S. Schwoon / D. Suwimonteerabuth Winter Term 2008/09

Fundamental Algorithms

Exercise Sheet 7

1. We have seen that the costs of operations on a binary tree such as searching and inserting depend on the height of the tree. Therefore, to ensure the efficiency of such operations one would wish that the tree height always remain reasonably low, ideally $\mathcal{O}(\log n)$, where n is the number of nodes in the tree. One solution is to insist on an extra structural called *balanced*: no node is allowed to get too deep.

   Listed below are simple balance conditions that one might come up at first. Why are they not sufficient? Justify your answers.

   (a) The left and right subtrees of the root have the same height.

   (b) The height of the left and right subtrees of the root can differ at most 1.

   (c) Every node must have left and right subtrees of the same height.

2. An AVL tree is a binary tree with the following balance condition: for every node in the tree, the height of the left and right subtrees can differ by at most 1. It can be shown that the height of an AVL tree is always $\mathcal{O}(\log n)$, ensuring that the tree operations can be performed in $\mathcal{O}(\log n)$ time. However, care must be taken when doing an insertion, since the new node could violate the AVL tree property. It turns out that if this is the case, then the property can be restored by a simple modification of the tree, known as a *rotation*.

   Let p be the node that must be rebalanced after an insertion. Since the height of its left and right subtrees differs exactly by two and any node has at most two children, it is easy to see that the violation is a consequence of one of the following four situations (the first two are cases 1 and 2 in the lecture, respectively):

   (i) The new node was inserted into the left subtree of the left child of p.

   (ii) The new node was inserted into the right subtree of the left child of p.

   (iii) The new node was inserted into the left subtree of the right child of p.

   (iv) The new node was inserted into the right subtree of the right child of p.

   Note that situations (i) and (iv) are symmetric, as well as situations (ii) and (iii). We usually call the rebalancing where the insertion occured on the "outside" (i.e. left-left or right-right) a *single* rotation of the tree, and on the "inside" (i.e. left-right or right-left) a *double* rotation of the tree.

   Figure 1, repeated from the lecture, shows the single *left* rotation that rebalances situation (i), where the trees on the left and right are before and after the rotation, respectively. Analogously, we define the single rotation that rebalances situation (iv) the *right* rotation.

   (a) Write down an algorithm for a double rotation (situation (ii) or (iii)) by using single left and right rotations.

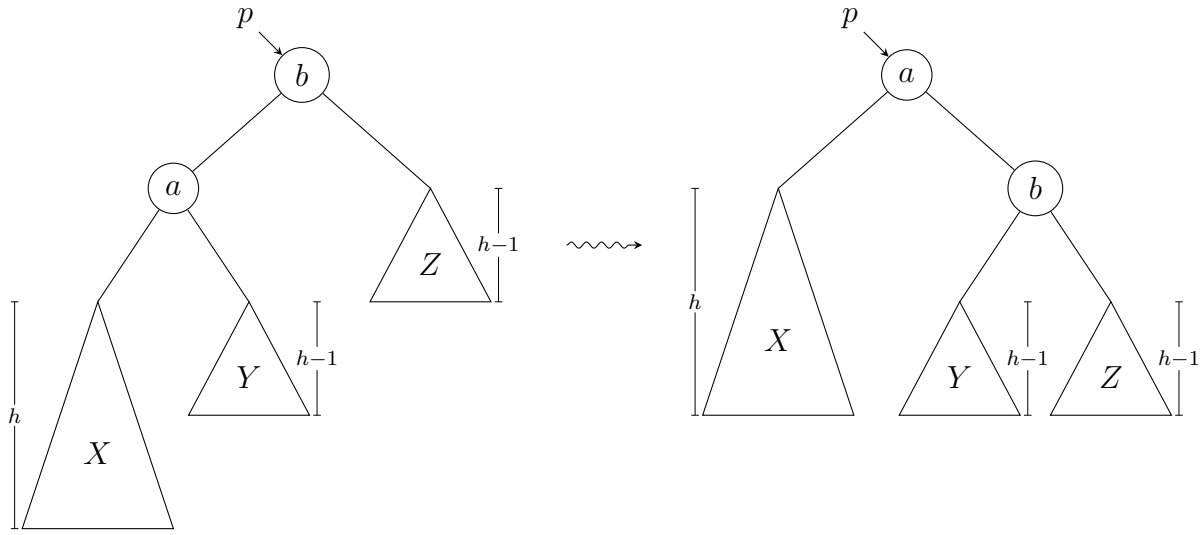

Figure 1: Single left rotation to rebalance situation 1

(b) Show the result of inserting $3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10, 8, 9$ into an initially empty AVL tree.

3. Let n be the minimal number of nodes in an AVL tree of height h. Given a height h, write a method to generate an AVL tree of height h with n nodes, assuming that values of the nodes are distinct and in the range $[1, n]$.