Austin Linder (all141) & Jacqueline Marx (JAMarx39)

<https://github.com/all141/Deliverable_3>

CS 1632 - DELIVERABLE 3: Performance Testing

The most difficult part of this assignment was unit testing. Many of our methods both inside the block class and in the main verifier program relied on other methods, and were rather long and complicated. The process of actually creating the functionality of the main program was not too difficult, as the hashing function was provided to us, and it was mostly a matter of learning to utilize the split command in order to get the data we needed from the blockchain organized. Getting the Flamegraph going was also not too difficult, it just took some time to understand what it was exactly, and we had to Google some things like fast\_stack and stackprof.

In our unit tests, we checked for several edge cases. For example, in ArgumentCheck\_test we checked for three edge cases in the number of arguments. The assignment required the program to only accept one argument, so we tested cases of 0,1, and 2 arguments. 0 and 2 were also failure cases.

After observing the Flamegraph and doing some analysis, it became clear what parts of the program were taking the most time. The Flamegraph was by far taking the most amount of time to create, as evidenced by the lowest and thickest bars on the graph. Next up was the verify\_hash method from verifier. This method takes a significant amount of time because it calls methods within it (such as compare\_current\_hash mentioned below), and many of those methods call different methods. However this can vary depending on the blockchain you are reading, because its behavior is conditional, and may run shorter methods when given different input. The next most demanding method was compare\_current\_hash from block.rb which verifies whether a block has the correct hash. This method calls create\_hash, another method, which itself is the next most demanding method. Create\_hash takes up a decent amount of attention because it has to iterate across an entire block with the map! function, and does math with large values each time.

To decrease the execution time, we disabled the flamegraph functionality to shave off about 4 seconds. Originally long.txt was running for approximately 31 seconds. After removing the functionality, the time was reduced to the results on the next page.

The results of the three tests run with long.txt are as follows:

1: 28.3903892

2: 28.5011082

3: 28.3005088

Mean: 28.3973354

Median: 28.3903892



