CSCI 152, Performance and Data Structures, Lab Exercise 7

Deadline: April 21st 11PM

1 Introduction

In this exercise, you will measure the performance of your own implementation of map ADT and compare it with the two implementations of map in the C^{++} standard template library. You will also compare all three implementation with the theoretical performance as explained in the lecture.

In order to make the measurements, you must use the same tools as were used in Exercise 6. Reread the explanations at the top of Exercise 6. When making measurements, remember the following:

- Make sure that no other processes are running on your computer. (No web browsers, no music playing.)
- If you are using a laptop, make sure that power is connected. Most laptops are slower when using battery, to save energy.
- Make sure that the total run time is not less than 10 seconds. Short run times (less than 0.1 second) are meaningless because the start up times will spoil the measurements.
- Make sure that no updates are being downloaded.
- Make sure that compiler optimization is `-O3 -flto`. Don't use valgrind during measurements. If you are not using Linux, you will still need to turn on compiler optimization. Eclipse allows to set flags to the compiler. The flags must be still `-O3 -flto`.

2 Measurements

You need to measure three implementations of the map ADT. All implementations use keytype and value type equal to `std::string`. Use ordinary comparison (case sensitive). The implementations are:

- Your own implementation of **map** based on non-balancing BST, using

```
using keytype = std::string;
using keycmp = standard_cmp< keytype > ;
// using keycmp = case_insensitive_cmp;
using valtype = std::string;
```

- `std::map< std::string, std::string >`. The implementation uses red/black trees.
- `std::unordered_map< std::string, std::string >`. The implementation uses hashing with closed addressing and separate chaining.

Each of the implementations must be measured in two ways:

- Keys are entered in random order.
- Keys are entered in sorted order.

This gives a total of 6 combinations to be tested. For each of the combinations set the condition in the for loop in such a way that the runtime is reasonable, around 30 seconds. Save the tables and include in your **main.cpp**.

For each table, answer the following questions: What seems to be the asymptotic complexity? Is it the same as the expected theoretical complexity? Do you notice anything unexpected? If yes, what?

3 Submission

Submit your **main.cpp**. Don't use an archiver. Include your six performance tables, in a comment or between `#if 0` and `#endif`.

Check that your code can be compiled with the comments in it.

Mark unambiguously which table correponds to which ADT implementation, and whether keys were sorted or unsorted. For each combination, answer the questions above.