EECS 2011: Assignment 3

July 24, 2017
9 % of the course grade
Due: Monday, July 31, 2017, 23:59 EDT

Motivation

The purpose of this assignment is to evaluate two implementations of Maps in Java in terms of their performance when used with two kinds of keys.

Introduction

A *Map* is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical *function* abstraction (functions also map a value to a value). The `Map` interface includes methods for basic operations (such as `put`, `get`, `remove`, `containsKey`, `containsValue`, `size`, and `empty`), bulk operations (such as `putAll` and `clear`), and collection views (such as `keySet`, `entrySet`, and `values`).

The Java platform contains three general-purpose Map implementations: `HashMap`, `TreeMap`, and `LinkedHashMap`. Their behavior and performance are precisely analogous to `HashSet`, `TreeSet`, and `LinkedHashSet`.

Description

In this assignment, you will use two implementations of `Map` interface (included in java), one that uses [**balanced Red-Black] Trees**, and the other, based on Hashing (`TreeMap` and `HashMap`, respectively). You will have to test the performance of insert and search operations with these implementations: based on `put`, `containsKey` and `containsValue` methods).

Part 1 (6 points out of 9)

Name your class `MapTester`.
~~Take both of your tree implementations and compare them when used to implement a TreeSort sorting algorithm.~~

You will use four `Map` objects: two (for `Strings` and for `Integers`) `HashMap` and two `TreeMap` objects.

For numbers N = {10, 100, 1000, 10000, 100000, 1000000}
a) Create an *array* of $2N$ random `Integer`-s
Insert half of the (same) numbers into the maps (keep the other half for later), using the same values for both keys and values, and measure how long each operation takes on average (divide the total time by N). This number is to go in the table later.

b) Create an *array* of 2*N* random `String`-s of 16 characters. Feel free to pick any random string generator available online (e.g., [1]; remember to document your sources). Insert half of the (same) strings into the maps (keep the other half for later), using the same values for both keys and values, and measure how long each operation takes on average (divide the total time by *N*). This number is to go in the table as well.

c) take the first half of the numbers and strings, search for random 1/10 of these numbers or strings in both maps, using both `containsKey` and `containsValue`, and measure how long each search takes, on average.

d) take the second half of the numbers and strings, search for random 1/10 of these numbers or strings in both maps, using both `containsKey` and `containsValue`, and measure how long each search takes, on average.

At the end, produce the following table (the timing values below are just placeholders and do not relate to any real measurements):

```
N = 10 (times are per operation):
                            Strings       Numbers
TreeMap, put                456 ms        456 ms
(when key/value present)
TreeMap, containsKey        456 ms        456 ms
TreeMap, containsValue      456 ms        456 ms
(when key/value absent)
TreeMap, containsKey        456 ms        456 ms
TreeMap, containsValue      456 ms        456 ms

HashMap, put                456 ms        456 ms
(when key/value present)
HashMap, containsKey        456 ms        456 ms
HashMap, containsValue      456 ms        456 ms
(when key/value absent)
HashMap, containsKey        456 ms        456 ms
HashMap, containsValue      456 ms        456 ms


N = 100:
 …
N = 1000:
…
<repeat for all values of N>
```

Save the result of your program execution in a file `testrun.txt` and submit it together with your other files.

Part 2 (3 points out of 9)

The suggested length for each of your answers is about three lines. Put your answers in a file `answers.txt` and submit it together with your other files

[1] https://stackoverflow.com/questions/41107/how-to-generate-a-random-alpha-numeric-string

a) When using Java's HashMap, how can one optimize the data structure's performance and space use?

b) Imagine a tree map implementation which uses heaps, instead of binary search trees. How would performance of such a data structure differ from the actual implementation you used in Part 1?

c) How are the TreeMap and TreeSet classes in Java related?

NOTES:
1. Make sure you reset the timer (or save the intermediate time before the next measurement); i.e., make sure you measured time contains only the time to perform one set of operations that was supposed to be timed.

2. In case the operations for larger N numbers take too long (e.g., more than 30 s) you may reduce the number to a smaller one or eliminate it (so that you will have a range from, say, 1 to 100000).

3. Do not use *package*-s in your project (put your classes in a `default` package). Using packages will cost you a 20 % deduction from the assignment mark.

4. Name your classes as specified. Using incorrect names will cost you a 20 % deduction from the assignment mark.

5. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions). It is also imperative you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is.

6. Your code should include Javadoc comments. Also, part of your mark will be based on coding style.

Submission

Submit your work using the `submit` command. Remember that you first need to find your workspace directory, then you need to find your project directory.
`submit 2011 a3 <list of your files>`
(The directory will be created soon).
You can check the usage examples by executing `man submit`.

Alternatively, you may use the web form at
`https://webapp.eecs.yorku.ca/submit/index.php`

You only need to submit 3 files (the tester, the test run, and the answers); optionally, you may also submit a file `readme.txt` containing comments for the marker. Make sure you submit Java source code files, and not the compiled classes.

Late penalty is 20 % per day. Submission 5 days or more after deadline will be given a mark of zero (0). Contact the instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Academic Honesty

Direct collaboration (e.g., sharing code or answers) is not allowed (plagiarism detection software[2] will be employed). However, you're allowed to discuss the questions, ideas, approaches you take, etc.

State all sources you use (online sources, books, etc.). Using textbook examples is allowed (still needs to be cited).

[2] http://theory.stanford.edu/~aiken/moss/