CS 1632 – DELIVERABLE 4: Performance Testing

Sean Mizerski and Ronen Orland

GitHub Usernames: stm107 and orlandronen1

GitHub URL: <https://github.com/orlandronen1/D4.git>

There were a few challenging aspects to this deliverable. First, Sean specifically didn’t know that much about Blockchain, so he had to study up on the concepts introduced by this deliverable in order to be successful at understanding the implementation. Luckily, the billcoin md file had a good basic overview and was helpful. Another challenge came in deciding the kinds of data structures to use when manipulating all of the data from the files. Everything can be done with string manipulation, but its repetitive and costly. Then we thought we would use arrays for storage, but lookups were frequent and also costly. We finally decided on hashes (like java hash maps) for faster look ups. Our initial run on long.txt ran at a comparable speed to the initial run in the sample\_output.txt file so that seemed ok.

Ronen started by opening our repo and listing a series of corner and edges cases, along with the given constraints for the block data. We considered not just the basic format of the file, but the lengths of the data, the characters that each should be, the order that they should be in, the number of elements that should exist, and if any of those elements can be or cannot be null. When handling our errors, we incorporated several error handlers that go beyond the stated error handlers given in sample\_output.txt.

The method that took the most time was related to verifying the hash. Our initial implementation tried to remedy a bit of the pain associated with this method by using hashes instead of lists and to try not to repeat string manipulations. We then attempted to use multithreading to run somewhat costly processes simultaneously to cut down on runtime, but this ended up being tricky to implement in Ruby. We then switched to storing the calculated hash value of each character so that we didn’t need to re-calculate hash values that we have already calculated. This change presented the greatest time improvement. Finally, instead of storing the whole hash value, we stored the modulo value to make it smaller.

Initial:



Execution times:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time in seconds | Run 1 | Run 2 | Run 3 | Mean | Median |
| Initial | 48.887 | 46.234 | 46.408 | 47.176 | 46.408 |
| Optimized |  |  |  |  |  |