# 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:______________ Student ID:______________ Email: ___________________

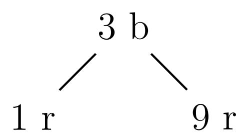
**Hint:** You can use the package **tikz** to draw trees.

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.

   **Solution.**

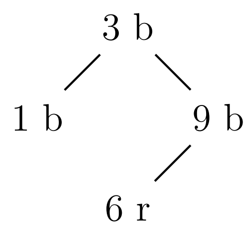
   □




   (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.

   **Solution.**

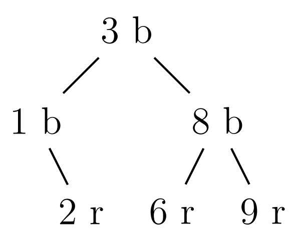
   □




   (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.

   **Solution.**

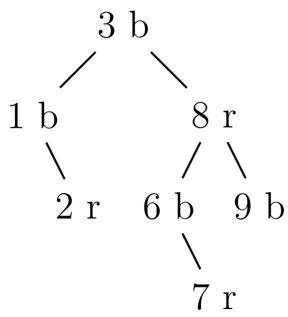
   □




   (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.

   **Solution.**

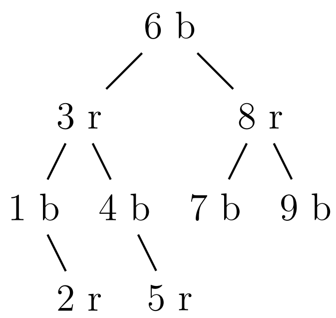
   □




(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.

**Solution.**

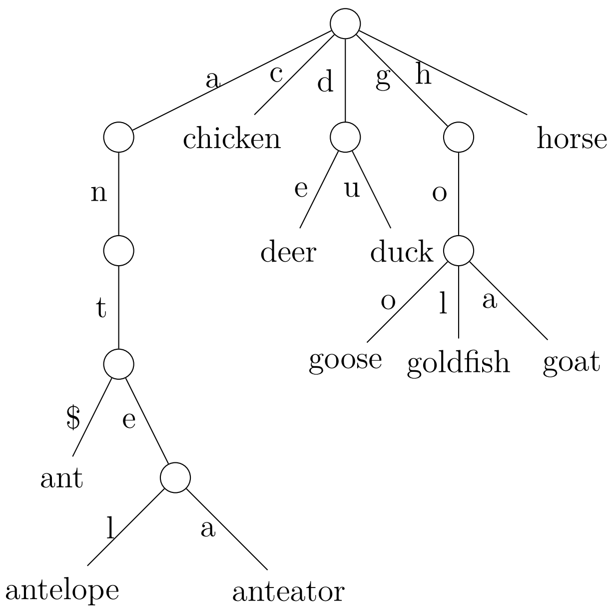
□




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.**
   **First step: to prove at most $n-1$ right rotations suffice to transform the tree into a right-skewed binary search tree.**
   We can use a mathematical induction to prove it:
   1.When n=2 and the tree is a left-skewed tree, 1 rotation is needed to transform the tree.
   2.Suppose for trees which has node number=$n(n = 2, 3, 4, ..., (k-1))$, at most $(n-1)$ rotations are needed.
   3.When $n = k$, we can split the tree into its root, its left subtree, and its right subtree, suppose there are $a$ elements in its left subtree and $b$ elements in its right subtree, and $a + b = n - 1$. if $b > 0$, we combine the left subtree and the root as a whole tree while leaving the right subtree unchanged, we rotate the combined trees made up of the root node and the left subtree to get a right-skewed binary tree. Since the overall nodes is $(a + 1)$, and $a + 1 \le n$, so at most we need $a$ right rotations. Then, we rotate the right subtree into a right-skewed binary tree, since $b < n$, at most we need $(b - 1)$ rotations. therefore, we need at most $(a + b - 1)$ right rotations to transform the tree into a right-skewed tree, which is also equal to $(n - 2)$ rotations.
   if $b = 0$, then $a = n - 1$, therefore, at most we need $(n - 1)$ rotations.
   **Therefore, we can prove by induction that at most $n-1$ right rotations suffice to transform the tree into a right-skewed binary search tree.**

   **Second step: to prove a binary search tree can be transformed into any other n-node binary trees using $O(n)$ rotations.**
   Since we can freely rotate in either right or left direction, so any n-node binary tree with same keys but different structure can be rotated to at last. If the original tree is named as $A$, and we want to transform it into an arbitrary binary tree $B$. We can firstly transform $A$ into right-skewed subtree, which will cost at most $(n - 1)$ rotations and then transform the right-skewed subtree into $B$. Transforming the right-skewed subtree into B is the reverse of transforming $B$ into the right-skewed subtrees, so it will also cost at most $(n - 1)$ rotations. Therefore, **we need $O(n)$ rotations to transform $A$ into any other binary trees with n nodes**.

   □

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.** For each node, after insertion, at most 1 subtree will change its height.

(a) Before insertion, since $P$ is balanced, $B_P \in \{-1, 0, 1\}$. Since the key is inserted in the left subtree of $P$, the height of left subtree will either increase by one or remain the same and that of right tree will not change. Therefore, after insertion $B_P \in \{-1, 0, 1, -2\}$. Since $P$ has a balance violation, $B_P = 2$. Correspondingly before insertion $B_P = 1$.

(b) From (a) we have known that the height of left subtree, subtree $A$ here, has increased by one. If $B_A = 1$ (resp. -1), then after insertion we have $B_A = 2$ (resp. -2). $A$ is not balanced. Therefore, $B_A = 0$. After insertion, since height of $A$ changed and at most one subtree of $A$ changed its height while another remains the same, $B_A$ cannot be 0 any more.

□