University of North Carolina at Greensboro
CSC 330: Advanced Data Structures
Prof. Stephen R. Tate

Handout 7
October 5, 2009

Assignment 3 — Performance of Red-Black Trees

Due: Monday, October 19

Objective: The objective of this assignment is for students to experiment with red-black trees to gain insight into typical performance of this data structure.

High-Level Description: The performance of lookup operations in a red-black tree depends on the depth of the tree, which is guaranteed to be at most $2\log_2(n+1)$ (see page 702 of the textbook). This formula gives an absolute, guaranteed upper-bound on performance, but leaves open the question of what *typical* performance is. For this assignment, you are to modify the word-counting program from Assignment 1 to use the red-black tree implementation provided by the textbook authors in file `d_rbtree.h` (you may start from either your solution to Assignment 1 or the instructor-written solution), and then print statistics out about the actual red-black tree that has been constructed.

Details: For this assignment you will be modifying both the word-counting program from Assignment 1 and the file `d_rbtree.h`. Note that, like in Assignment 1, the version of files distributed by the textbook authors doesn't work on modern compilers, so you should download the modified versions from the class web site. You should add a public method `printStats()` to the `rbtree` class (adding the code to `d_rbtree.h`) that prints the following values: The calculated minimum and maximum possible depths for any red-black tree with this many nodes, the actual depth of this tree, the average depth of the tree where each node is equally likely, and the average depth of the tree where nodes are weighted by the count of the number of times the word occurs (more details below and discussed in class).

To calculate the minimum and maximum values, it is useful to be able to calculate logarithms, floors, and ceilings. These functions are available if you put `#include <cmath>` at the top of your file. The signatures for the functions you might need are:

```
double log2(double x);
double ceil(double x);
double floor(double x);
```

The average depth (unweighted and weighted) gives an indication of average lookup time in the tree. The idea with the unweighted average depth is this: if you pick a word at random, where all words have equal probability of being chosen, what is the expected depth of the node containing that word? For the weighted average the idea is the same, except that words don't have equal probabilities — the words have probabilities that are equal to their proportion of occurrence in the input.

Mathematically, consider that we have n different words in the file, with counts $c_1, c_2, \ldots, c_n$. After constructing the tree, these words are at depths $d_1, d_2, \ldots, d_n$. The unweighted average depth and weighted average depth are given by

$$\frac{1}{n}\sum_{i=1}^{n} d_i \qquad \text{and} \qquad \frac{\sum_{i=1}^{n} c_i d_i}{\sum_{i=1}^{n} c_i}\,.$$

To get counts, simply write your function that computes the weighted average depth assuming that it will only be called with a `pairClass` object in the nodes, and you can call `getCount()` (or whatever you called this method in your class). This is a huge assumption, and such code may not work with all C++ compilers — however, it does work on the `g++` compiler on our `cmpunix` system.

To Turn In: Use the `330submit` program (see Handout 3) in order to turn in your code, using assignment name `assign3`. You need to turn in a printout of your code in class, and if you do the extra credit (see below) you should turn that written solution in along with your printout.

Extra Credit (up to 20 points): What is the absolute worst-case red-black tree that you can construct (in other words, the largest possible depth for some number n of nodes)? Can you make a tree that actually has depth $2\log_2(n+1)$? To answer this question, describe a general process for creating a worst-case tree — it should be possible to follow your construction process to make larger and larger trees following the same process/pattern. After clearly describing the process, analyze the relation between the number of nodes (n) and the depth (d). Can you get close to $d = 2\log_2(n+1)$?

Sample Input/Output:

SAMPLE INPUT

```
Peter Piper picked a peck of pickled peppers,
A peck of pickled peppers Peter Piper picked;
If Peter Piper picked a peck of pickled peppers,
Where's the peck of pickled peppers Peter Piper picked?
```

SAMPLE OUTPUT

```
Minimum possible depth: 3
Maximum possible depth: 7
Actual depth of tree: 3
Average (unweighted) depth: 2
Average (weighted) depth: 1.88235
```