HW07 Solutions

Data Structure and Algorithm (New York University)

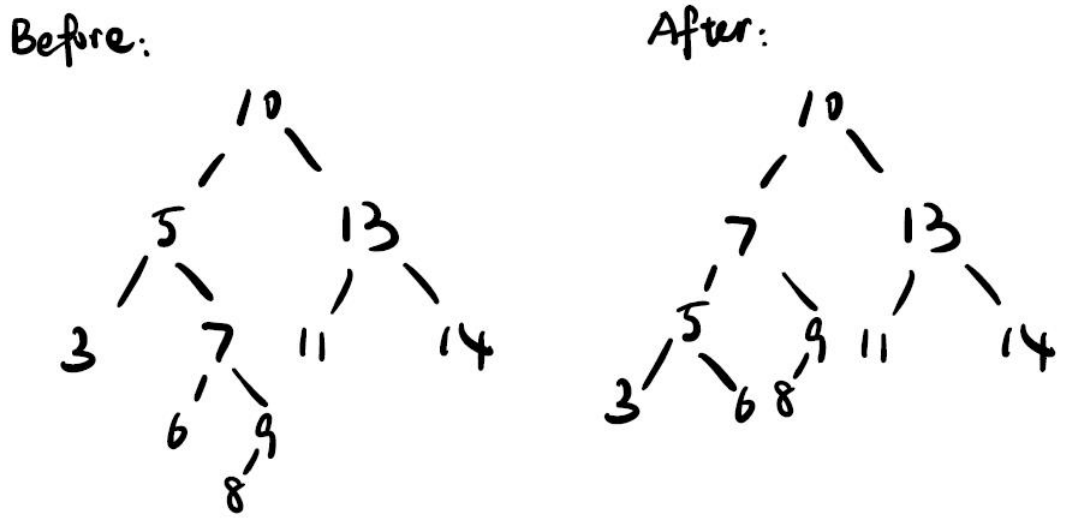
EL9343 Homework 7

Due: Oct. 25th 8:00 p.m.

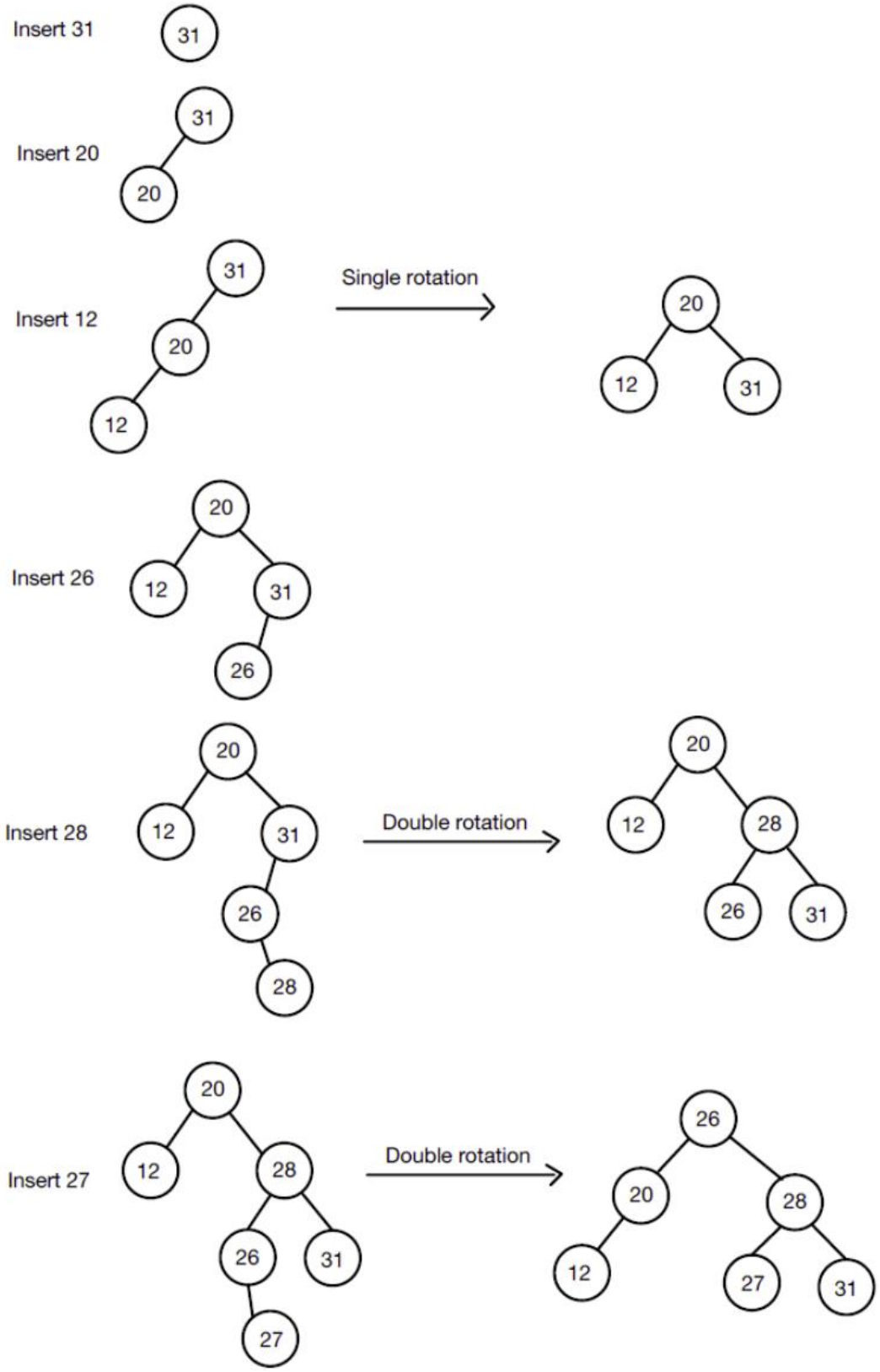
1. True or False:
   (a) BST operations (search, predecessor, successor, minimum, insert, delete) are $O(h)$, where h is height of the tree.
   (b) The height of a binary search tree with n elements is $\Theta(\log(n))$.
   (c) In-order tree walk of a binary search tree outputs an unsorted sequence.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

   Solution:
   (a) True
   (b) False
   (c) False
   (d) False
   (e) False
   (f) 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**).
   Solution: The BST is shown on the left, and the AVL Tree is on the right. It 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.
   Solution:



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 $i = 0, 1, 2, \ldots p$.

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 $a = a_0 a_1 a_2 \ldots a_p$, go left at a node of depth i if $a_i = 0$ and right if $a_i = 1$.

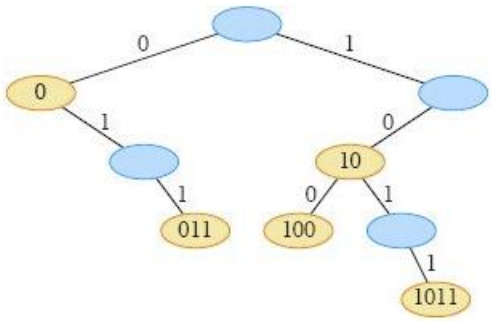


Figure 12.5 A radix tree storing the bit strings 1011, 10, 011, 100, and 0. To determine each node's key, traverse the simple path from the root to that node. There is no need, therefore, to store the keys in the nodes. The keys appear here for illustrative purposes only. Keys corresponding to blue nodes 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.

(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?

Solution:

(a) To insert a bit string of length l into a radix tree, we start at the root, visit exactly l nodes, and sometimes create $O(l)$ nodes. Thus, this inserting cost $\Theta(l)$. Since bit strings in S have lengths than sum to n, if we build the radix tree by inserting the bit strings one by one, the total cost will be $\Theta(n)$, and the number of nodes in the tree will be $O(n)$.
Then, the pre-order traversal of this radix tree is just the sorted output. Note that there are only $O(n)$ nodes in the tree, so pre-order traversal will cost $O(n)$ as well. Therefore, totally the cost is $\Theta(n)$.

(b) Yes, the cost is still $\Theta(n)$. Increasing the size k only brings an increase in the constant part, i.e. the maximum number of operation inside a node. Inserting a string of length l still require visiting l nodes, thus the cost in big-theta notation is the same.