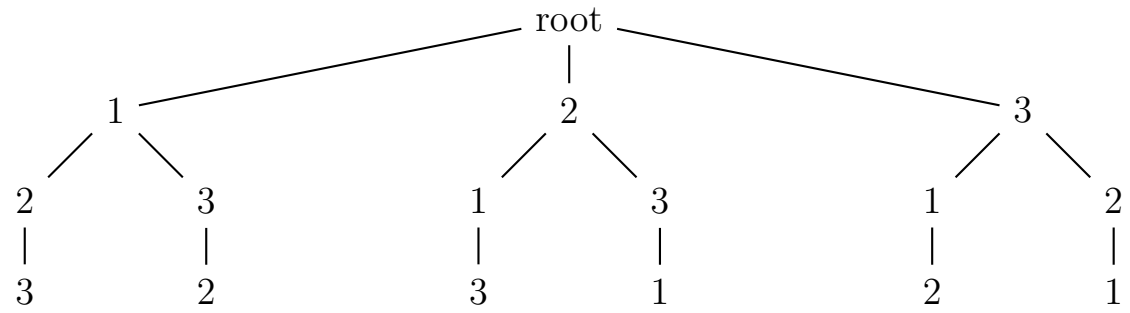
Lab07-Trees

VE281 - Data Structures and Algorithms, Xiaofeng Gao, TA: Qingmin Liu, Autumn 2019

∗ Please upload your assignment to website. Contact webmaster for any questions.
∗ Name:Wu Jiayao Student ID:517370910257 Email: jiayaowu1999@sjtu.edu.cn

Hint: You can use the package **tikz** to draw trees.



1. Red-black Tree

   (a) Suppose that we insert a sequence of keys 9, 3, 1 into an initially empty red-black tree. Draw the resulting red-black tree.

   (b) Suppose that we further insert key 6 into the red-black tree you get in Problem (1-a). Draw the resulting red-black tree.

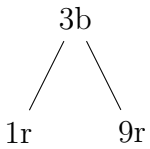
   (c) Suppose that we further insert keys 2, 8 into the red-black tree you get in Problem (1-b). Draw the resulting red-black tree.

   (d) Suppose that we further insert key 7 into the red-black tree you get in Problem (1-c). Draw the resulting red-black tree.

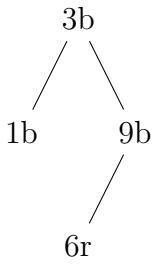
   (e) Suppose that we further insert keys 4, 5 into the red-black tree you get in Problem (1-d). Draw the resulting red-black tree.

   When you draw the red-black tree, please indicate the color of each node in the tree. For example, you can color each node or put a letter **b/r** near each node.

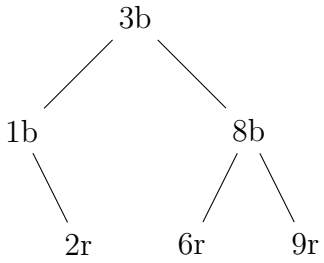
   Solution. (a)



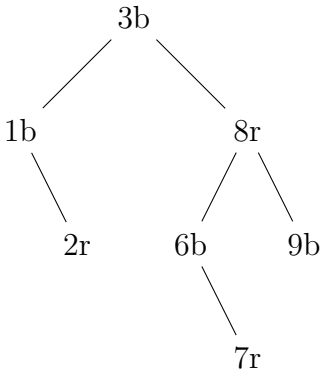
(b)



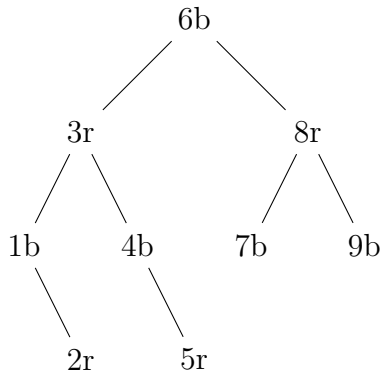
(c)



(d)



(e)



□

2. Show the alphabet trie for the following collection of words: {chicken, goose, deer, horse, antelope, anteater, goldfish, ant, goat, duck}.

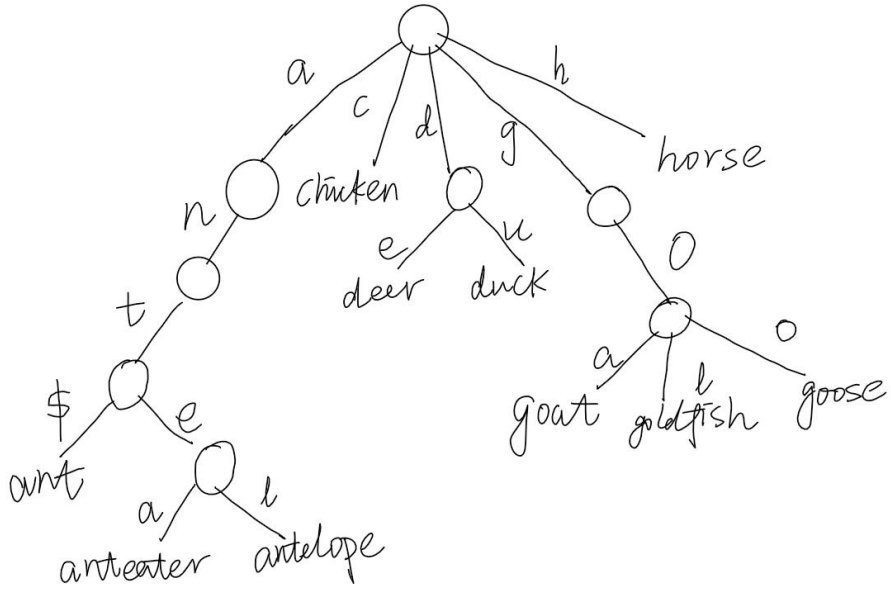


Figure 1: Handmade tries

Solution. □

3. Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations.

   Hint: First show that at most $n-1$ right rotations suffice to transform the tree into a right-skewed binary search tree.

   Solution. For every node that is at the left of the root, if it has a right child, rotate left (the old left child becomes the root, the old root becomes its left child). For every node that is at the right of the root, if it has a left child, rotate right (the old right child becomes the root, the old root becomes its right child). Such operations is $\mathcal{O}(n)$, since it visits every node once. The result tree (just call it a tree) is an increasing, going from left to right chain, with one node marked as root.

   Such operations above can be reversed to form another binary search tree from an increasing, going from left to right chain. Hence, we do a set of above operations to turn a BST into a chain, then change the root of the chain, do the reverse operations, then we get a new BST tree. It takes $\mathcal{O}(n)$ operations. □

4. Suppose that an AVL tree insertion breaks the AVL balance condition. Suppose node P is the first node that has a balance condition violation in the insertion access path from the leaf. Assume the key is inserted into the left subtree of P and the left child of P is node A. Prove the following claims:

   (a) Before insertion, the balance factor of node P is 1. After insertion and before applying rotation to fix the violation, the balance factor of node P is 2.

   (b) Before insertion, the balance factor of node A is 0. After insertion and before applying rotation to fix the violation, the balance factor of node A cannot be 0.

   Solution. (a) The problem can be transformed into solving an equation, since insertion is done on the left subtree. By insertion $h_l \leq h_l^{'} \leq h_l + 1$.

$$\begin{cases} |B_P| = |h_l - h_r| \leq 1 \\ |B_P^{'}| = |h_l^{'} - h_r| > 1 \end{cases} \tag{1}$$

   If $h_l^{'} = h_l$, there is no solution, thus no balance is broken. Hence, $h_l^{'} = h_l + 1$, that $h_l - h_r$=1, which means that the factor of node P is 1. After insertion and before fixing, the factor of node P is 2.

   (b) The problem can be transformed into solving an equation, since insertion is done on the left subtree. By insertion $h_l \leq h_l^{'} \leq h_l + 1$.

$$\begin{cases} |B_P| = |h_l - h_r| \leq 1 \\ |B_P^{'}| = |h_l^{'} - h_r| > 1 \end{cases} \tag{2}$$

   If $h_l^{'} = h_l$, there is no solution, thus no balance is broken. Hence, $h_l^{'} = h_l + 1$, $h_l - h_r$=1. Denote the left child of A is M, the right is N. What about h_M and h_N?

   - $h_M = h_N + 1$. If the insertion happens in subtree N, no balance is broken. If happens in subtree M, the first node will be A rather than P, since B_A will be 2.
   - $h_N = h_M + 1$. If the insertion happens in subtree M, no balance is broken. If happens in subtree N, the first node will be A rather than P, since B_A will be -2.
   - $h_N = h_M$. It is correct. When inserted at rather M or N, $h_l^{'} = h_l + 1$, $|B_A| > 0$. Since $h_l^{'} - h_r$=2, balance is broken first at P.

   Hence ,$h_N = h_M$, the balance factor of A is 0. After insertion, the balance factor of node A cannot be 0.

   □