Assignment 4: Red-Black Trees
CS 758/858, Fall 2024
Due at **11:30pm on Wed, Sep 18**

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 record 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. It's up to you how closely you want to follow the book's pseudocode; if you want to implement an actual 'nil' node, that's fine.

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 (try `-help`).

`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 can use the `-o <filesuffix>` option to write the plots to files instead (suffix should end in ".pdf",".ps" or ".png"). The green dashed line is mean insertion time, the red dashed line is mean scan time.

To vary the test data, the `--sorted` flag produces sequential requests. You might find this useful when you're comparing the relative performance of the data structures.

It is possible to make the harness segfault if your iterator is buggy. The harness also doesn't like leading whitespace.

Written Problems

1. Briefly list any parts of your program which are not fully working. Include transcripts or plots showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?
2. What kind of behavior do you expect from your program when run with various inputs? Test your hypotheses by generating runtime scaling plots.
3. Write 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 change your psuedo-code until it is correct.

4. Exercise 12.2–4 from CLRS.
5. (Those in 858 only) Exercise 12.3-3 from CLRS.
6. Exercise 13.1–7 in CLRS.
7. Exercise 13.3–1 in CLRS.
8. Exercise 13.3–4 in CLRS.
9. (Those in 858 only) Exercise 13.4–7 in CLRS.
10. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit your work as usual (eg, `~cs758/scripts/sub758 4-undergrad your-asn4-dir`). Your files should include your `written.pdf` as well (feel free to use LaTeX or to scan/photograph handwritten work) and plots showing the performance of your implementation.

Evaluation

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

6 tree implementation

4 written problems