CSE 3318 Lab Assignment 4

Due November 29

Goals:

1. Understanding of unbalanced binary search trees.

2. Understanding of tombstones as a simple mechanism for supporting deletions in a data structure.

Requirements:

1. Create C code for maintaining an unbalanced binary search tree (with a sentinel at the bottom) 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 sizes must be updated. (Processing a duplicate *x* is handled as an update, even though there is no satellite data.) Also, see command `6` regarding the recycling list.

   `2` *x* - Logically delete the item for positive integer key *x* by using a *tombstone*. If there is no item, then the operation is ignored. Subtree sizes must be updated.

   `3` *x* - Find the rank of *x*, i.e. the number of keys in the tree that are not larger than *x* (error message if *x* is not in the tree). Tombstoned items are not included!

   `4` *k* - Find the key with rank *k* (error message if *k* is not legal). Tombstoned items are not included!

   `5` - Print statistics - number of live nodes, number of dead nodes, number of nodes in the recycling list (which is likely to vary from the provided output), and the number of nodes on the longest path to the sentinel.

   `6` - Rebuild tree by collecting live nodes in an ascending order linked list and placing dead nodes on a recycling list (to be used later when inserting). The tree is then (recursively) rebuilt using an inorder approach *without* calling the insertion code. (The sample program `load.c` shows how to do this using an ordered input file instead of the list of live nodes.)

   `7` - Print the tree as is done in `load.c`, but indicating dead keys with surrounding parentheses and also outputting the subtree sizes.

   `8` - Perform an audit to check the tree for the inorder traversal property and correct subtree sizes (number of live nodes in each subtree) to give a final indication that the tree is "clean" or "corrupt".

   Each command must be echoed to standard output. Commands `1`, `2`, `3`, and `4` must be processed in $\Theta(h)$ time, where *h* is the tree height. Commands `6` and `8` must be processed in $\Theta(n)$ time. For command `5`, the first three statistics must be processed in $\Theta(1)$ time and the last statistic in $\Theta(n)$ time.

2. Submit your C source files as *a single zipped file* on Canvas by 5:00 pm on November 29. One of the comment lines should include the compilation command used on OMEGA. You will lose points for putting all code in a single source file.

Getting Started:

1. A suitable driver is available at `https://ranger.uta.edu/~weems/NOTES3318/LAB/LAB4FALL22/lab4fall22.c`

2. A useful header file for your unbalanced binary search tree implementation is available at: `https://ranger.uta.edu/~weems/NOTES3318/LAB/LAB4FALL22/bst.h`. You are allowed to modify this file.

3. The recursive code on p. 5 of Notes 11 may be modified for processing commands `3` and `4`.