Lab 2

Implementing AVL Trees

1. Purpose

This lab assignment focuses on balanced binary search trees.

2. Background

- An AVL (Adelson-Velskii and Landis) tree is a binary search tree with a balance condition.

- The balance condition is easy to maintain, and it ensures that the depth of the tree is O(lg n).

- An AVL tree is identical to a binary search tree, except that for every node in the tree, the height of the left and right subtrees can differ by at most 1. (The height of an empty tree is defined to be 0.)

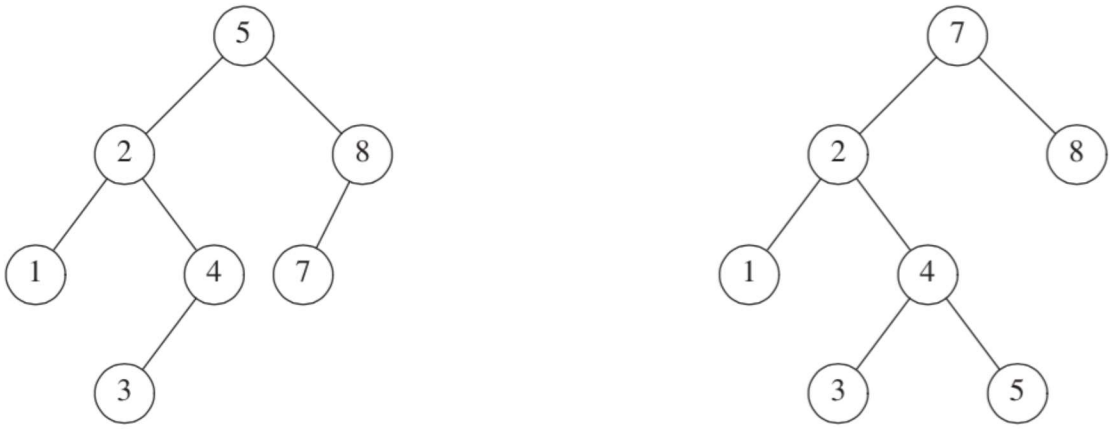

Figure 1: Two binary search trees. Only the left tree is AVL

- In Figure 1 the tree on the left is an AVL tree, but the tree on the right is not. Balancing information is kept for each node (in the node structure).
- All the tree operations can be performed in O(lg n) time. The reason that insertions and deletions are potentially difficult is that the operation could violate the AVL tree property.
- The balance of an AVL tree can be maintained with a simple modification to the tree, known as a rotation.

For instance, inserting 6 into the AVL tree in Figure 1 would destroy the balance condition at the node with key

2.1 Rotations

Let us call the node that must be rebalanced **α**. Since any node has at most two children, and a height imbalance requires that **α** 's two subtrees height differ by two, it is easy to see that a violation might occur in four cases:
1. An insertion into the left subtree of the left child of **α**.
2. An insertion into the right subtree of the left child of **α**.
3. An insertion into the left subtree of the right child of **α**.
4. An insertion into the right subtree of the right child of **α**.

Cases 1 and 4 are mirror image symmetries with respect to, as are cases 2 and 3.

The first case, in which the insertion occurs on the “outside" (i.e., left-left or right-right), is fixed by a single rotation of the tree.

The second case, in which the insertion occurs on the “inside"(i.e., left-right or right-left) is handled by the slightly more complex double rotation.

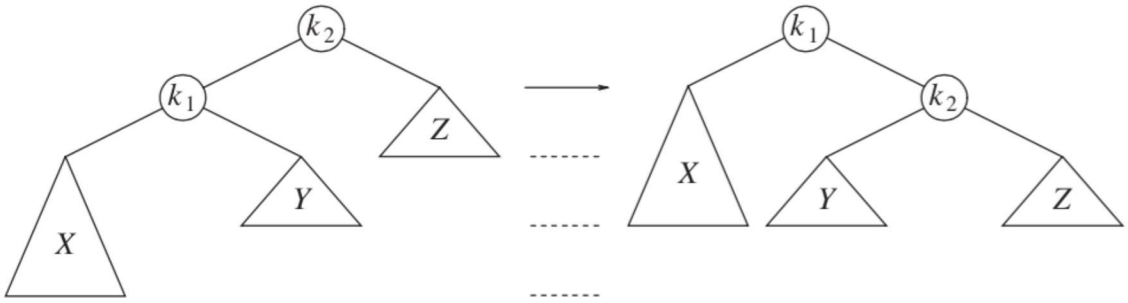

Figure 2: Single rotation to fix case 1

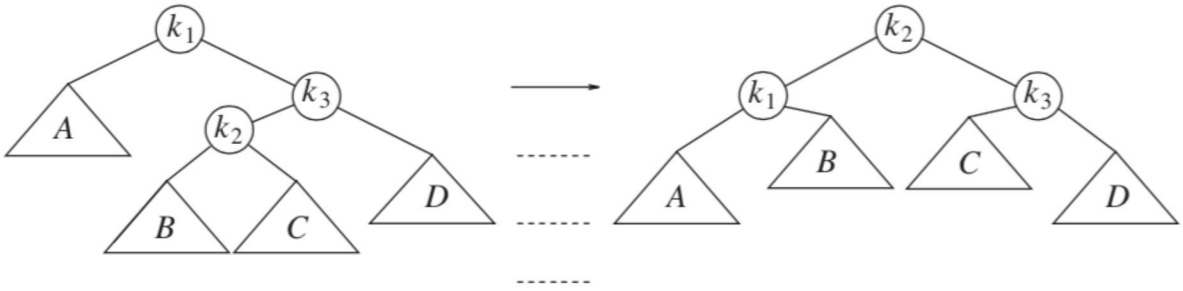

Figure 3: Right left double rotation to fix case 3

3 Requirements

3.1 AVL Tree Implementation

You are required to implement the AVL Tree data structure supporting the following operations:

1. Search: Search for a specific element in an AVL Tree.

2. Insertion: Insert a new node in an AVL tree. Tree balance must be maintained via the rotation operations.

3. Deletions: Delete a node from an AVL tree. Tree balance must be maintained via the rotation operations.

4. Print Tree Height: Print the height of the AVL tree. This is the longest path from the root to a leaf-node. Please refer to the reference below for more implementation details.

3.2 Application: English Dictionary

As an application based on your AVL Tree implementation, you are required to implement a simple English dictionary, with a simple text-based user interface, supporting the following functionalities:

- **Load Dictionary**:
  - You will be provided with a text file, “dictionary.txt", containing a list of words. Each word will be in a separate line.
  - You should load the dictionary into an AVL Tree data structure to support efficient insertions, deletions and search operations.

- **Print Dictionary Size:**
  - Prints the current size of your dictionary.

- **Insert Word:**
  - Takes a word from the user and inserts it, only if it is not already in the dictionary. Otherwise, print the appropriate error message (e.g. “ERROR: Word already in the dictionary!").

- **Look-up a Word:**
  - Takes a word from the user and prints “YES" or “NO" according to whether it is found or not.

- **Remove Word:**
  - Takes a word from the user and removes it from the dictionary. If the word is not in the dictionary, then print the appropriate error message.

- **Batch Look-ups**:
  - You will be provided with a text file, “queries.txt", containing a list of words to look up in your dictionary.
  - You are required to print the total number of found words. Also, for each word, you should print the word and “YES" if it exists in your dictionary. Otherwise, print “NO".

- **Batch Deletions:**
    - You will be provided with a text file “deletions.txt" containing a list of words to remove from your dictionary.

Note: For validation purposes, you are required to print both the size of the dictionary and the height of your AVL tree after each insertion or deletion. When performing batch insertions/deletions, you may print the height only once after the performed operation.

4. References

Weiss, Mark Allen. “Data structures and algorithm analysis in Java." Addison-Wesley Long-man Publishing Co., Inc., 1998.

5. Notes

- Implement your algorithms using (Java or C/C++)

- You should work in groups **of 2 members.**

Good Luck