CS 1632 - DELIVERABLE 4: Performance Testing

Group Member: Zikai Zhang, Jie Zhou

GitHub Username: zzklachlan, jayzhou125

Repo Link: <https://github.com/zzklachlan/D4>

**Summary:**

For D4, we are asked to write a program to verify if a blockchain is valid. And the verify process should be as quick as possible. As far as practical experience concerned, we worry the optimization part of the assignment in the first place.

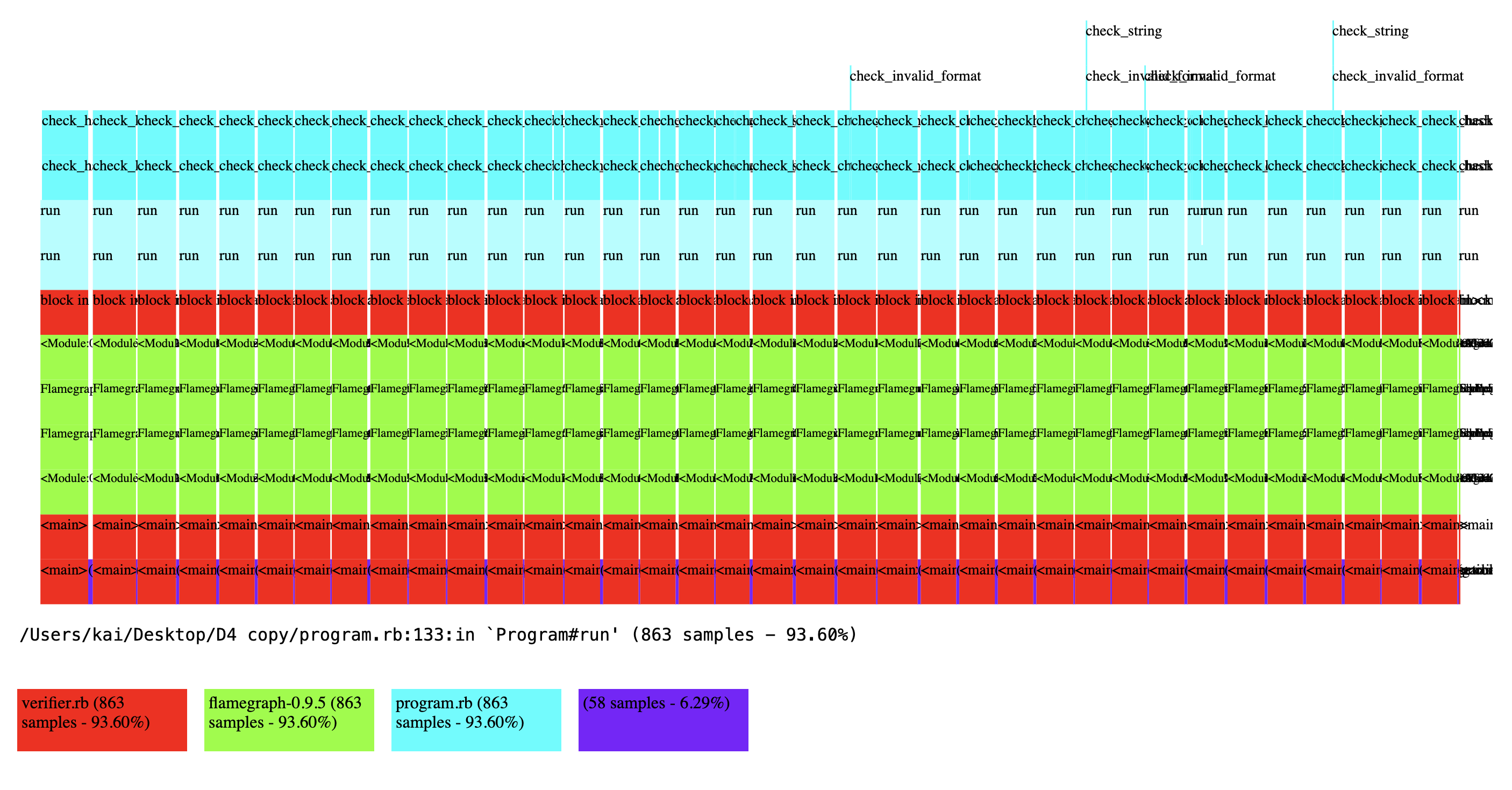
However, the most challenging part actually comes from the Pre-testing preparation. When first trying to get the program work and running, we were to match up the program’s output with the sample-output that was given. So we would write up a check for certain function and output the corresponding output and exit. We know that we should exit the program only once for unit test purpose and we figure it is easy to adjust the exit point later on when we got the program working. Turned out that we spend lot of time trying to refactor the code to exit at a single point of the program. That was the design flaw that we faced. We should have just write the program in a way that is easier for testing.

Edge cases and failure that we thought about are: when user passed in more than one argument, when a single block from a blockchain has wrong number of pipes, wrong pairs of parentheses, address that contain both numeric and string like “4567s9”.

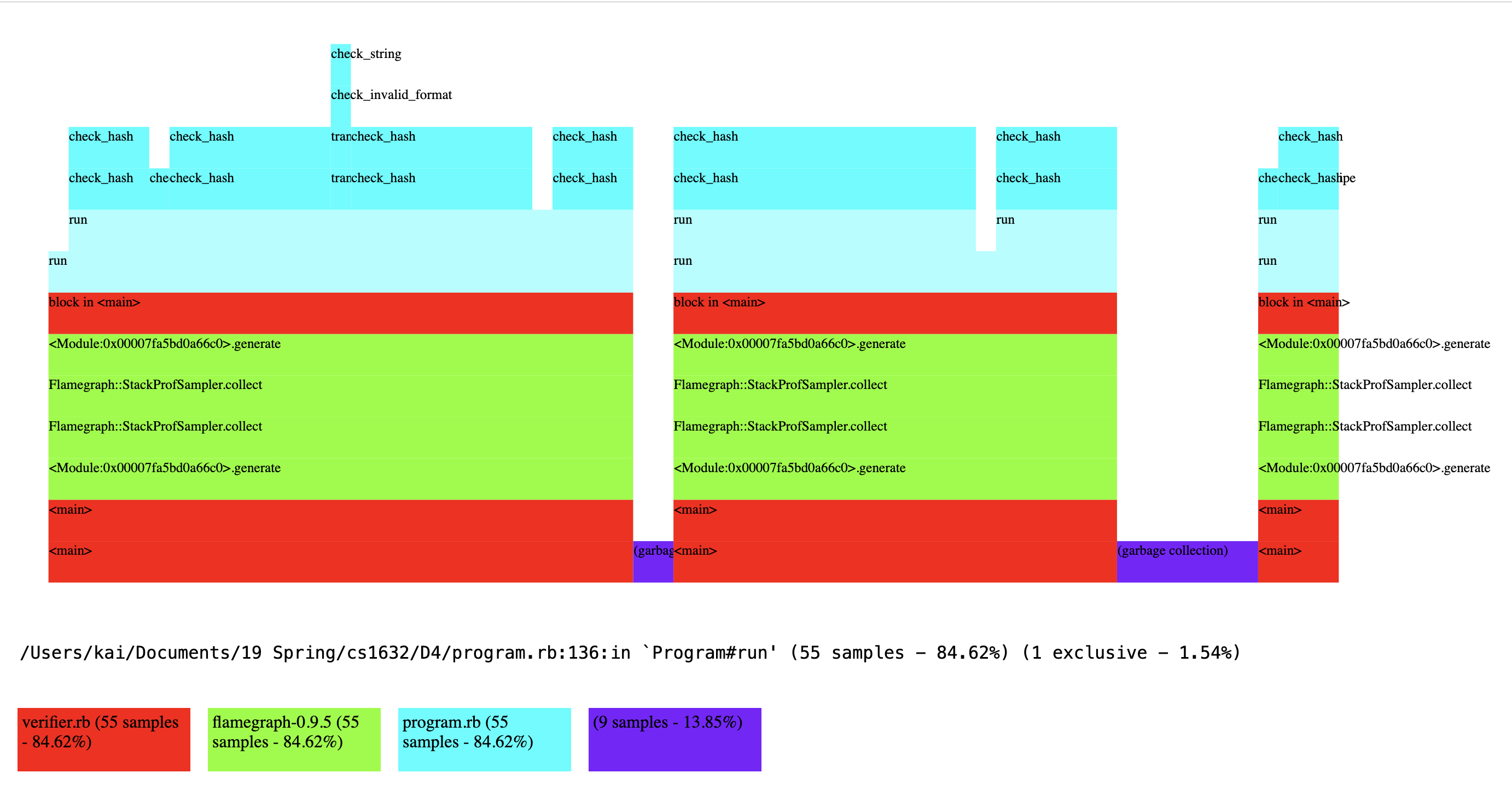
Using flame graph, the **check\_hash** method was taking most CPU run time. The method checks if the blockchain is hash value is correct. It is expected base of what the function does is basically recalculate the entire hashing process. What we did is using cashe trying to memorize a certain hash value during the calculation. That way, though at first the method would be slow, but later on when dealing with larger and larger input, the process time would be much faster.

**Flamegraph:**

Before optimization:

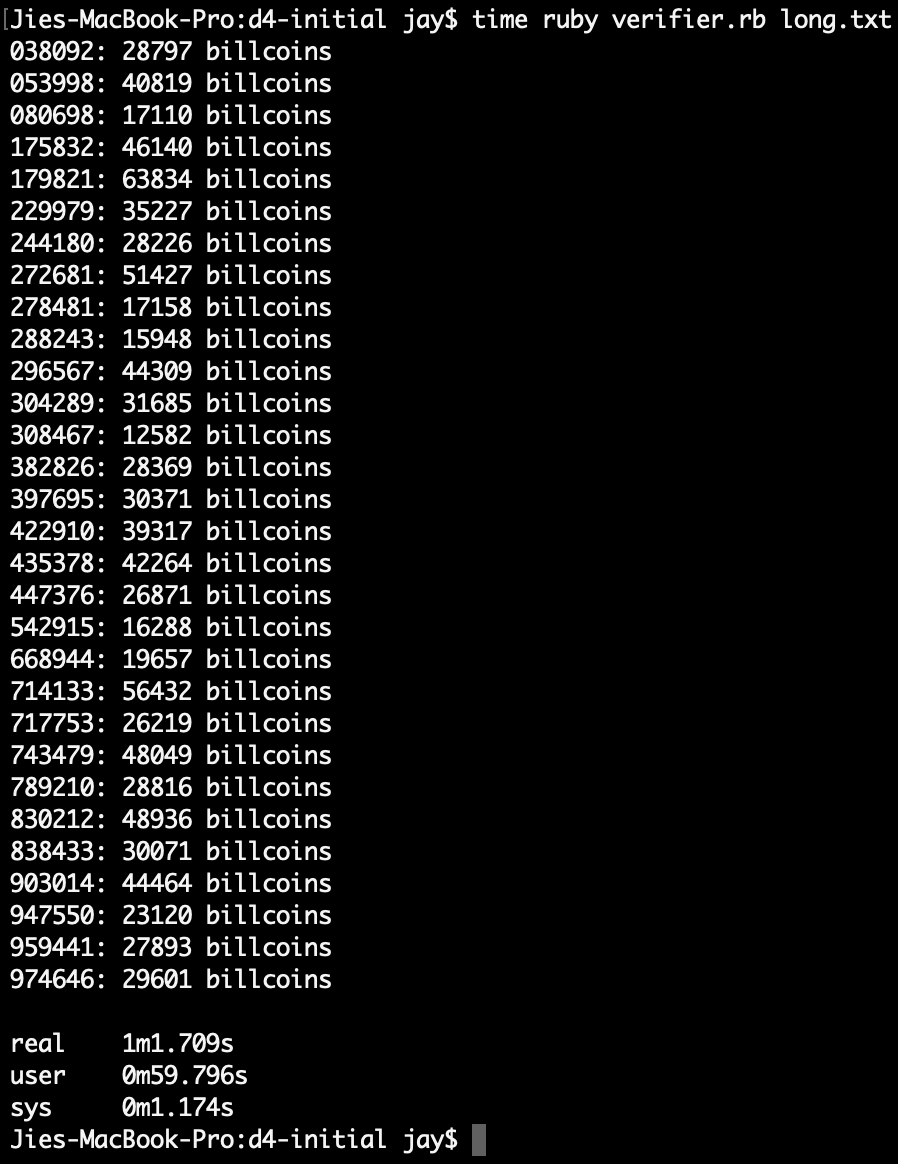
****

After optimization:



**Time:**

Initial:



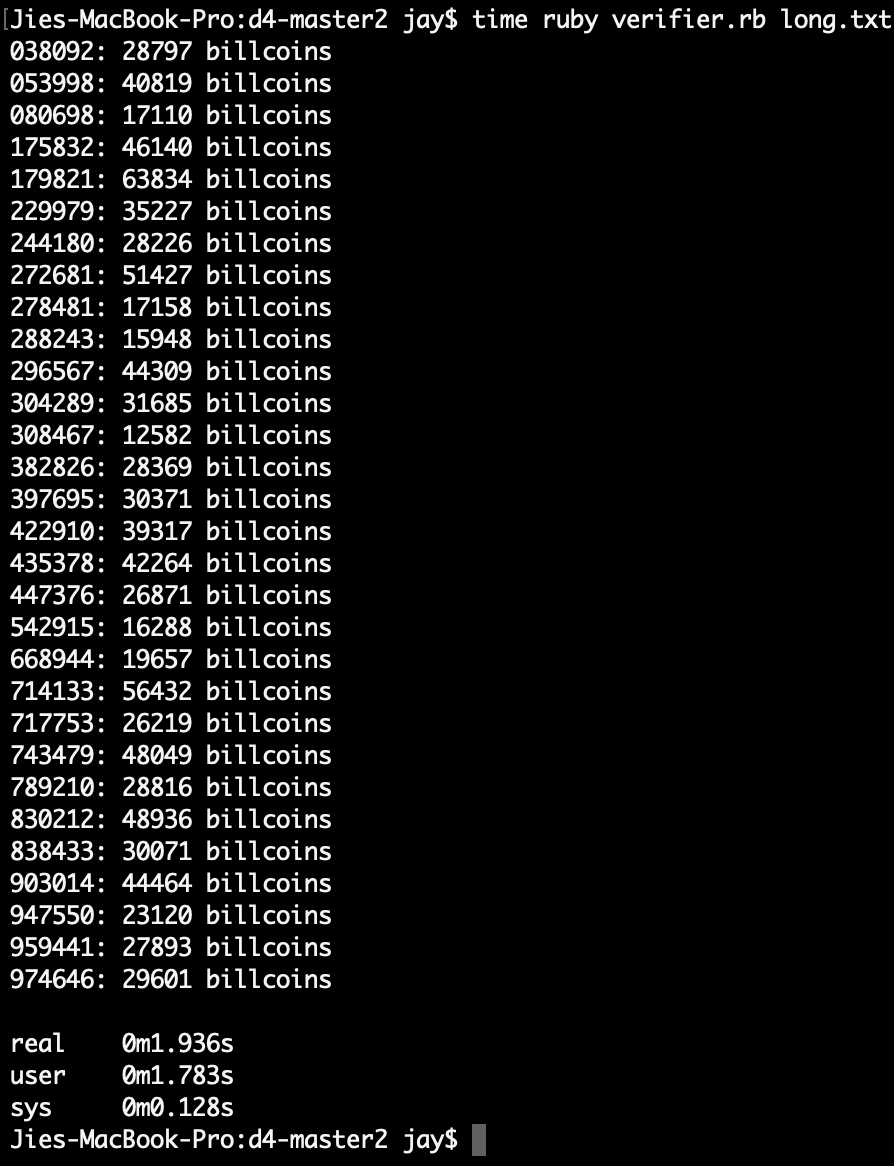
Mean time: (61.709 + 65.944 + 62.681)/3 = 63.445s

Median: 62.681s

Max: 65.944s

Min: 61.709s

Optimized:



Mean time: (1.936 + 1.950 + 1.949) = 1.945s

Median: 1.949s

Max: 1.950s

Min: 1.936s