CSE 5311 Lab Assignment 1

Due June 29, 2005

Goals:

1. Review of red-black trees.

2. Understanding of a variation on red-black trees.

Requirements:

1. Write (and test) a C or C++ program that processes the commands below based on the red-black tree variant in the attached paper by C. Okasaki. Each command appears on a separate line. Your program must compile and execute on OMEGA. There should be a comment near the beginning of your code that indicates how to compile on OMEGA. Be sure that your program follows the requirements under "Getting Started".

   a. `1 x` - insert key `x` into the tree. If `x` is already present, leave the tree alone. Nothing is to be output.

   b. `2 x` - search for key `x` in the tree. Output (on the same line) each key on the search path. If the key is found, it should appear last on the line. Otherwise, print the negation of `x`.

   c. `3` - output the keys in ascending order. After outputting the current number of keys on a line, output each key with its color (`RED`, `BLACK`) and the number of black nodes (including itself) on the path from the root. There should be no output for the sentinel.

   d. `4` - terminate the program.

2. Email your code (as attachments) to `ghosh@cse.uta.edu` before 3:45 pm on June 29. The subject should include your name as recorded by the University.

Getting Started:

1. Figures 1. and 2. in the attached paper describe a variation of red-black trees that may use O(log n) rotations for performing a single insertion. This variation has no counterpart to the color flips that occur in insertion case 1 in CLRS.

2. Do not prompt for a file name. Use a shell redirect (<) to access test case files.

3. Tracing should be deactivated in the version that you submit.

4. The number of keys in the tree will not exceed 20,000.

5. All keys will be positive.

6. The pseudocode will lead to an O(log n) implementation for search and insertion, but could be improved:

   a. `IsRed()` should not test for a `NULL` pointer.

   b. The recursive function `Ins()` is based on two phases, searching down the tree and rebalancing up the tree. This division leads to the possibility of testing for all four cases in Figure 1, even when none of the cases are needed. Your code should be structured, regardless of whether recursion is used, to test for no more than one of these cases at a node.