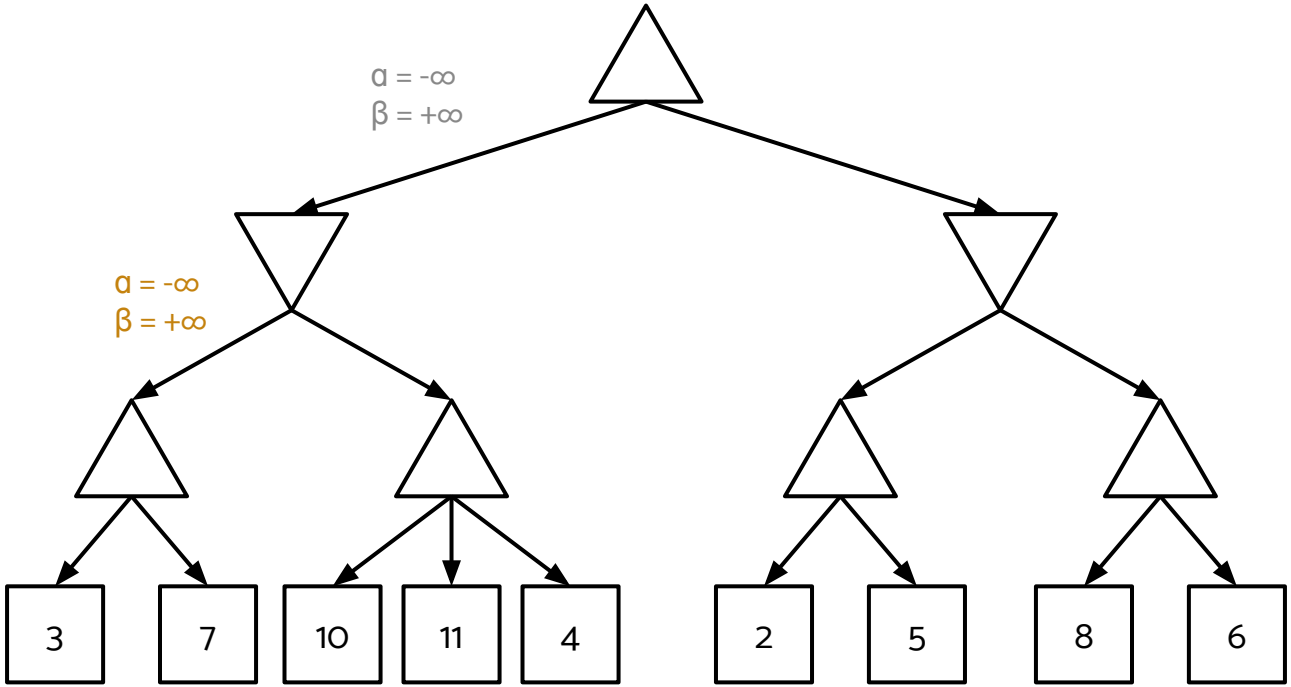
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



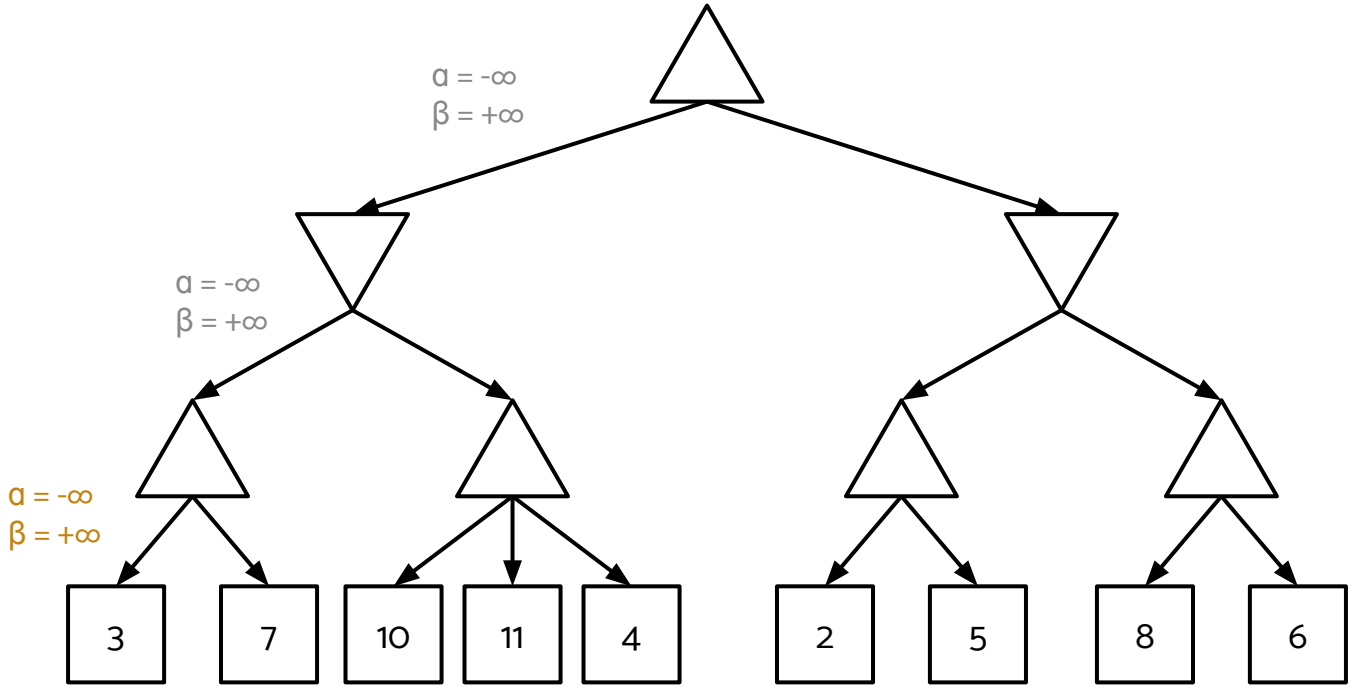
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



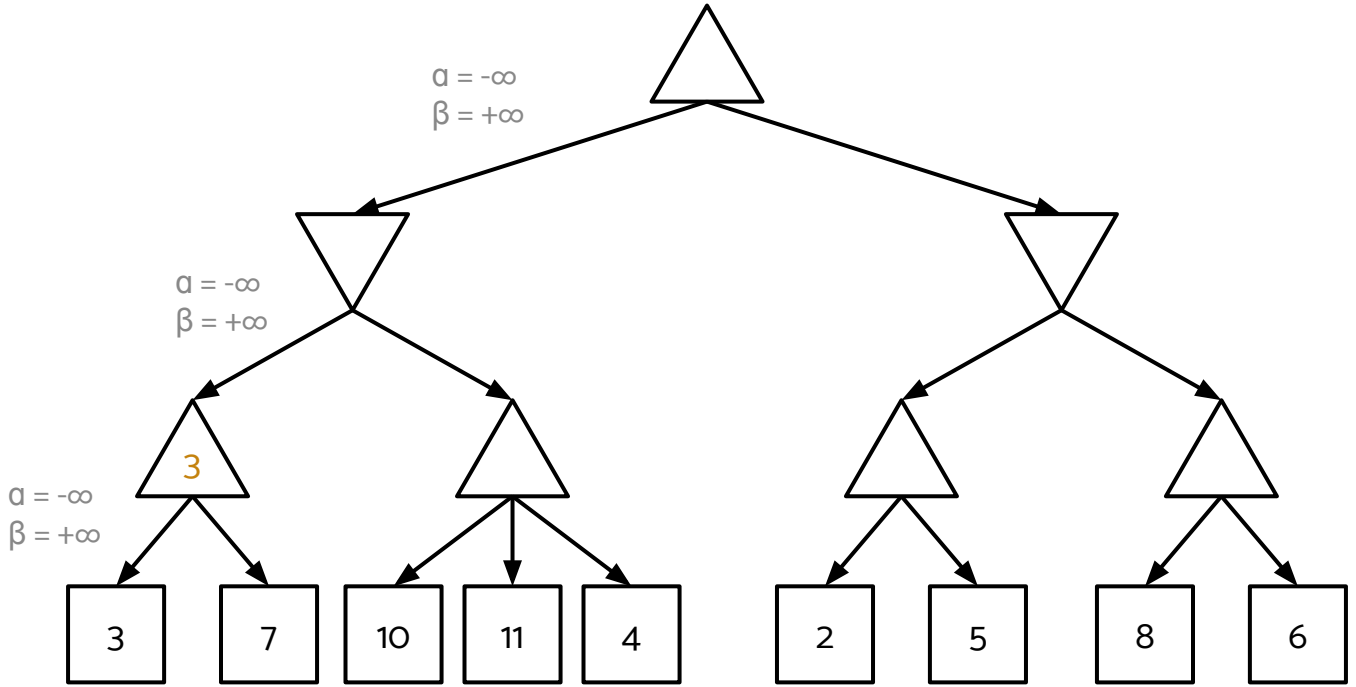
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



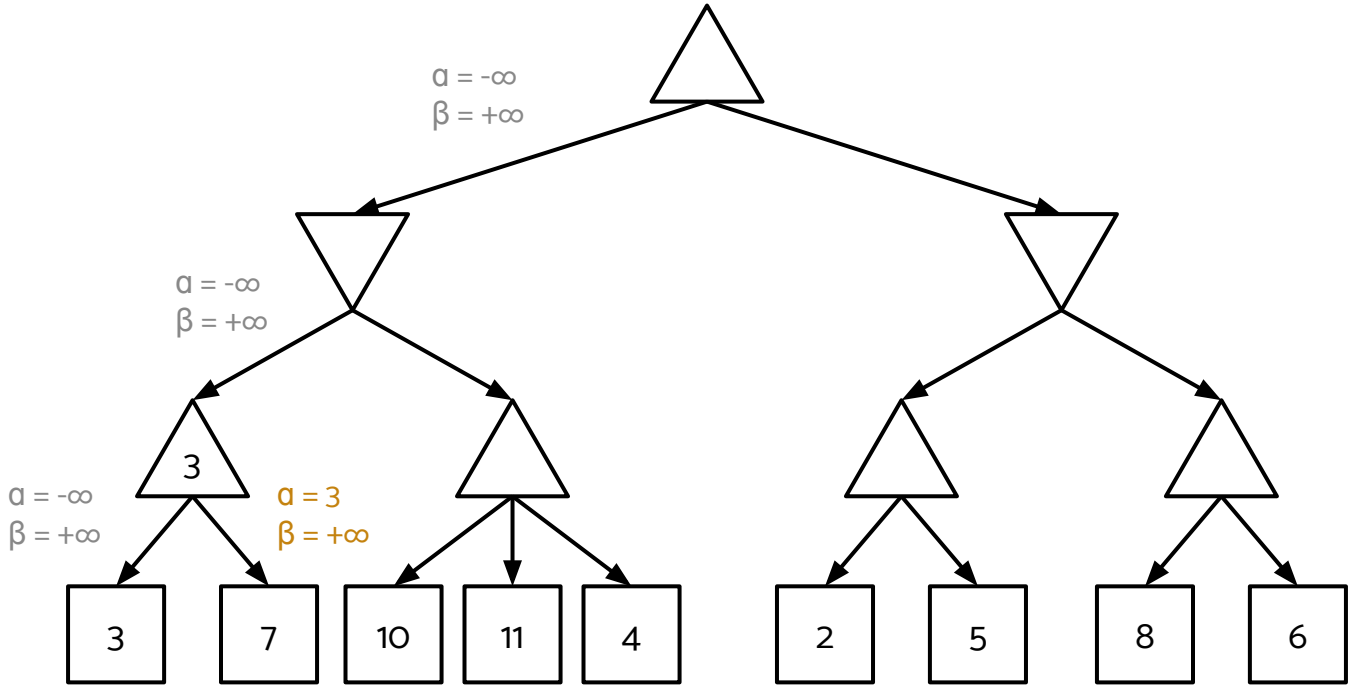
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



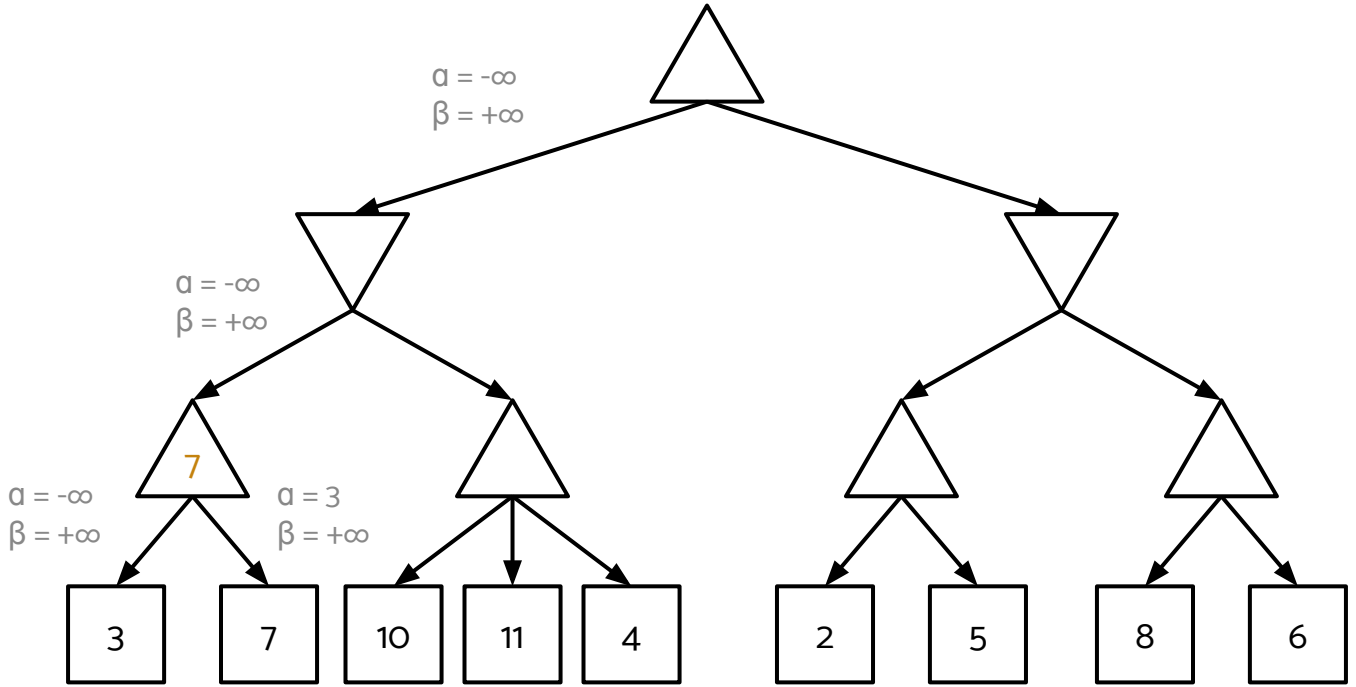
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



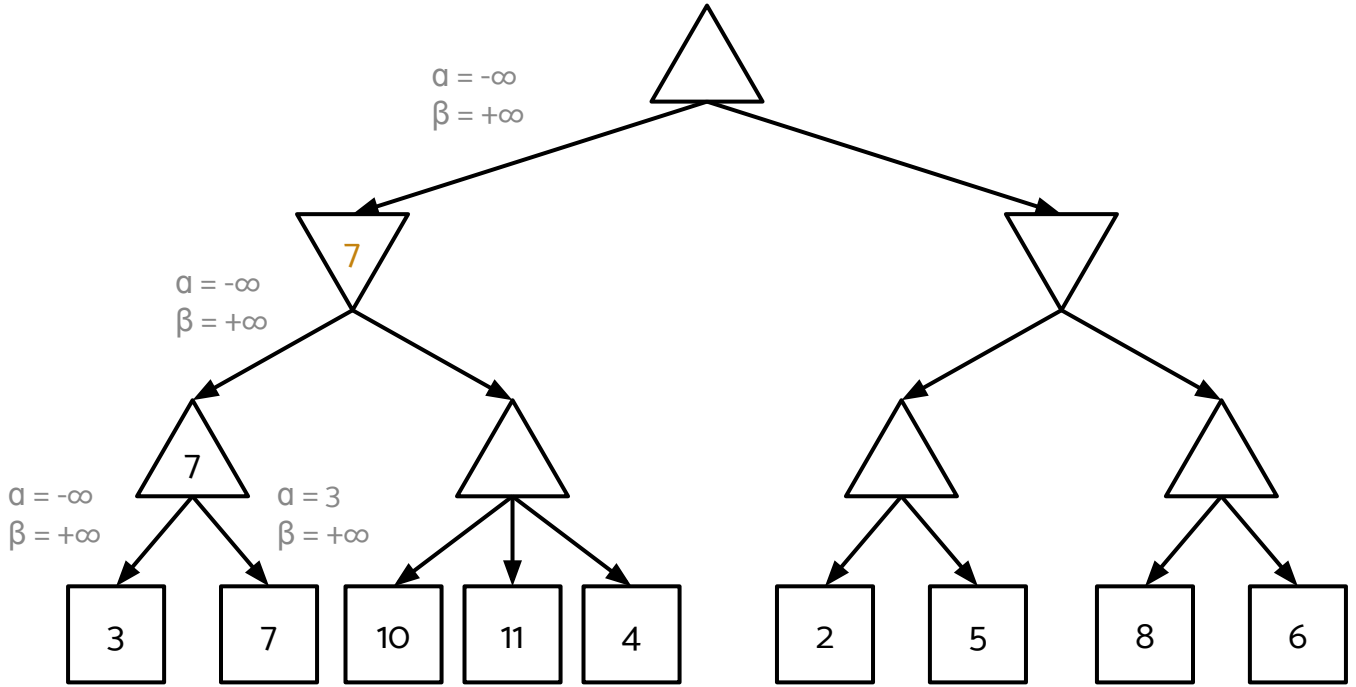
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



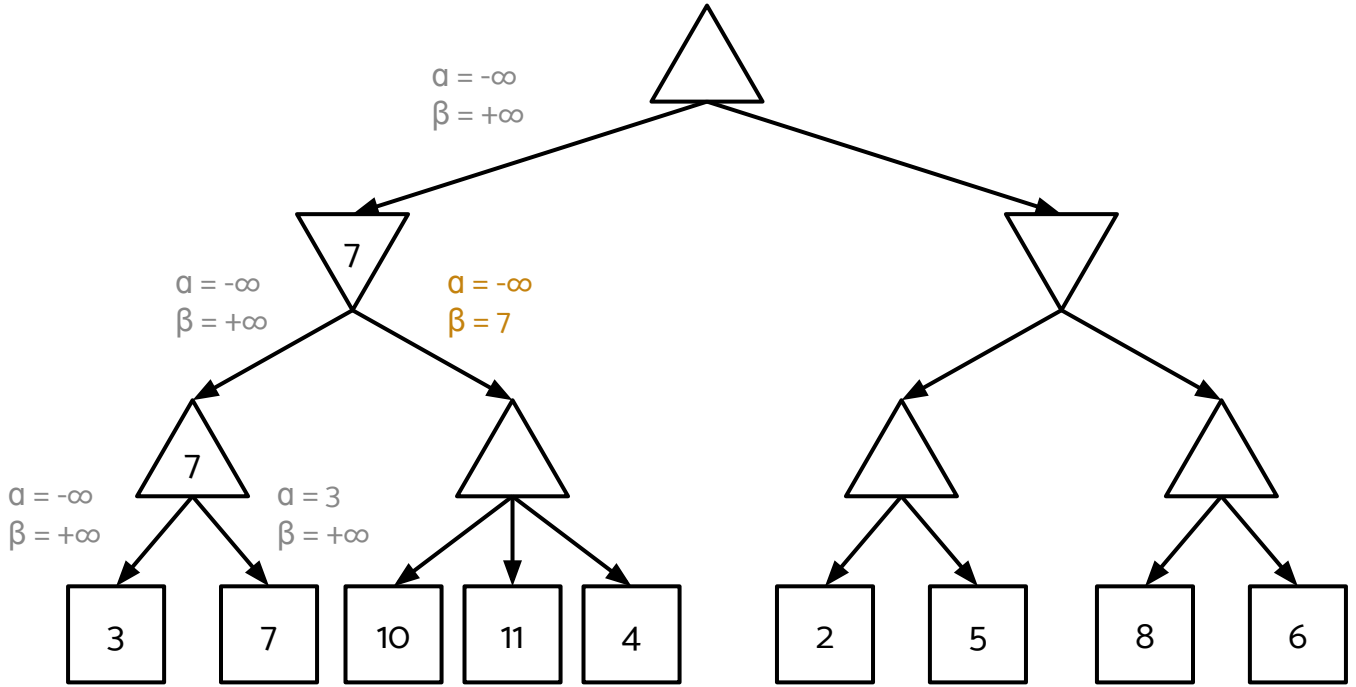
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



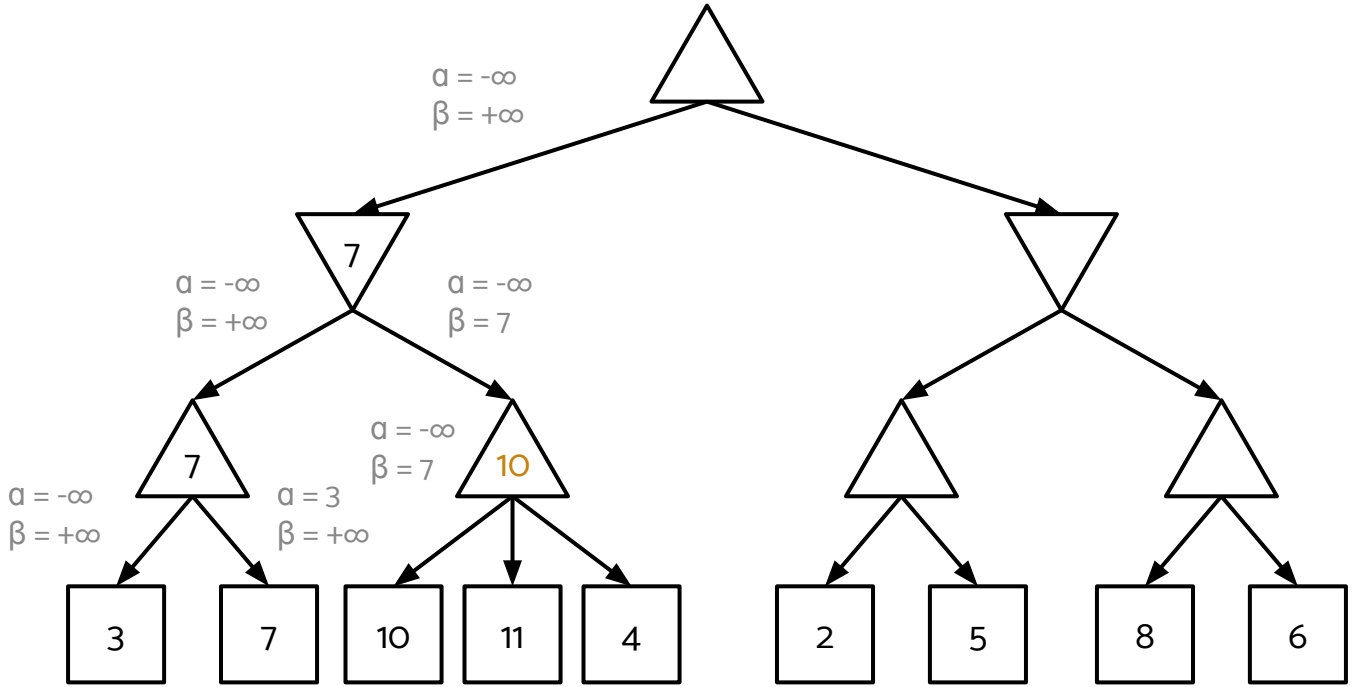
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



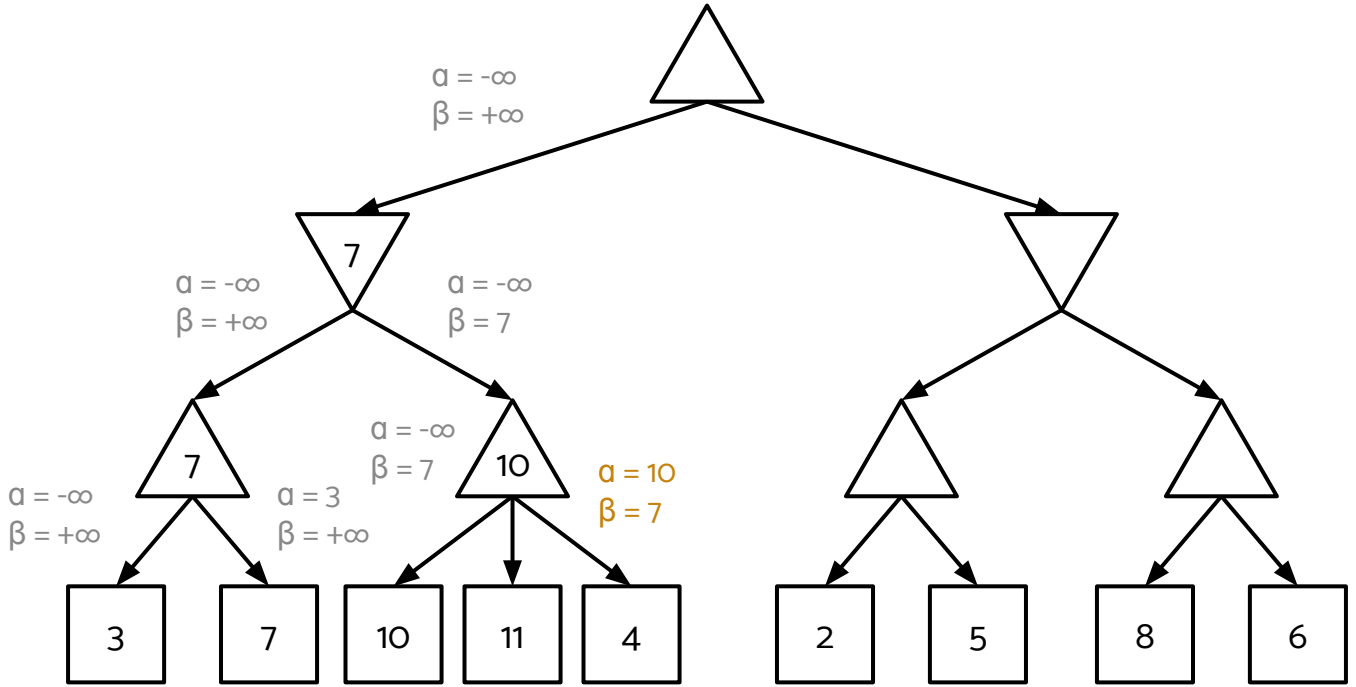
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



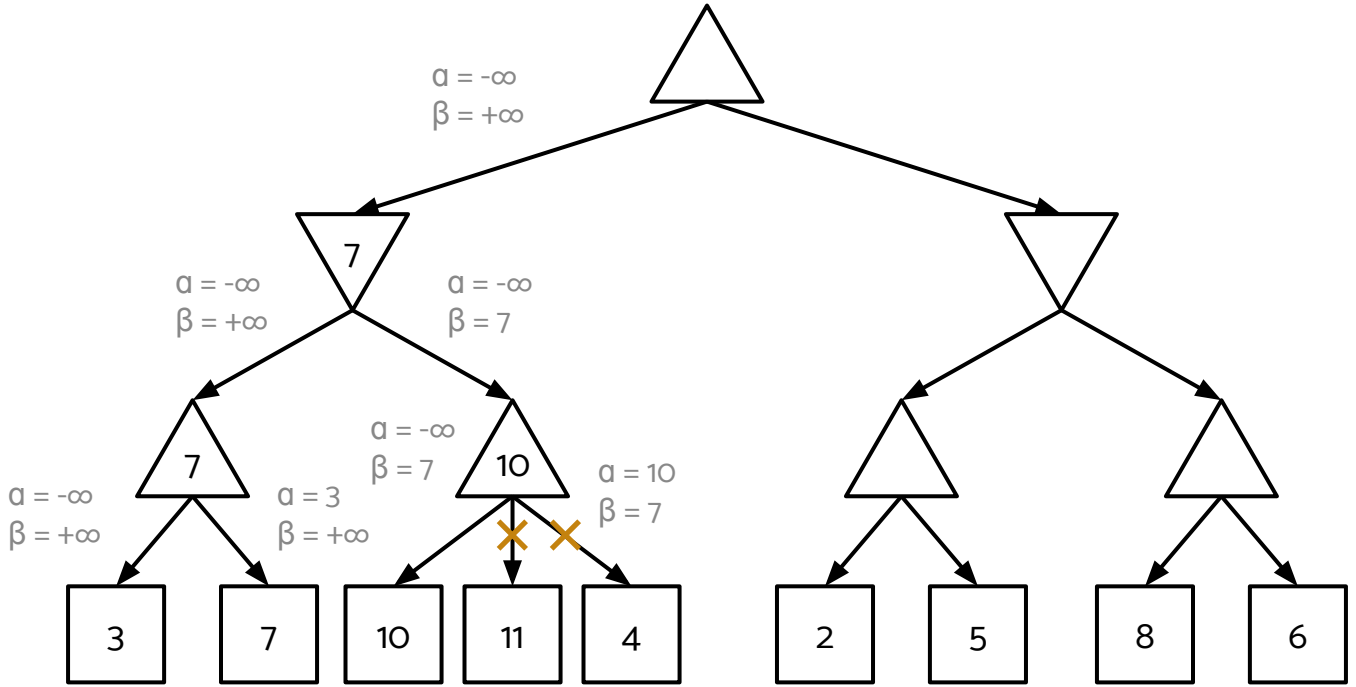
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



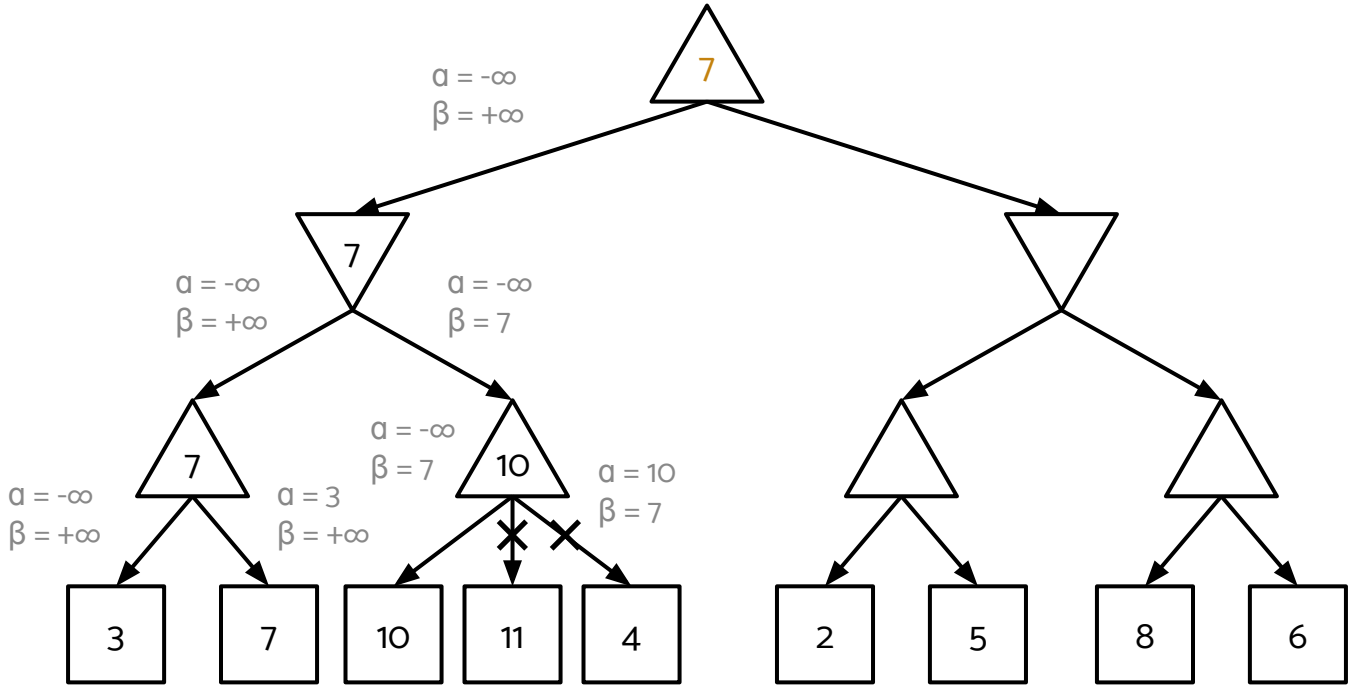
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



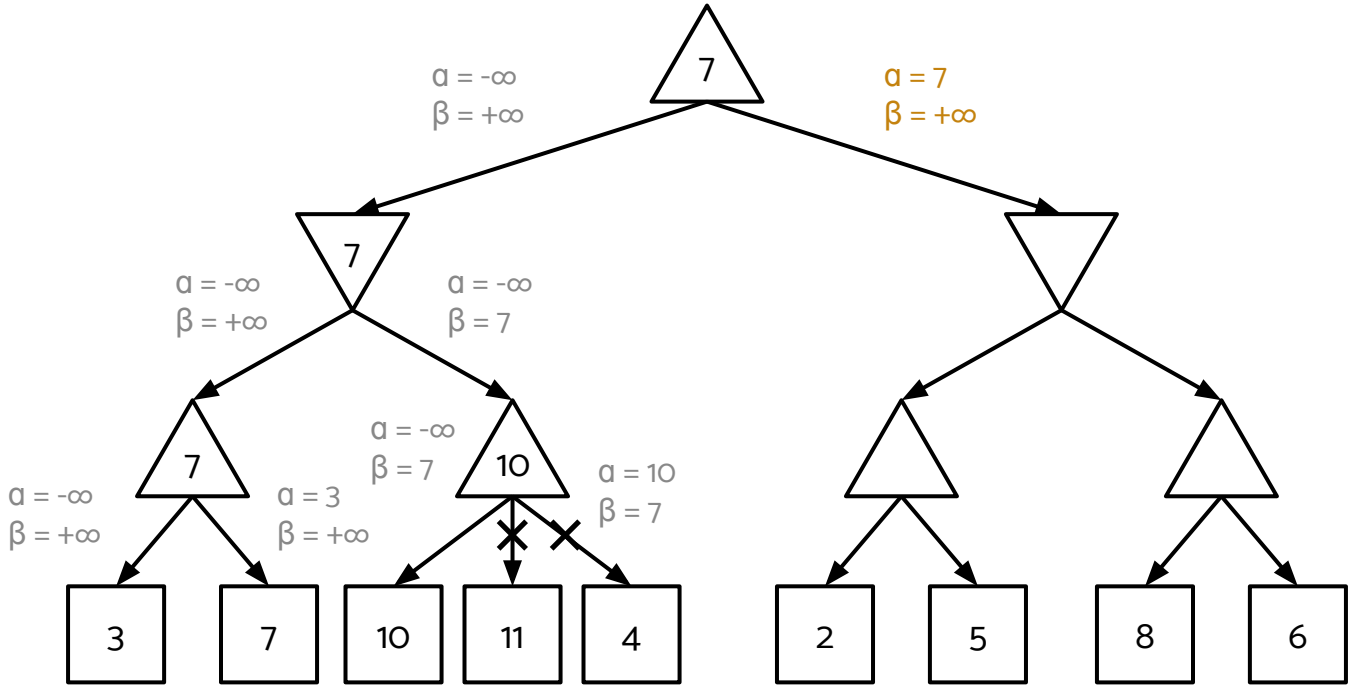
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



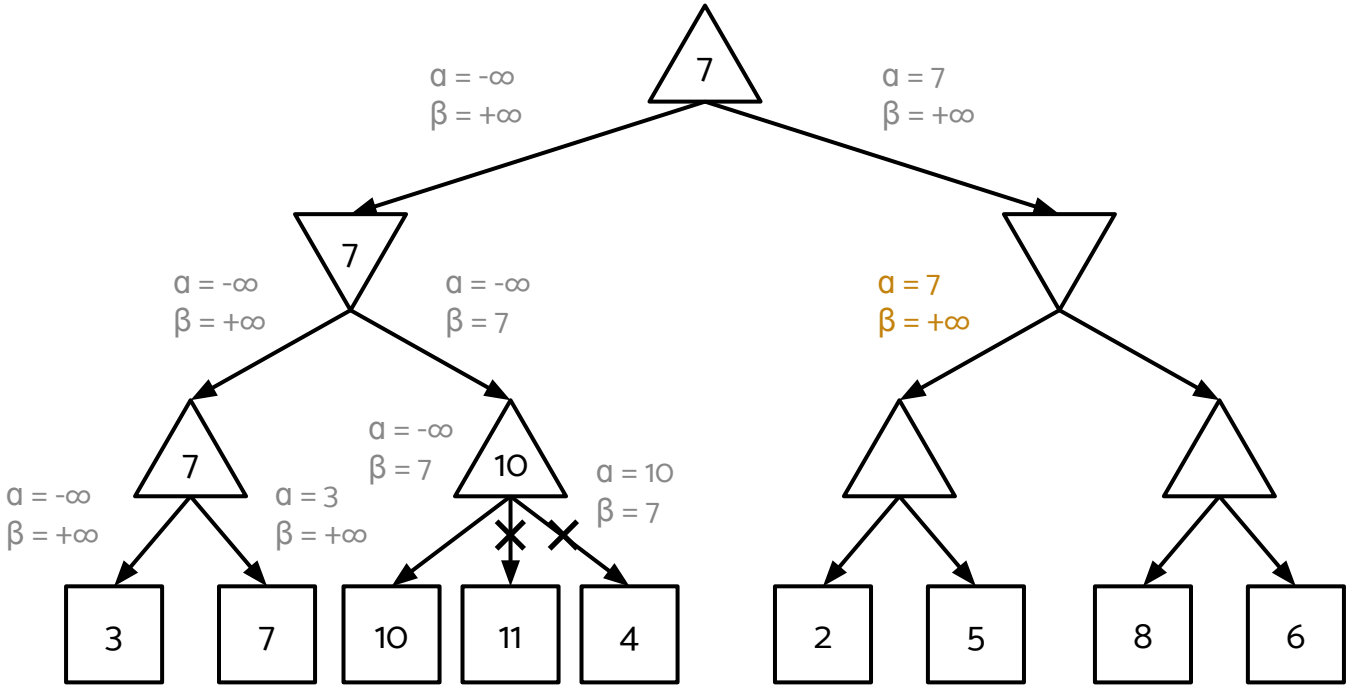
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



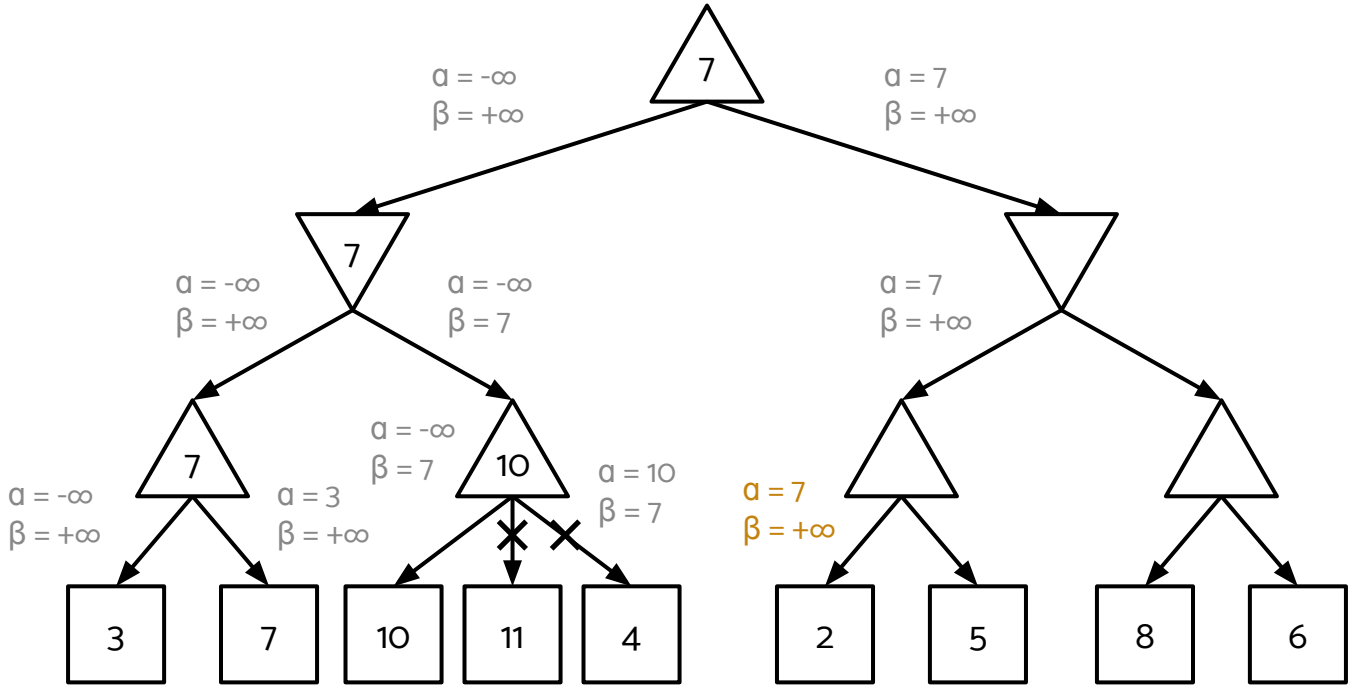
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



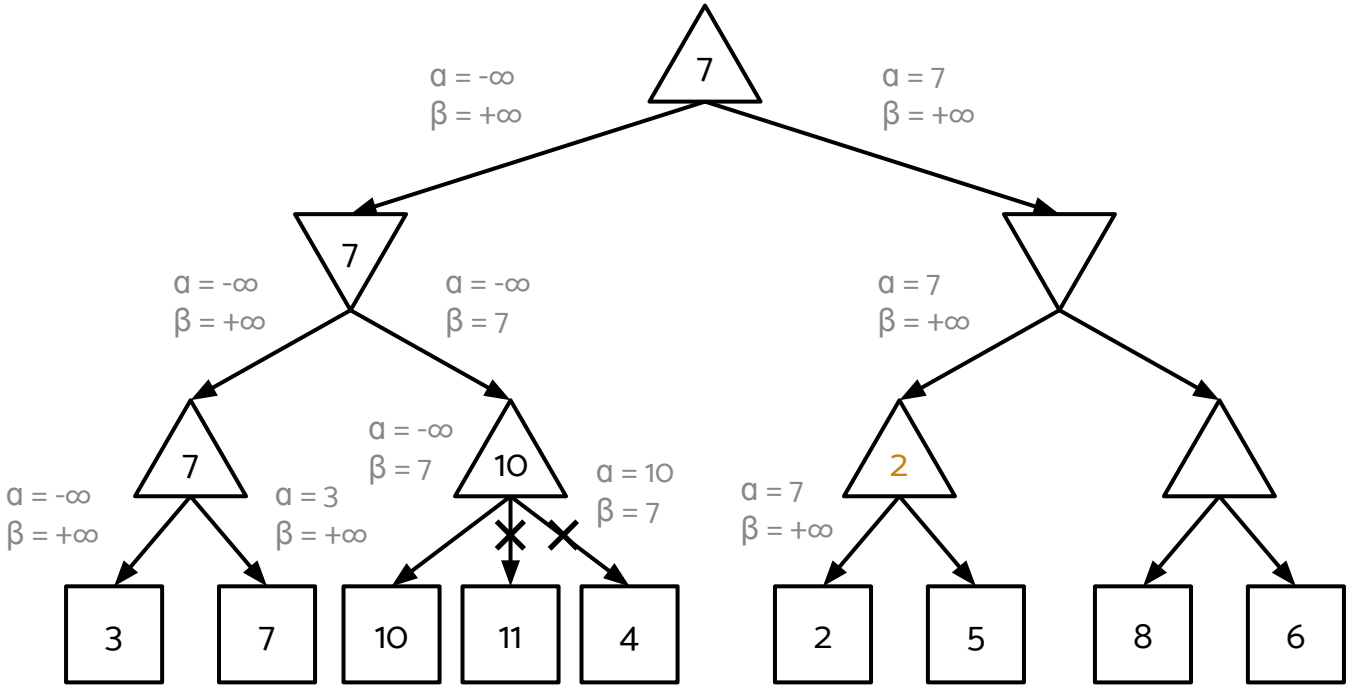
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



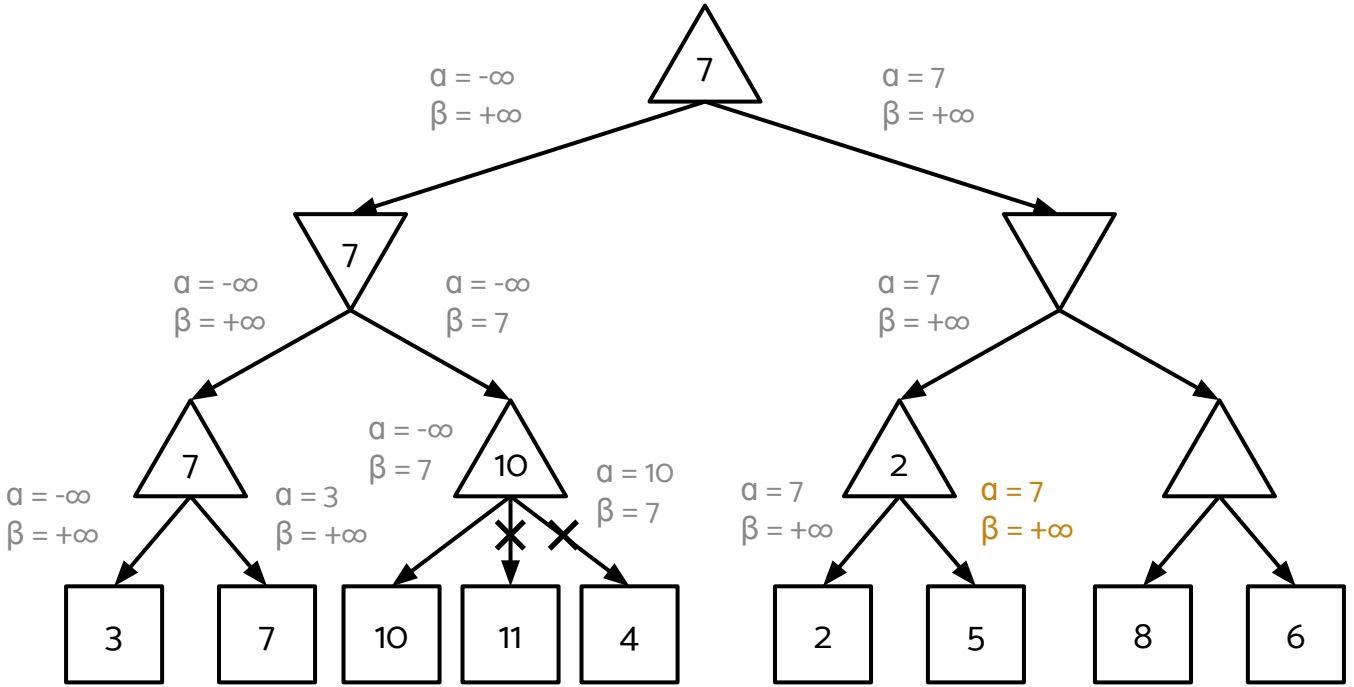
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



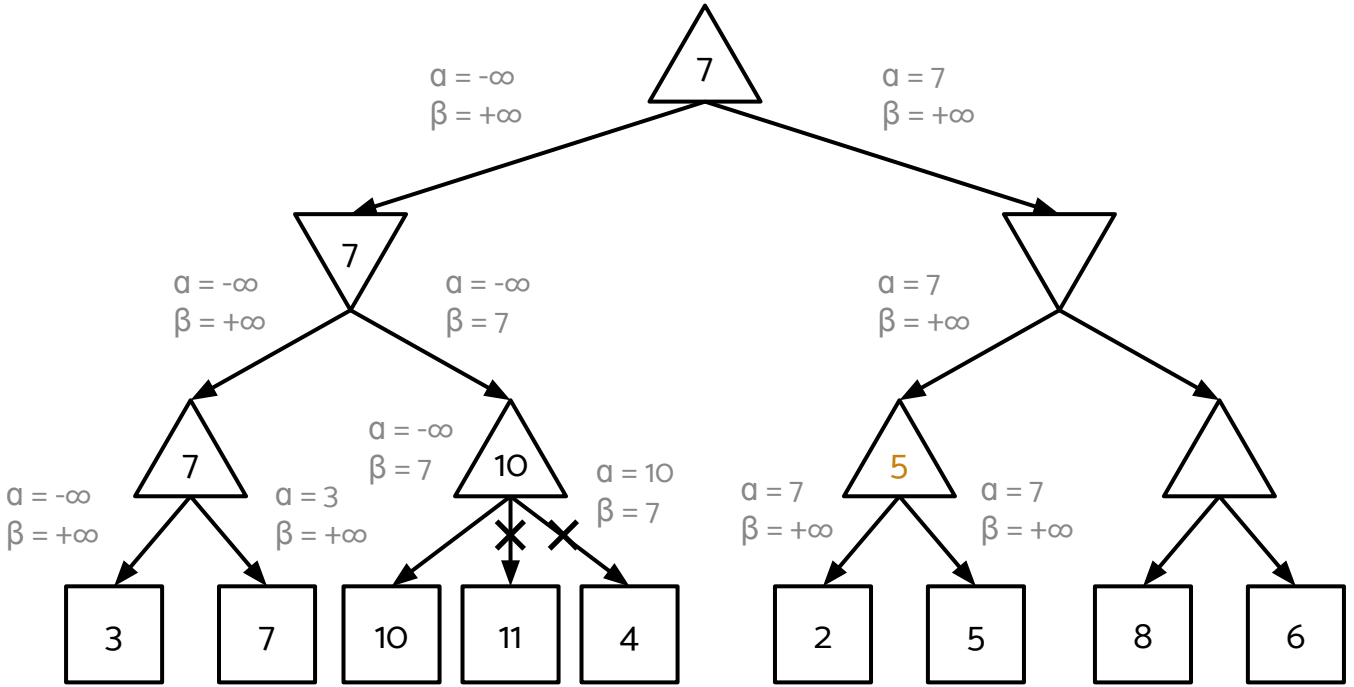
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



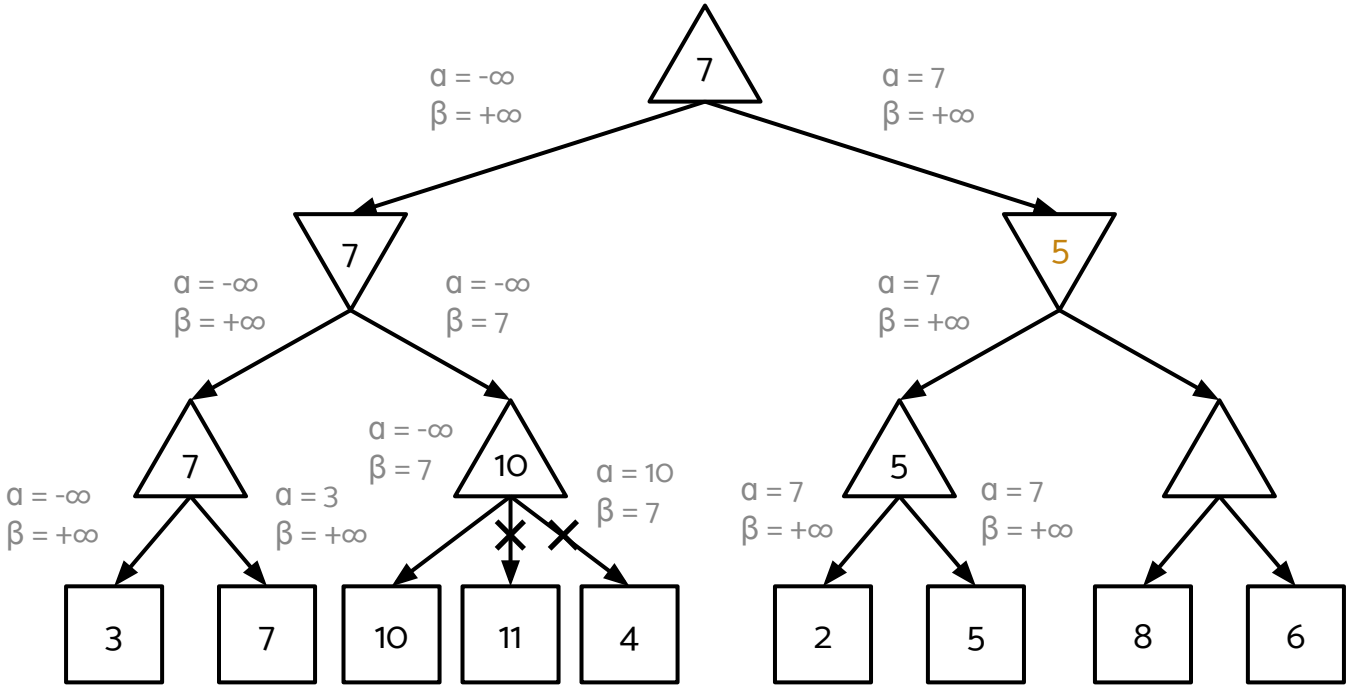
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



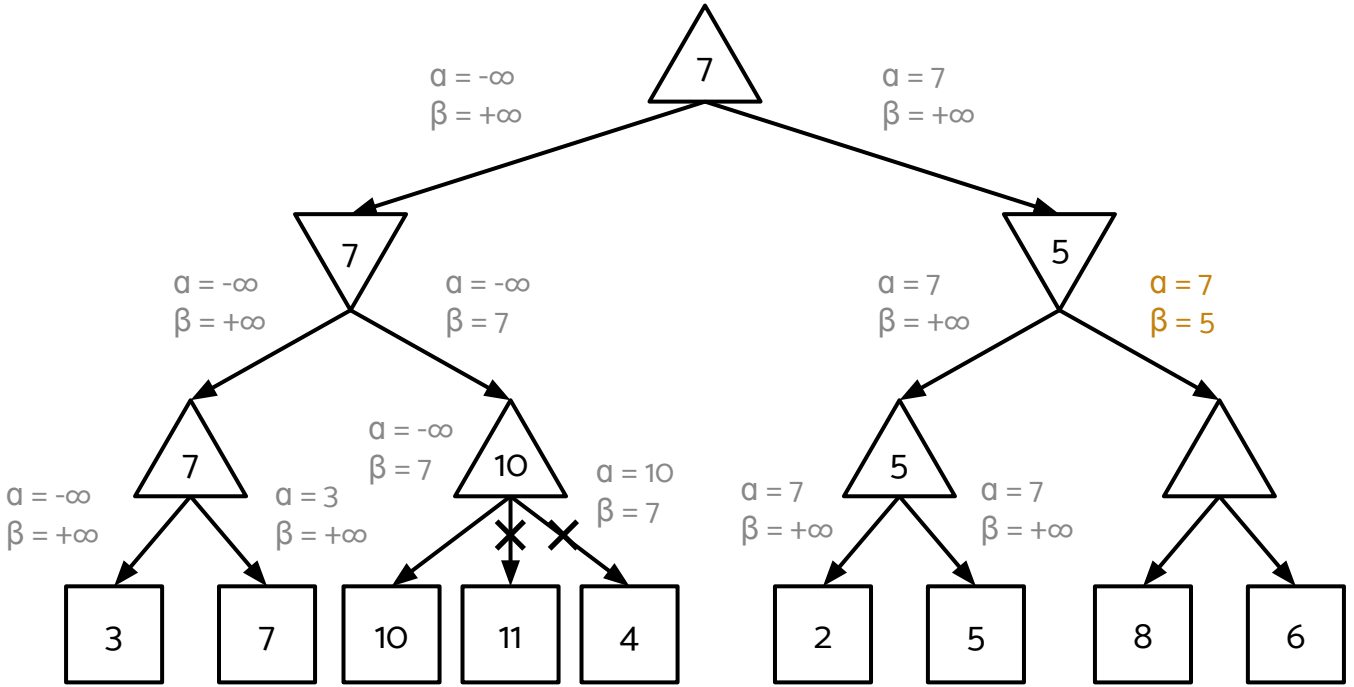
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



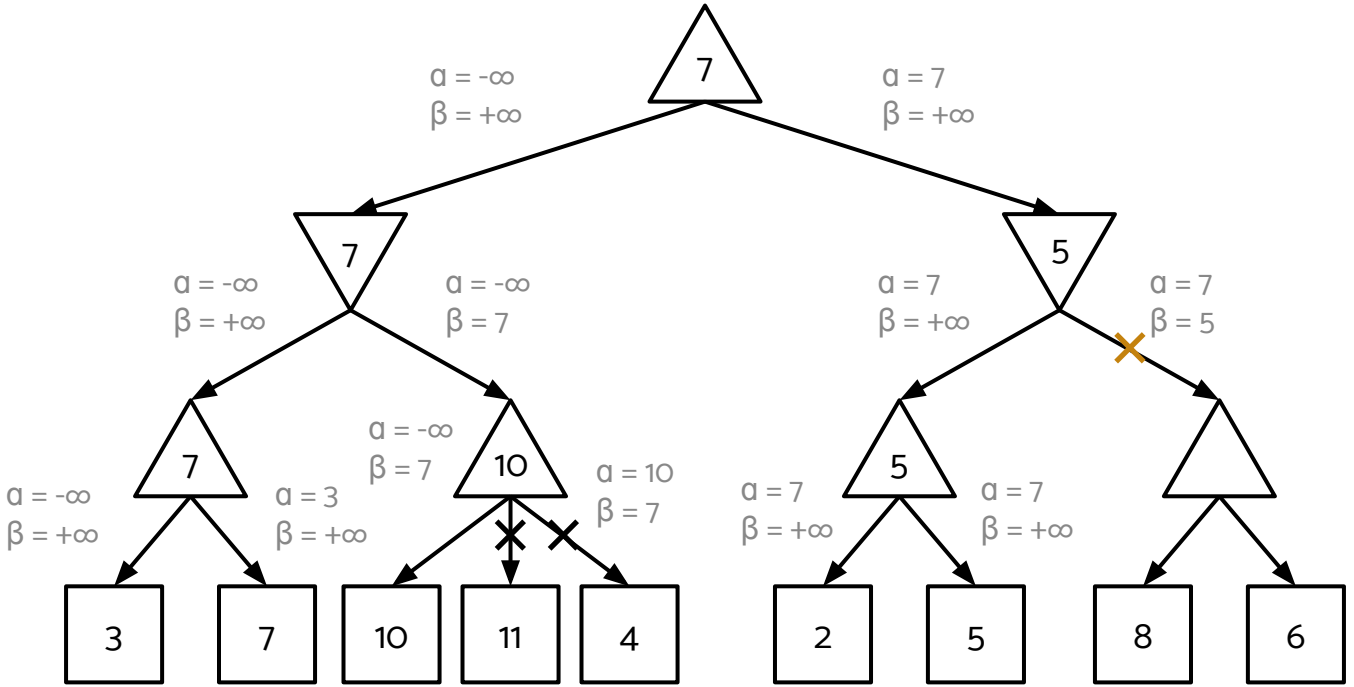
3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



3 Shall We Play a Game?

Fill in the following game tree and prune using alpha-beta pruning.



4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(________); }
}
public static void sumPaths(TreeNode T, int k, String path) {









}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); } // Path starts out empty
}
public static void sumPaths(TreeNode T, int k, String path) {









}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } // Once we hit a leaf with the right sum, we can print out the path that we’ve built




}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } else {
          path += T.val + “ “; // Otherwise we need to add to the path




     }
}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } else {
          path += T.val + “ “;
          if (T.left != null) { // Keep going down the left path if its not null
               sumPaths(T.left, k - T.val, path); // Subtract value from goal
          }



     }
}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } else {
          path += T.val + “ “;
          if (T.left != null) {
               sumPaths(T.left, k - T.val, path);
          }
          if (T.right != null) { // Same thing for right side
               sumPaths(T.right, k - T.val, path);
          }
     }
}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } else {
          path += T.val + “ “;
          if (T.left != null) {
               sumPaths(T.left, k - T.val, path);
          }
          if (T.right != null) {
               sumPaths(T.right, k - T.val, path);
          }
     } // If the path to the leaf doesn’t sum to k, we just return without doing anything
}
```

4 Sum Paths *Extra*

Write printSumPaths such that it prints out all root-to-leaf paths whose values sum to k.

```
public static void printSumPaths(TreeNode T, int k) {
     if (T != null) { sumPaths(T, k, “”); }
}
public static void sumPaths(TreeNode T, int k, String path) {
     if (T.left == null && T.right == null && k == T.val) {
          System.out.println(path + T.val);
     } else {
          path += T.val + “ “;
          if (T.left != null) {
               sumPaths(T.left, k - T.val, path);
          }
          if (T.right != null) {
               sumPaths(T.right, k - T.val, path);
          }
     }
}
```