Assignment 4: Red-Black Trees
CS 758/858, Spring 2014
Due Monday, September 29 by **1:40pm**

Implementation

The skeleton code on the course web page is the start of a program to schedule I/O requests to a hard disk. Each I/O request is for a specific track and sector. Your program must track all incoming requests and then, when told where the drive head is and which direction it is traveling, be able to return the next n I/O requests that lie in that direction, where n is a supplied parameter. (This is the 'elevator' method of I/O scheduling.) We will make the simplifying assumption that requests can be ordered lexicographically, i.e., the first sector on track $i + 1$ is further away than the last sector on track i. Currently, the program uses a simple binary tree that gives poor performance for many workloads. Improve the program by implementing a balanced tree data structure and the functions for using it. Then measure the performance improvement.

Those in 858 will need to support canceling I/O requests (for example, after they have been completed), which will involve deleting nodes from the red-black tree.

Testing

On the course web page, we supply skeleton code and a test harness. Most of the programs we distribute in this class will tell you their command-line arguments if you run them with the `-help` option.

`disk-sched-harness` runs your program, checks its output, and optionally displays a plot of the performance. For example:

```
disk-sched-harness -d -a list -a bst 100 1000 10000
```

will run the disk scheduler on workloads of size 100, 1000 and 10000 using both a list and binary search tree. If you are running the harness on a system without an X display then you may use the `-o <filesuffix>` option to output the plots to files instead (suffix should end in ".pdf",".ps" or ".png").

To vary the test data, the `--sorted` and `--partial` flags produce sequential and mostly sequential requests, respectively. You might find this useful when you're comparing the relative performance of the data structures.

Written Problems

1. Is there anything special that we should know when evaluating your implementation work?
2. If you didn't do it in class, write down precise psuedo-code for binary tree insertion without looking at any books or notes.

   Now, prove that your psuedo-code is correct. If you discover a flaw, you are allowed to write new psuedo-code without any penalty, but please turn in both the original and the final versions.
3. Exercise 13.1–7 in CLRS.
4. Exercise 13.3–1 in CLRS.
5. Exercise 13.3–4 in CLRS.
6. (Those in 858 only) Exercise 13.4–6 in CLRS.
7. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit your source code using the script on agate (eg, `~cs758/scripts/sub758 4 your-asn4-dir`). Also hand in a listing of your source code (2 pages per page, as with `a2ps -2`), along with your written work, to the TA in class.

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.
Tentative breakdown:

6 tree implementation

4 written problems