CS 1632 - DELIVERABLE 4:

Performance Testing

David Stropkey

Github username: dms80790

Github url: <https://github.com/dms80790/D4>

**Summary**

The most challenging part of this assignment was figuring out how to optimize the program. While it was obvious that the source of time consumption was mostly due to the large calculations taking place in the calc\_hash() function by looking at the Flamegraph, what wasn’t initially apparent was how to go about decreasing the amount of time this function was running the cpu. The other challenging part was trying to think of all the corner cases that need covered to prevent the program from crashing or giving a false result.

I considered many edge cases for this program. One example is i having invalid characters in different sections of the blockchain, such as non-numeric characters as account numbers, non-numeric characters in the block number, and non-alphanumeric values in block hashes. I also considered incorrect length values for block hashes, previous block hashes, account numbers, and number or accounts partaking in a single transaction. There are many other edge cases that I also considered, such as a non-existing file argument, bad timestamps, and incorrectly calculated hash values.

In order to optimize my program, I figured out that we didn’t actually have to calculate the hash values at all in our verification if we already knew them. So, I wrote a quick program that calculates the hash values of all characters a-z, A-Z, 0-9, ., :, >, (, ), and |, as these are the possible characters in our blockchain. I had the program write these values to a file, and then read the values into my verifier.rb file and put them in a hash map, which was passed to my verifier\_checker.rb file. This way, any time a character value needed to be known, it could simply be accessed via the hash map in O(1) time.

**Program Run Times**

*Unoptimized Run Times:*

Run 1: 51.731648 s

Run 2: 52.0888853 s

Run 3: 51.4239571 s

Mean: 51.7481634667 s

Median: 51.731648 s

*Optimized Run Times*

Run 1: 2.6232998 s

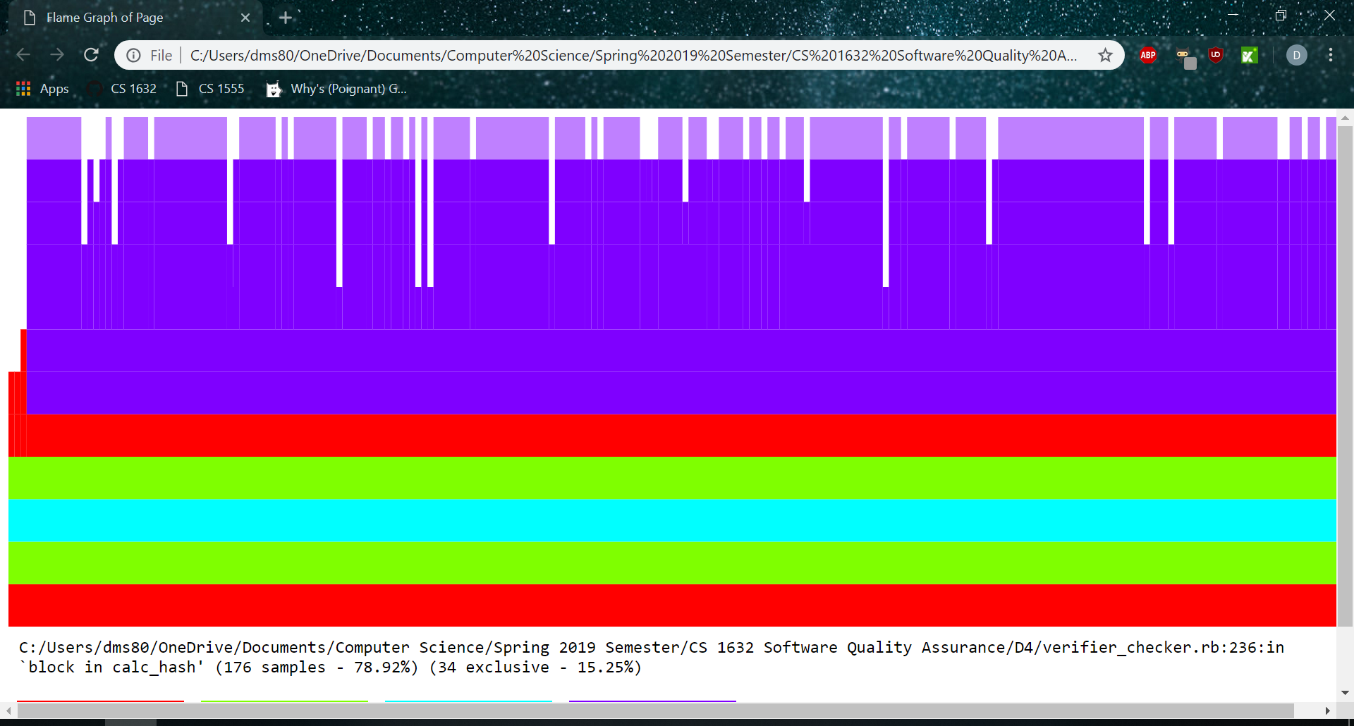
Run 2: 2.5839095 s

Run 3: 2.5821102 s

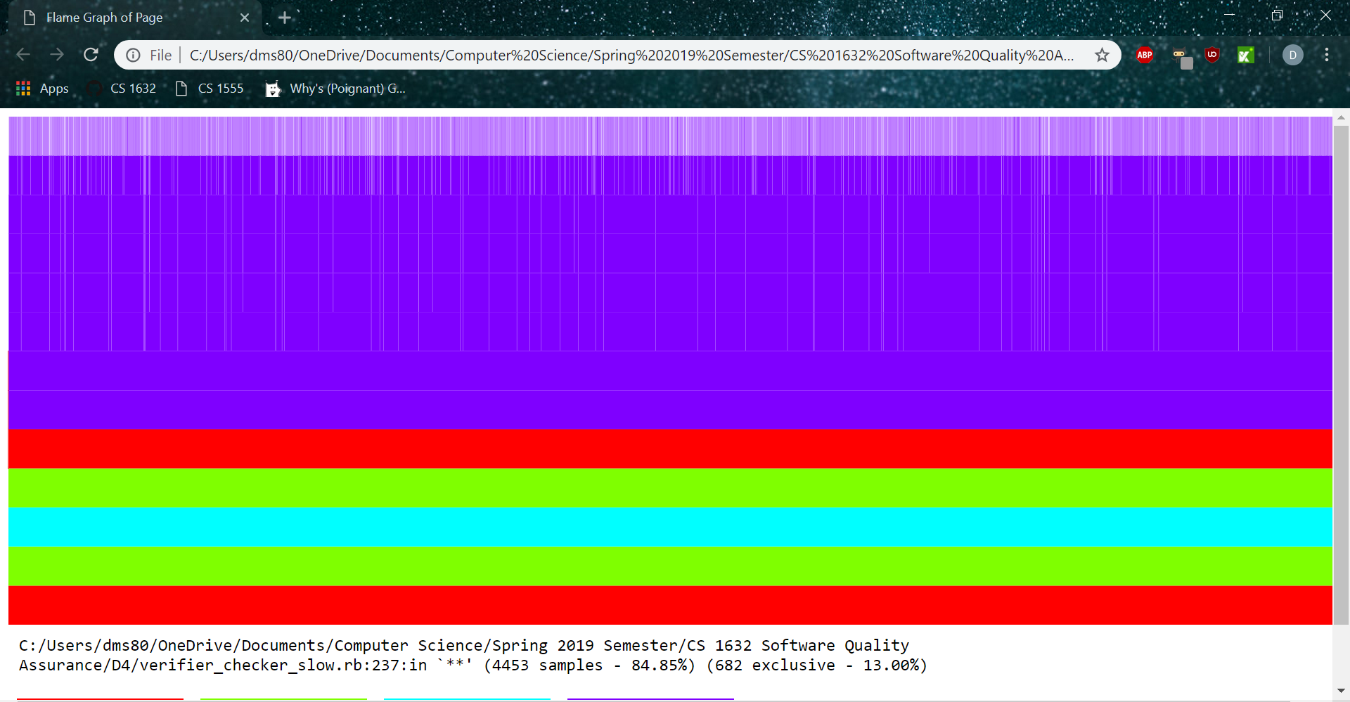
Mean: 2.59643983333 s

Median: 2.5839095 s

**Flamegraphs**



Optimized Flamegraph



Unoptimized Flamegraph.