1 Introduction

In this homework, we are faced with a single-threaded binary tree implementation that is running much slower than expected. Your job is to figure out what's a reasonable performance estimate, identify the bugs, describe how the bugs caused a performance degradation (and any other problems), and fix them.

2 Prerequisites

The main tools you need for this homework are profiling tools, gdb, and printf.

3 Assignment

The template for this assignment is available at `https://github.com/uicperformance/slowtree`. To build the program, simply run `make` in the checked-out repository folder.

The program, `benchmark_tree_single_thread` when run with the parameters `-i 100000 -r 200000 -d 3000` takes so long to run that you may not have the patience to let it finish. (You could try with `-i 10000 -r 20000 -d 3000` at first.) This is despite the fact that `-d 3000` means "run for 3 seconds".

The `-i 100000` says to start with a tree with 100,000 random keys, then search for (50%), delete (25%) and insert (25%) random key-value pairs. The `-r 200000` sets the key range to 0–200000.

3.1 Guesstimate Performance

Knowing that this is a binary tree implementation, and that keys are selected uniformly at random from a set range, make some quick predictions about the performance of this program. (a) what should be the size of the tree at the end of the run. (b) what should be the depth of the tree? (c) how long should one operation (of any kind) take? Provide a back-of-the-envelope justification.

3.2 Identify bugs

- One bug is that the program is not guaranteed to finish after 3 seconds: it could be a little more, or possibly a lot more, depending on data structure performance and size. Describe the cause of this erroneous behavior, and fix the bug.
- An implementation bug dramatically increases the complexity of operations. It also causes data loss: the data structure doesn't have a test suite, so the developers totally missed that.
- Finally, a small design change could significantly improve cache performance and thereby data structure throughput (probably by more than

10 times on your machine), without altering the complexity, algorithm or functionality of the data structure.

3.3 turn-in instructions

Similar to the previous homework, turn-in is by git classroom, using this invitation link `https://classroom.github.com/a/V7Jfrq3A`. Add a PDF describing the bugs you identified, called hw3.pdf, to your template folder. Fix the bugs, and make sure your program comiles and runs connectly. Then commit, then push it to the turn-in repository as in hw1.

3.4 Double-check your submission

To make sure that you submitted everything you think you submitted (`git` can be a little tricky until you get used to it), *git clone* your turn-in repository into a fresh folder, check that your `report.pdf` is in the splash2 folder, and that RAYTRACE compiles as you would expect.