CS 1632 - DELIVERABLE 4: Performance Testing

Repo URL: https://github.com/MisterZW/D4

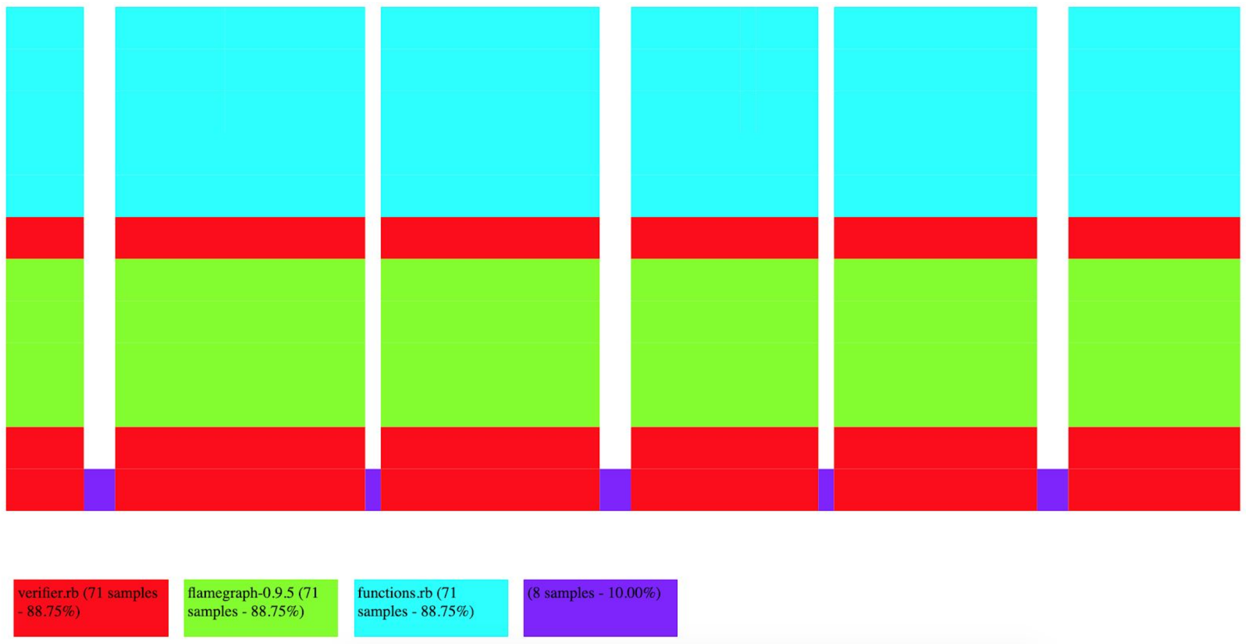
Zachary Whitney, zdw9, MisterZW
Michael Schropp, mfs35, mfschropp

The two largest challenges we ran into during the deliverable dealt with the requirements and the tools. Starting out, neither of our group members had experience with blockchain. While the requirements were fairly straightforward, there was a lot of information to unpack. To get past this, we worked through the requirements together and did most of our error checking while we were together in person. The actual parsing and checking of the blockchain was relatively simple when following the requirements, but the error checking took longer to finish. The other issue we ran into was using the flamegraph tool. We found that it worked fine for sample.txt for the unoptimized version, but our optimized version ran sample.txt too quickly to get a good flamegraph formed. To address this, we used two different files for our flamegraph images--one for sample.txt and one for 1000.txt.

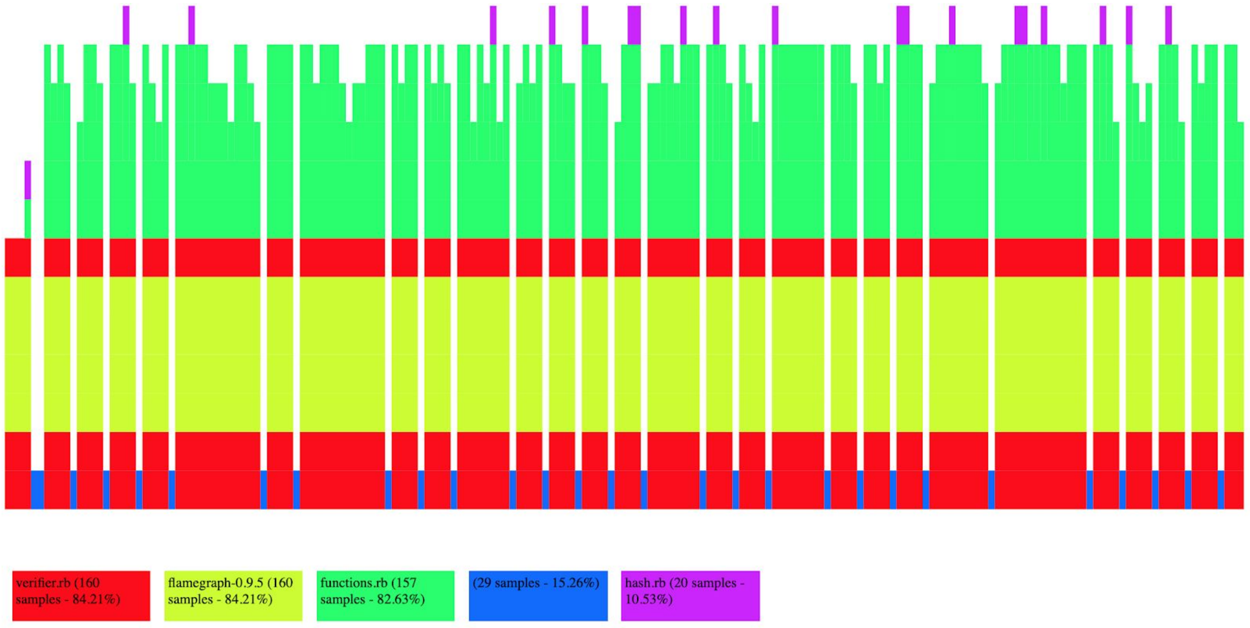
As mentioned previously, the error checking was the portion of the project that took us the longest. There were a lot of locations where something could be wrong in the format of the blockchain. Most of the error checking related to the format of the file. Since we wanted to isolate errors specific to the blockchain formatting in our error handling, we created our own error and error message. Some other error checking included checking the hash was correct and checking for non-negative balances at the end of transactions.

With this, we had a pretty slow initial program, running at an average of 50.083 seconds. Using flamegraph, we noticed that the largest time was being taken up calculating the hash for each character, specifically in our calculate_hash function. Our first thought to reduce the run time was dynamic programming. This would let us store the values we have hit before so we did not have to calculate them. Instead, we opted to calculate each hash value ahead of time, before program execution, as shown in our hash.rb file. Instead of calculating a hash we would now just do a lookup for that value in the Ruby Hash. The result of this lowered our average runtime down to 1.078 seconds. This was the largest decrease in runtime, but other small changes led us to the average runtime shown below (e.g., disabling the garbage collector).

Unoptimized Flamegraph (on sample.txt):



Optimized Flamegraph (on 1000.txt):



Unoptimized Times for long.txt *(in seconds)*

Run 1: 49.941

Run 2: 49.379

Run 3: 50.929

Mean: 50.083

Median: 49.941

Optimized Times for long.txt *(in seconds)*

Run 1: 0.606

Run 2: 0.602

Run 3: 0.610

Mean: 0.606

Median: 0.606