CSE 2320 Lab Assignment 3

Due April 11, 2013

Goals:

1. Understanding of red-black trees.

2. Understanding of recursive binary tree processing.

3. Understanding of subtree sizes in binary search trees for supporting ranking queries.

Requirements:

1. Modify the provided C code for maintaining a red-black tree to perform the following processing:

   a. Read a value *n* and then read and process *n* values, one per line, by inserting into a red-black tree. There will be no duplicates and a message should be printed for each insert. After completing all inserts, use `printTree()` to dump the tree.

   b. Read and process a sequence of retrieval commands on the tree that was created:

      `0` - Exit the program

      `1` *x* - sets the *current node* to the node with the smallest key that is not smaller than *x*. After processing, print 1) the current key (not necessarily *x*) and 2) the current rank. $\Theta(\log n)$ time

      `2` *k* - sets the current node to the node with rank *k*. After processing, print 1) the current key and 2) the current rank. $\Theta(\log n)$ time

      `3` *j k* - performs *j* iterations of moving the current node pointer forward *k* nodes in an inorder traversal *taking advantage of subtree sizes*. After each move, print 1) the current key and 2) the current rank. $\Theta(\log n + j \log k)$ time

      `4` *j k* - performs *j* iterations of moving the current node pointer backward *k* nodes in an inorder traversal taking advantage of subtree sizes. After each move, print 1) the current key and 2) the current rank. $\Theta(\log n + j \log k)$ time

      x, *j*, and *k* are non-negative values.

      If a command cannot be processed completely, print a message and terminate the entire execution. The sequence of commands may not begin with a `3` or a `4`.

2. Email your program source files to `mdmehrab.shahriar@mavs.uta.edu` by 3:15 p.m. on April 11.

Getting Started:

1. A small number of global variables are permissible for handling the retrieval commands.

2. The retrieval commands will be processed by one recursive function (a "treewalker"). After processing a command, you will be nested within a number of function calls. At this point, get the next command. If the next command is a `1` or `2`, you may retreat out of the recursion before starting to search.