Kaunas University of Technology
Faculty of Informatics

Report

JMH speed tests

P175B014 Data Structures

Author: Rasa Kundrotaite, IFD-0/2
Teacher: Cenker Canbulut

Kaunas, 2021

Table of contents

What is JMH?

Java microbenchmark harness (JMH) is a Java tool used to build, run and analyze nano/micro/mili/macro benchmarks written in Java and other languages targeting the JVM. Needless to say, the JMH tool does not guarantee that a benchmark is implemented correctly. According to its creators, there are multiple optimizations implemented on the JVM/OS or hardware side that make it challenging to measure the performance of a code.

Research

To start with, I have analyzed a few examples online on how JMH speed tests work (source, source). After learning that it is possible to configure JMH in three ways (annotations, Java API, and Command-line), I have started testing the performance of my chosen methods:

Class BstSet: remove()	Class java.util.TreeSet<E>: remove()

I have researched online their asymptotic complexities. *GeeksForGeeks* page has confirmed my hypothesis that for BstSet remove() method asymptotic complexity is O(n). On *StackOverflow* I have found that TreeSet<E> remove() method has O(log(n)) asymptotic complexity.

Tests

I have implemented two benchmark tests. The input data varied from 10000 to 100000. I have chosen to view the average time benchmark mode in µs.

These results can be seen from JMH speed tests:

```
Benchmark                  (elementCount)  Mode  Cnt   Score    Error  Units
Benchmark.removeBstSet              10000  avgt    5   6.378 ±  0.408  us/op
Benchmark.removeBstSet              20000  avgt    5  13.952 ±  0.922  us/op
Benchmark.removeBstSet              40000  avgt    5  29.156 ±  2.499  us/op
Benchmark.removeBstSet              80000  avgt    5  57.983 ± 25.481  us/op
Benchmark.removeBstSet             100000  avgt    5  73.306 ±  7.340  us/op
Benchmark.removeTreeSet             10000  avgt    5   0.011 ±  0.001  us/op
Benchmark.removeTreeSet             20000  avgt    5   0.012 ±  0.004  us/op
Benchmark.removeTreeSet             40000  avgt    5   0.011 ±  0.001  us/op
Benchmark.removeTreeSet             80000  avgt    5   0.012 ±  0.001  us/op
Benchmark.removeTreeSet            100000  avgt    5   0.012 ±  0.001  us/op
```

The main parameters of the device used are below:

- Processor - AMD Ryzen 3 4300U with Radeon Graphics 2.70 GHz;
- Installed RAM - 8.00 GB (7.37 GB usable);
- System type - 64-bit operating system, x64-based processor;
- Pen and touch - not available.

Results

As we can see TreeSet<E> remove() method is being executed much quicker than BstSet remove() method. While BstSet remove() method's speed highly depends on the amount of input data (the more input data we provide, the slower the execution is), it is the opposite with TreeSet<E> remove() method, as its result for 100000 does not diverge much from 10000 input data. These results ideally represent the O(n) asymptotic complexity of the BstSet remove() method and O(log(n)) asymptotic complexity of TreeSet<E> remove() method. Overall, it would be wiser to use TreeSet<E> remove() method as it performs better than the BstSet remove() method.

Sources

1. https://developpaper.com/how-to-benchmark-using-jmh-in-java/
2. https://www.awesome-testing.com/2019/05/performance-testing-benchmarking-java.html
3. https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/
4. https://stackoverflow.com/questions/14379515/computational-complexity-of-treeset-methods-in-java