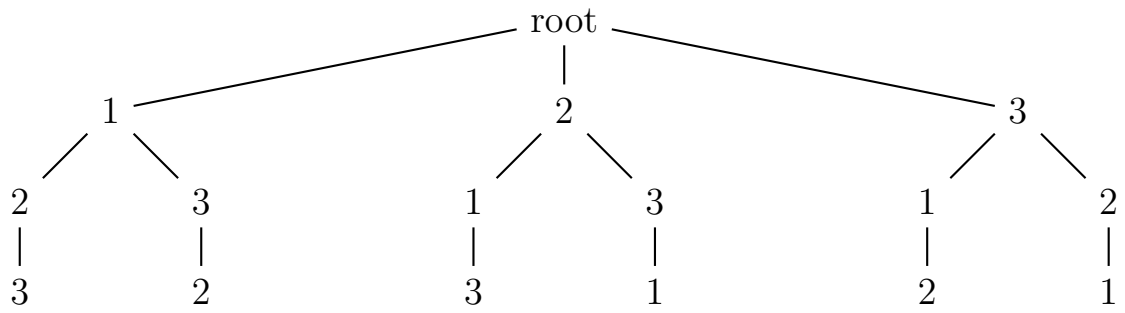
Lab07-Trees

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

∗ Please upload your assignment to website. Contact webmaster for any questions.
∗ Name: Sun Yiwen Student ID: 517370910213 Email: sunyw99@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.

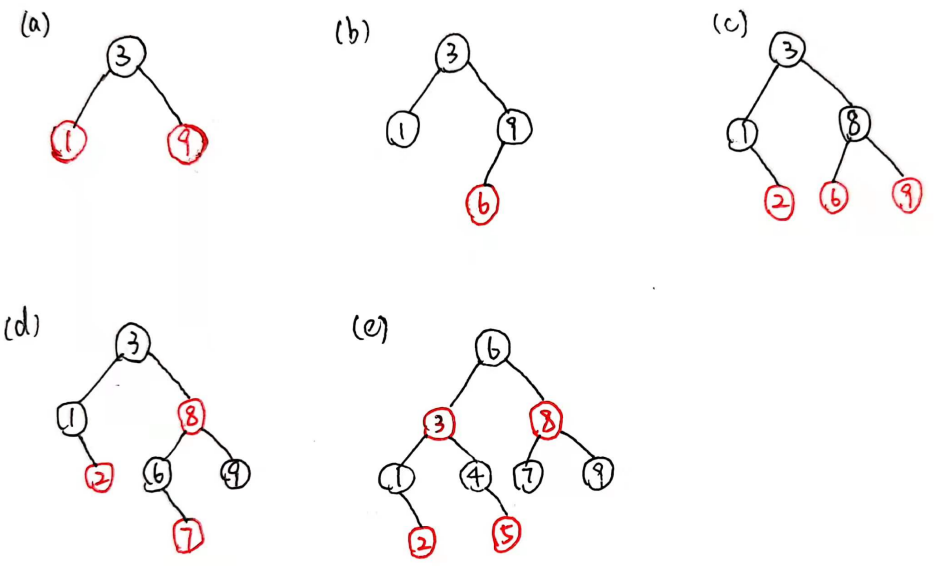
   (c) Suppose that we further insert keys 2, 8 into the red-black tree you get in Problem (1-b). Draw the resulting red-black tree.

   (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.

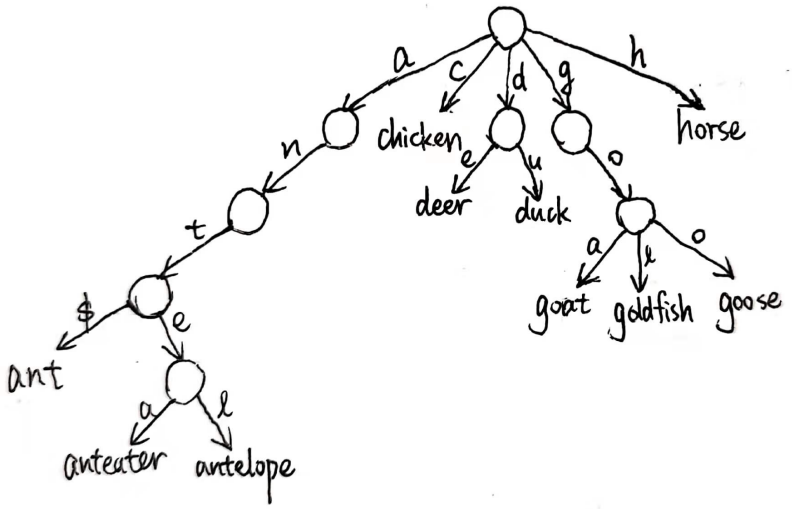
   Solution.



□

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

Solution.

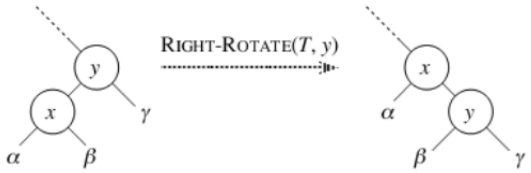


□

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.
As long as the tree is not transformed into a right-skewed tree (which is a right-going chain), repeatedly find some node y on the right spine that has a left child x and then perform a right rotation on y, as shown in the following figure:



This step of right rotation increases the number of nodes in the right spine by 1 and decreases the number of nodes not in the right spine by 1. Any binary search tree starts out with at least one node (the root) in the right spine. Moreover, if there are any nodes not on the right spine, then at least it has a ancestor on the right spine. Thus, at most n-1 right rotations are needed to put all nodes in the right spine, and the tree becomes a right-skewed tree.
If we know how to convert any arbitrary n-node binary search tree into an n-node right-skewed tree, than we know how to convert this right-skewed tree into any arbitrary n-node binary search tree. What we need to do is to reverse the sequence of the right rotation steps and change them into their inverse left rotation.
Therefore, we have proved that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations. □

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) Suppose the balance factor of node P before insertion is x. Correspondingly, the balance factor of node P after insertion and before applying rotation to fix the violation is $x+1$ because the key is inserted into the left subtree of P. Since node P is the first node that has a balance condition violation in the insertion access path from the leaf, we get that $x < 2$ and $x + 1 \geq 2$. Therefore, we get $x = 1$. 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 has only three possible value:1,0,-1, or else it would have break the AVL balance already.

i. If before insertion, the balance factor of A is 1.
If the key is inserted into the left subtree of A, then the balance factor of A after insertion is 2 and A becomes the first node that has a balance condition violation. This contradicts with the problem statement. If the key is inserted into the right subtree of A, then the height of the left subtree of P remains unchanged and the balance factor of P remains 1 and will not break the AVL balance, which also contradicts with the problem statement. Thus, the balance factor of A before insertion can't be 1.

ii. If before insertion, the balance factor of A is -1.
If the key is inserted into the right subtree of A, then the balance factor of A after insertion is -2 and A becomes the first node that has a balance condition violation. This contradicts with the problem statement. If the key is inserted into the left subtree of A, then the height of the left subtree of P remains unchanged and the balance factor of P remains 1 and will not break the AVL balance, which also contradicts with the problem statement. Thus, the balance factor of A before insertion can't be -1.

iii. Therefore, the balance factor of A before insertion must be 0. Also, if we want P to be the first node that violates the AVL balance, then the insertion must increase the height of the left subtree of P by 1. If after one insertion, the balance factor of A is still 1, then the height of P's left subtree will not change. Therefore, the balance factor of node A after insertion and before fixing the violation cannot be 0.

□