CSE 2320 Lab Assignment 4

Due August 5, 2011

Goals:

1. Understanding of red-black trees.

2. Understanding of recursive binary tree processing.

3. Understanding of subtree *root ranks* in binary search trees for supporting ranking queries.

Requirements:

0. Code related to this lab that uses *subtree sizes* is available from the course website.

1. Modify the provided Java code for maintaining a red-black tree to process a sequence of commands (standard input) from the following list:

   `0` - Exit the program

   `1` *x* - Insert positive integer key *x*, unless *x* is already present in the tree. Besides inserting the key, subtree root ranks must be updated. (Error message if *x* is a duplicate.)

   `2` *x* - Find the smallest key in the tree that is not smaller than *x* (error message if there is no such key).

   `3` *x* - Find the largest key in the tree that is not larger than *x* (error message if there is no such key).

   `4` *x y* - List all keys (5 per line) that are not smaller than *x* and not larger than *y*.

   `5` *x* - Find the rank of *x*, i.e. the number of keys in the tree that are no larger than *x* (error message if *x* is not in the tree).

   `6` *k* - Find the key with rank *k* (error message if *k* is not between 1 and *n*, inclusive).

   `7` - Perform an audit on the subtree rank at each node to give a final indication that the tree is "clean" or "corrupt".

   Each command must be echoed to standard output. Commands 1, 2, 3, 5, and 6 must be processed in $\Theta(\log n)$ time. Command 4 must be processed in $\Theta(\log n + m)$ time, where *m* is the number of keys output.

2. Email your program source files to `hafizfahad.sheikh@mavs.uta.edu` by 12:45 pm on August 5.

Getting Started:

1. The code changes for maintaining subtree root ranks during command `1` are minor and will include the rotation code.

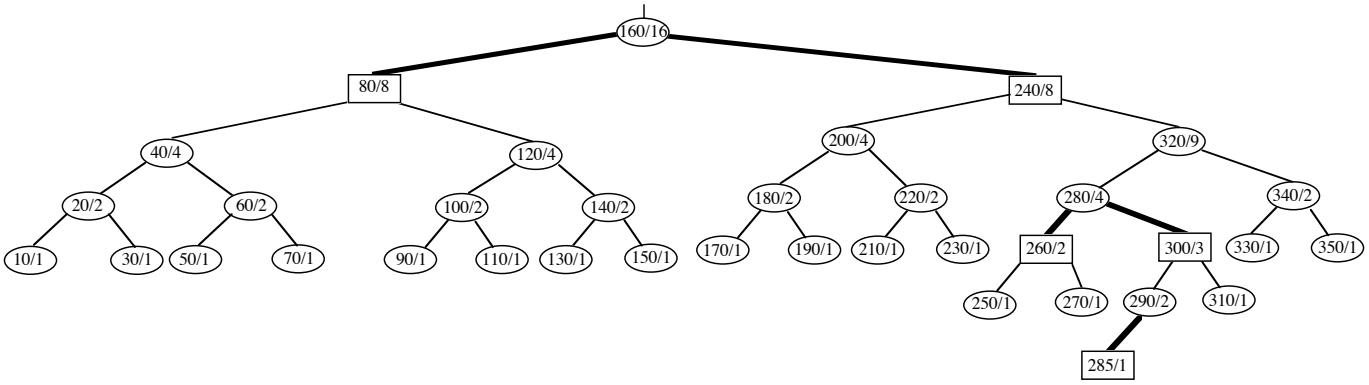
2. The code for commands `2`, `3`, and `4` may be reused without change.

3. The code for commands `5` and `6` may be coded as variations on recursive search of a binary search tree.

4. Command `7` should traverse the tree, compute the rank of the root of each subtree, and verify the stored ranks. Optionally, you could also check 1) the inorder key property, 2) that there is no downward path with consecutive red nodes, and 3) the black-heights are consistent.

5. You are not required to maintain any satellite data related to the integer keys.

6. An example of a red-black tree with subtree root ranks:



7. The same tree as 6., but using subtree sizes: