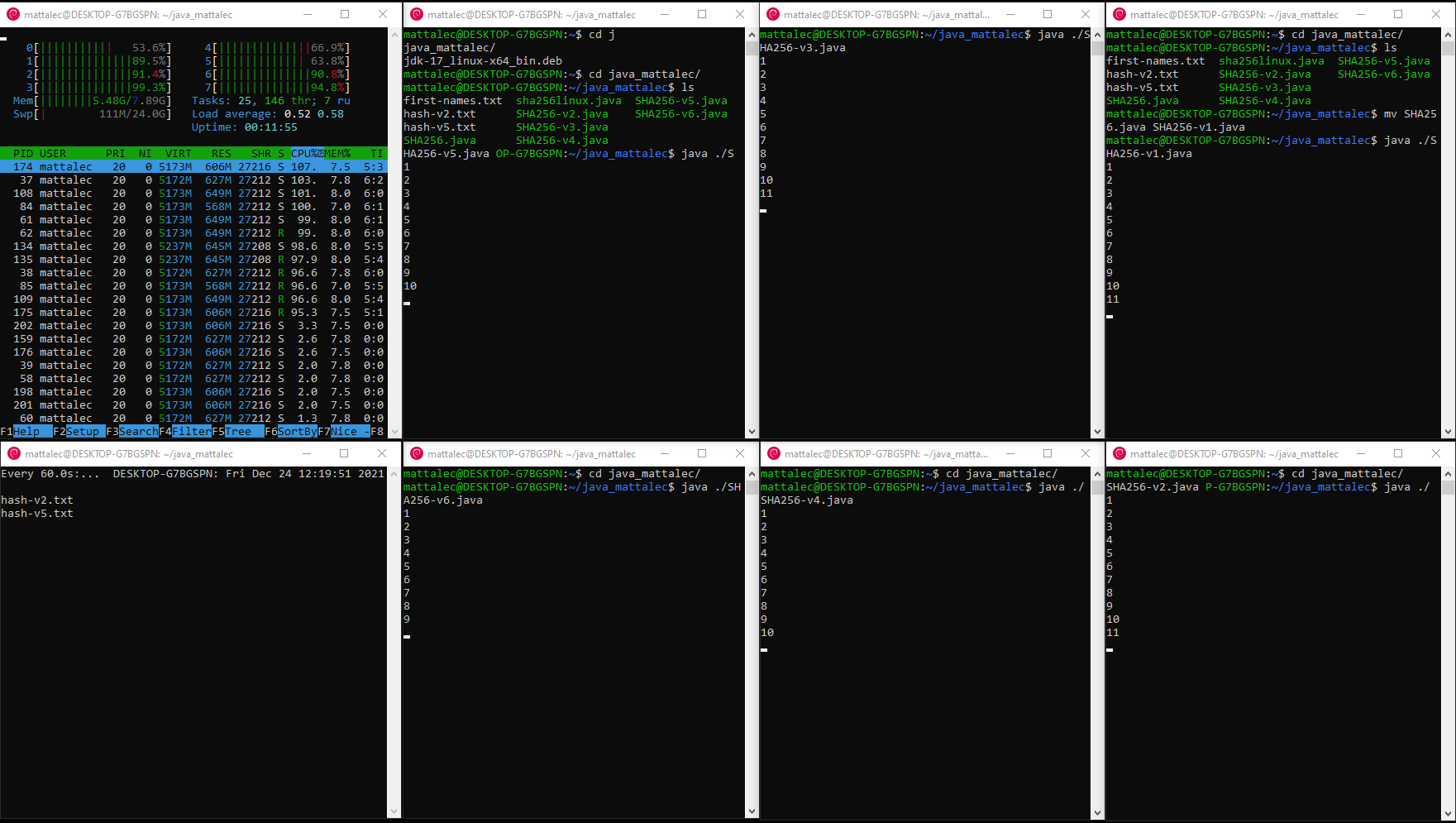
<https://www.ssa.gov/oact/babynames/names.zip>

Matthew Kenny 20384361 CS210

Overview

In this project my objective was to find two Strings that had similar sha256 64-bit hash values. When I received this project, I read the instructions and then started working with the code that I was provided. My first idea was to use the dictionary provided and compare two randomly chosen strings. While running this for a couple of minutes I found two hashes that had 19 compares. Then I tried to loop through the whole dictionary file and compare every word against every other word. This was the idea I thought would work best but would take a long time to run through the whole text document of words. Then the next thing I had to think about was how I was going to create my valid strings. The best idea I produced was to have a text file of people's names and either randomly put them in to a valid string or loop through every possible name in a string against every other name in a string. I got a zip file from ssa web site that had all the names of babies from before the 1900’s all the way up to today. This gave me a big dictionary to pull from. The zip file had multiple text files, so I had to concatenate them together. I did this by using Debian and ran **Cat yob\* > wordsname.txt**. There was still unneeded text and a lot of duplicate names in the text file. So, I ran this line in the terminal to only have one accordance of each name in the file. **(cut -d"," -f1 wordsname.txt | sort | uniq > first-names.txt)** If you compare every word in the file to every other word there is a total of the word count of the file to the power of two combinations. In the text file I had there was a total of 100,000 names so this means there was a total of 10 billion combinations. So, I decided that I would put two names in each sentence. This would increase the total combinations to word count in the file to the power of 4. So, I implemented this idea. Compare Strings that include two random names to each other and ran it during the day on the 25th and the 26th of December. The best comparison I could get was 22. I ran the code in six separate Debian terminals to try and maximize my search.

 I kept running it, but I could not seem to get higher than 22. Then I thought of a way of making it more efficient. I did this by making a million hashes that are read into an array read into array. By doing this I will not have to run the sha method or the dictionary method again. This will save a lot of time. One disadvantage is that the array takes up a lot of memory and I only have 8gb of ram and only 6gb of ram able to be used. I thought to try and make two arrays, but it was too taxing on the memory. The best idea I could produce is two randomly generated names included in a string compared to the 1 million hashes in the array. The possible combinations for this 1\*10^13. This is useful because it could be run for days on end. I ran it for a 24-hour period using this code in 6 different terminals and got a lot of 22 compares and a dozen 23 compares.

**Code structure**

The algorithm I used throughout the project was a linear algorithm. It seemed like the best and most efficient algorithm to use in this project. Saying that there is no pattern to the sha256 values I could only think to use that sort of algorithm. When the user runs the code the hash builder method is ran. This method fills an array with a million hashes. Each String that is converted into a hash has a name and a number. For example, “Josh is five years old” At the top of the code before the main method I created two arrays, one for the hashes and one for the sentences. When a hash is read into the hash array, the sentence that created sha265 number is read into the sentence array. Every time the code is run the same array is created. These arrays are public and can be used anywhere in the code. Once the hash builder method has been ran, the code goes back up to the main method and runs the hash compare method. This method will create another sha256 value and compare it to all the values in the array that was created the hash builder method. The string is created by adding to randomly generated Strings to another String. For example, “My friends are joe and Frank.” In the text file I am using there are just over 100,000 names in the file. This means there are just over 10 billion different combinations of this String. I used a double loop to compare these strings to the whole hash array. In the second for loop, we go into another method called compare which compares the hashed character by character. In this method I have an if statement that check If the compare is over a certain number. If it is over, it will read the two sentences, there two hashes and the number of comparisons into a file.

Original I had written a different code structure. In this original one I had a one main method that compared two strings that are randomly generated. This method was not as efficient. This is because I had to create new strings every time it looped. With the method I have now creates one random string and compares them to a whole array of strings. I ran this method for 8 hours and got two 23’s and a lot of 22’s.

**Learning outcomes**

I have learnt a lot from this module. I found that it was a more advanced version of the first-year modules cs161 and cs162. In those modules you learn the basics of programming java. I found in this module that I learnt skills that will help me in any project or program I write in the future. I think the main thing I learnt is planning out what you want the program to do is very important. Also, we were challenged a lot in this module to make programs that not just worked but were efficient. These things helped me a lot in this project and helped me find diverse ways and more efficient ways of finding two similar sha265 values.