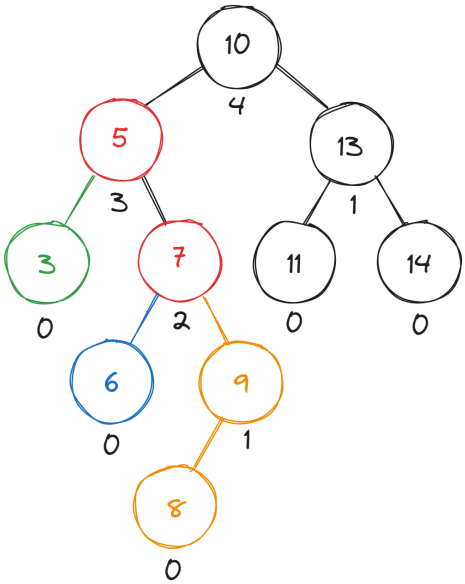
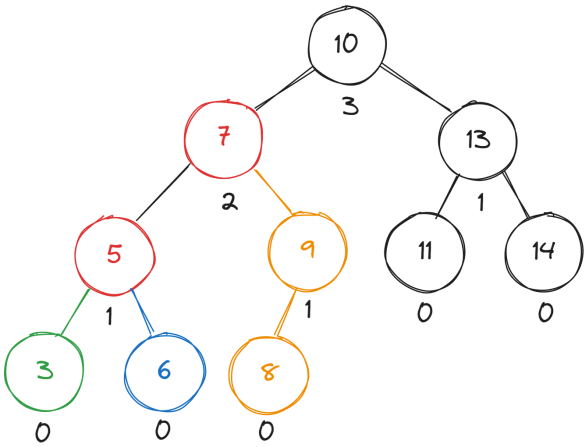
EL-GY 9343 Homework 7

Yihao Wang
yw7486@nyu.edu

1. True or False:
   (a) BST operations (search, predecessor, successor, minimum, insert, delete) are $O(h)$, where h is height of the tree. True
   (b) The height of a binary search tree with n elements is $\Theta(\log(n))$. False
   (c) In-order tree walk of a binary search tree outputs an unsorted sequence. False
   (d) When deleting node z with 2 children, we cannot replace z by its predecessor. False
   (e) In an AVL tree, the right subtree and the left subtree of any node have the same height. False
   (f) Worst-case tree height for AVL tree with n nodes is $\Theta(\log(n))$. True

2. Build an AVL Tree out of the BST according to the rotation operations in the lecture with one rotation operation. The pre-order traversal sequence of the BST is {10, 5, 3, 7, 6, 9, 8, 13, 11, 14}. First **draw the BST**, then do the rotation. Also answer the type of this rotation (**Single** or **Double**).
This is a single rotation.

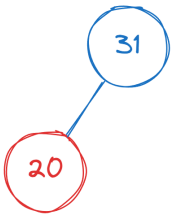




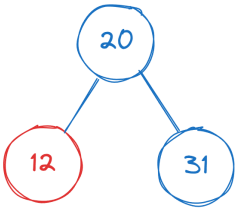
3. Build an AVL tree step by step. The keys are inserted in the order of:{31, 20, 12, 26, 28, 27}. Please draw the tree after each key is inserted, and mark the type of rotation taken, if any, at each step.

insert key 31;

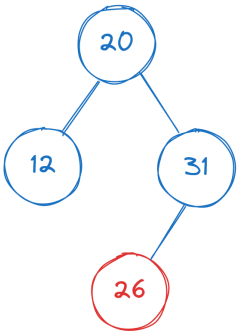
insert key 20;



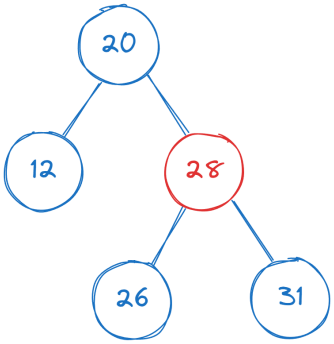
insert key 12;
(single rotation)



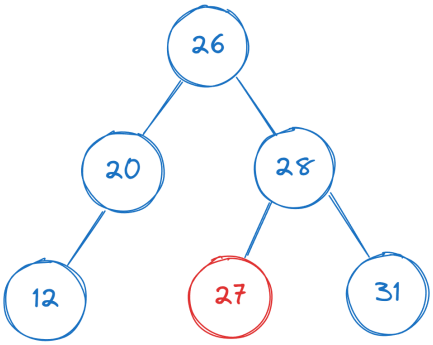
insert key 26;



insert key 28;
(double rotation)



insert key 27;
(double rotation)



4. (**Radix trees**) Given two strings, $a = a_0a_1a_2 \ldots a_p$ and $b = b_0b_1b_2 \ldots b_q$, where each a_i and each b_j belongs to some ordered set of characters, we say that string a is *lexicographically* less than string b if either

i there exists an integer j, where $0 \leq j \leq \min\{p, q\}$, such that $a_i = b_i$ for all $i = 0, 1, \ldots, j-1$ and $a_j < b_j$, or

ii $p < q$ and $a_i = b_i$ for all $i = 0, 1, 2, ...p$.

For example, if a and b are bit strings, we can verify that $10100 < 10110$ and $10100 < 101000$ following the above rules. This ordering is similar to that used in English-language dictionaries. The *radix tree* data structure shown in Figure 12.5 (a.k.a. *trie*) stores the bit strings 1011, 10, 011, 100 and 0. When searching for a key $a = a_0a_1a_2 \ldots a_p$, go left at a node of depth i if $a_i = 0$ and right if $a_i = 1$.

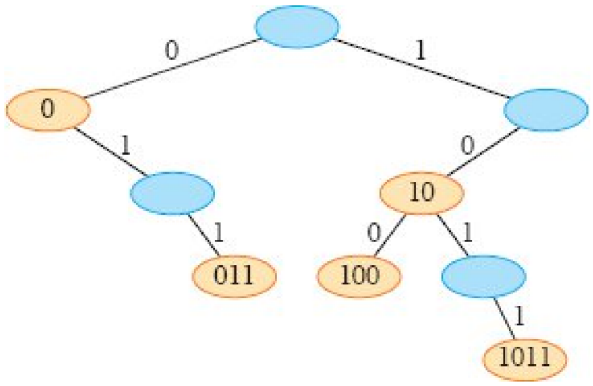


Figure 12.5 A radix tree storing the bit strings 1011, 10, 011, 100, and 0. To determine each node's key, traverse the simple path from the root to that node. There is no need, therefore, to store the keys in the nodes. The keys appear here for illustrative purposes only. Keys corresponding to blue nodes are not in the tree. Such nodes are present only to establish a path to other nodes.

(a) Let S be a set of distinct bit strings whose lengths **sum to** n. Show how to use a radix tree to sort S lexicographically in $\Theta(n)$ time.
Let lengths of bit strings in S be $\{l_1, l_2, \ldots, l_m\}$, such that $\sum_{i=1}^{m} l_i = n$. While inserting the i^{th} bit string into the radix tree, it costs $\Theta(l_i)$ because we need to trace from the root to the corresponding node according to every digit. Therefore, in total it costs $\sum_{i=1}^{m} \Theta(l_i) = \Theta(n)$ to build the complete radix tree. Then we can readily use in-order traversal to print out the sorted S.

(b) Now consider that the ordered set of characters have a size of $k > 2$ (for lower-case alphabetic letters, $k = 26$). Let S be a set of distinct strings whose lengths sum to n. Will lexicographically sorting still cost $\Theta(n)$ time? Why, or why not?
Yes, it will still cost $\Theta(n)$ because the only difference between binary bits and alphabetic letters is the number, which only influences the maximum children number of each node.