Discussion 8: Binary Trees

Discussion: Wed 4-5pm 120 Wheeler Hall
Lab: Thurs 3-5pm 275 Soda Hall
OH: Thurs 1-2pm in 220 Jacobs Hall

email: asetoodehnia@berkeley.edu
website: asetoodehnia.github.io (or find it through the CS 61B staff webpage)

Reminders:

- Project 2 (Tablut) spec is out!
- HW5 due **today!**

Today's Goals:

1. Review ***Trees***
2. Get through as many questions as possible

Trees

- **root** of a tree is a **non-empty node** with **no parent in that tree**
- The **order**, **arity**, or **degree** of a node (tree) is its number (maximum number) of children.
- A **leaf node** has no children
- The **height** of a node in a tree is the largest distance to a leaf
- The **depth** of a node in a tree is the distance to the root of that tree

Tree Traversals

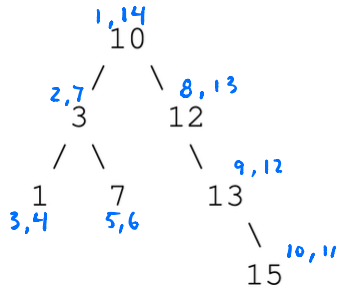
- Depth First Search (DFS)
  - **pre-order:** visit node, traverse its children
  - **post-order:** traverse children, visit node
  - **in-order:** traverse first child, visit node, traverse second child (binary trees only)
- Breadth First Search (BFS)

Binary Search Trees

- Tree nodes contain **keys**, and possibly other data
- All nodes in left subtree of node have **smaller** keys
- All nodes in right subtree of node have **larger** keys

1. Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals of the following binary search tree. For all traversals, process child nodes left to right.



DFS pre-order: take the pre-order numbers.

10, 3, 1, 7, 12, 13, 15

DFS post-order: take the post-order numbers.

1, 7, 3, 15, 13, 12, 10

DFS in-order:

1, 3, 7, 10, 12, 13, 15

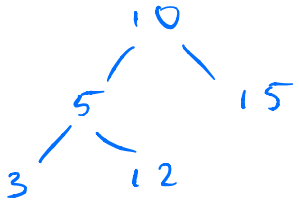
BFS:

10, 3, 12, 1, 7, 13, 15

2. Is This a BST?

(a) The following code should check if a given binary tree is a BST. However, for some trees, it is returning the wrong answer. Give an example of a binary tree for which the method fails.

```
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);
    }
}
```



↳ only checks the values of a node's children, not the entire left/right subtrees.

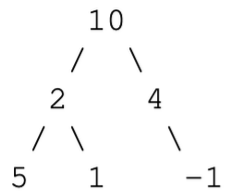
(b) Now, write `isBST` that fixes the error encountered in part (a).
Hint: You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

```
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;
    }
    return isBSTHelper(T.left, min, T.val) &&
           isBSTHelper(T.right, T.val, max);
}
```

3. Sum Paths

Define a root-to-leaf path as a sequence of nodes from the root of a tree to one of its leaves. Write a method `printSumPaths(TreeNode T, int k)` that prints out all root-to-leaf paths whose values sum to k. For example, if T is the binary tree in the diagram below and k is 13, then the program will print out `10 2 1` on one line and `10 4 -1` on another.



(a) Provide your solution by filling in the code below:

```
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);
        }
    }
}
```

(b) What is the worst case running time of the `printSumPaths` in terms of N, the number of nodes in the tree? What is the worst case running time in terms of h, the height of the tree?

string concat is linear $\Rightarrow 1+2+3+\cdots+N = \Theta(N^2)$

2^h leaves in worst case
string concat is $\Theta(h)$ $\Rightarrow \Theta(h2^h)$