Matthew Kenny 20384361 CS210

Overview

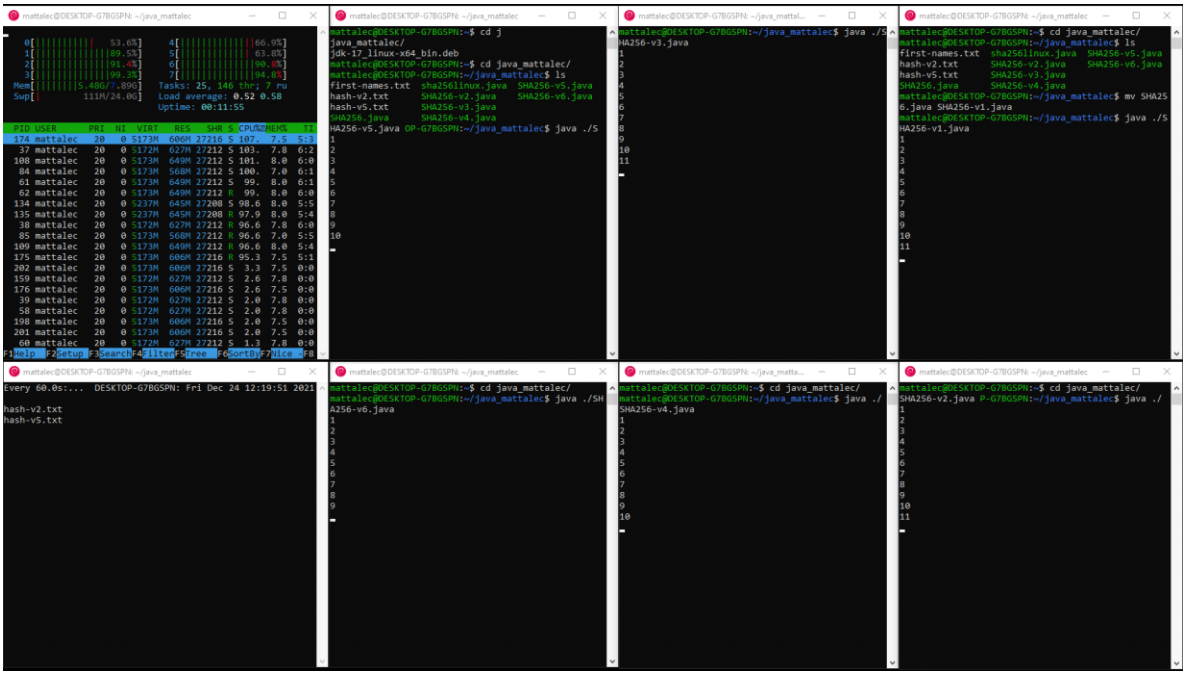
In this project my objective was to find two Strings that had similar SHA256 64-bit hash values. When I received this project, I read the instructions and started working with the code that I was provided. My first idea was to use the dictionary provided and compare two randomly chosen strings. While running this for a couple of minutes I found two hashes that had 19 matches. 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.

The best idea I produced was to have a text file of people's names and either randomly put them into a valid string or loop through every possible name in a string against every other name in a string. Given that there isn't enough valid capitalized words or names in the sample ***words.txt*** file, I decided to look for a better file to build valid sentence strings. At first I used a Scottish baby names file, but it only contained 32,000 names. Instead, I downloaded the list of baby names (**names.zip)** from the US social security web-site [1] containing names of babies from before 1900 all the way up to today. This gave me a big dictionary to pull from. The zip file had multiple text files (yob-*.*), so I had to concatenate them together. I accompanished this using a simple bash command in Debian as follows: ***cat yob* > wordsname.txt***

There was still unneeded text and a lot of duplicate names in the text file. So, I ran this bash command in the terminal to leave only one accordance of each name in the output file:

(cut -d"," -f1 wordsname.txt | sort | uniq > first-names.txt)

If you compare every word O(n) in the file to every other word the total word count of the file is $O(n^2)$. The data file ***first-name.txt*** contains a total of 100,000 names (10^5) so this means there was a total of 10 billion combinations (10^{10}) if the names are placed into valid sentences. 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 (10^{20}). So, I implemented this idea. I compared valid sentences including 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 and use most of the 8 CPUs in my Desktop machine.



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. By doing this I did 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 with 6Gb of usable RAM. One idea I had was to generate two arrays, but it was too taxing on the memory. The best idea I could produce was to create a non-termination while loop that creates a sentence containing two randomly generated names (10^{10} combinations) and compare that to 1 million (10^6) precomputed SHAs created by combining 10 ages (as words) and the 100000 first names. This results in 10 quadrillion combinations (10^{16}). This is useful because it could be run for days on end without exiting. 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. The code is over 20 times faster than creating two sentences and two corresponding SHAs (using no array structures).

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 (described above). Each string is converted into a hash containing one unique name and one of ten numbers. For example, “Josh is five years old”. The classes begins by creating two publically accessible arrays, one for the hashes and one for the sentences. When a hash is read into the hash array, the

sentence that created SHA256 number is read into the sentence array with the same matching index. Every time the code is run the same array is created. The overhead for this hash table building is only a few seconds. Because these arrays are public, they can be used anywhere in the code. Once the hash builder method has been run, 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 two randomly generated first names to a string. For example, “My friends are *Joe* and *Frank*.” 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 found two 23’s and approximately 11 time as many 22’s for every 23 found.

Running the code for two days I have in excess of twenty 23s but didn’t find any 24s. Based on the comparison of 22s to 23s I would have expected to have found two 24s, but there is the possibly of misses because of the randomness of the problem. After running it for a few more hours I didn't find any 24s but i did find one 25.

Best Matches

```
Lyelah and Longino are friends.
Manali is four years old.
c326a8c6f65df79a38fd3cde8babaa5b4a0024dadb92a3b24db0d351a00bb22e
c22848d6900bf70a092d77d84ba8816bdb6024deeb4e03b24d0324464e3bb2ea
25

Evaine and Ziv are friends.
Mayrim is four years old.
```

1d280a014db5644cd14c3fe74158d782c459f0792634ff5b12a203b1000804ef
34680a009db054d4f1bbddd5cf58a7882f0a90793e3cf29bc2ef03df705fc442
23

Eugenio and Bedford are friends.
Matika is three years old.
c7ed034dcb29aa44e1db8c58f58bbebd0f24546b105656bfaab4b756a1156110
909df3f5b38954d6ef8b8e59f5dfbe1b5ed66299148659bfa972135fe15a6110
23

Zamauria and Everton are friends.
Berl is seven years old.
2b3d85fc804828472c1fae15e67bbd4306bd23edc11356c610760de9603e18f9
bbefea8b8048a2e8dc534512e3721c53b2be23fdc75d5cccb0ca6dfe2f8318f9
23

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 SHA256 values.

References

[1] US SSA first names, “baby names from social security card applications national data”, https://www.ssa.gov/oact/babynames/names.zip, Dec 2022.

Appendix

```
import java.util.*;

import java.security.*;

import java.io.*;




public class Cs210Project {

public static String [] shaAry ;

public static String [] sentAry;

public static void main (String[] args) throws IOException{

hashbuilder(); //Runs the hash builder method that builds a array with 1 million sha265 values.

long startTime = System.currentTimeMillis(); //start time

mainCode();

long estimatedTime = System.currentTimeMillis() - startTime; //start time-end time

System.out.println(estimatedTime/1000.00 + " seconds"); //prints time spent running

}



public static String sha256(String input){

try{

MessageDigest mDigest = MessageDigest.getInstance("SHA-256");

byte[] salt = "CS210+".getBytes("UTF-8");

mDigest.update(salt);

byte[] data = mDigest.digest(input.getBytes("UTF-8"));

StringBuffer sb = new StringBuffer();
```

```
for (int i=0;i<data.length;i++){
sb.append(Integer.toString((data[i]&0xff)+0x100,16).substring(1));
}
return sb.toString();
}catch(Exception e){
return(e.toString());
}
}
```

```
public static void compare(String s1,String s2,String x1,String x2) throws IOException
{
int count=0;
/*This loop checks each position in each String and checks if they are the same. If they are the same count
goes up by one. I have placed a second if statement to try and speed up the loop. It will break out of the
if count isnt high enough when it gets to an i value that can be set depending on what compare the user is
going for.*/
for(int i=0;i<x1.length();i++)
{
if(x1.charAt(i)==x2.charAt(i))
{
count++;
}
if(i>=54 && count < 6)
{
break;
}
```

```
}
/*This if statement is uses to print the best compares to the screen and to a
file so they dont't get lost.
I have comment out the file writer*/
if(count >= 19 && !s1.equals(s2))
{
//BufferedWriter writer = new BufferedWriter(new FileWriter("./hash-v1.txt",
true));
System.out.println(s1 + "\n" + s2 +"\n" + x1 + "\n" + x2 +"\n"+ count +"\n\n" );
//writer.write(s1 + "\n" + s2 +"\n" + x1 + "\n" + x2 +"\n"+ count +"\n\n" );
//writer.close();
}


}



public static void mainCode() throws IOException
{
Dictionary dict = new Dictionary();
Random rand = new Random();
String sentence1="";
String sentence2="";
String shaOne="";


int i=0;


/* This double for loop compares a randomly created string to the whole array
that is created in the hashbuilder
```

```
method. The compare is done in a different method called comapare() */


while(true)
{
if(i%100==0)
{
if(i==0)
{
System.out.println("Start of code...");
}
else if(i<1000)
{
System.out.println(i + " million compares ran.");
}
else if (i>=1000)
{
double num = i/1000.0;
System.out.println(num + " billion compares ran.");
}
}
sentence1 = dict.getWord(rand.nextInt(dict.getSize())) + " and " +
dict.getWord(rand.nextInt(dict.getSize())) + " are friends.";
shaOne = sha256(sentence1);
for(int j =0;j<dict.getSize()*10;j++) {
compare(sentence1, sentAry[j], shaOne, shaAry[j]);
}
i++;
}
}
```

```
public static void hashbuilder()

{

Dictionary dict = new Dictionary();

sentAry = new String[dict.getSize()*10];

shaAry = new String[dict.getSize()*10];

String                                  []ages                                  =
{"one","two","three","four","five","six","seven","eight","nine","ten"};

int size = dict.getSize();

int k =0;

/*This method creates an array of just over 1 million sha265 values. This is done
with the double for loop

below.*/



for(int j=0;j<ages.length;j++)

{

for(int i=0;i<size;i++)

{

sentAry[k] = dict.getWord(i) + " is " + ages[j] + " years old.";

shaAry[k] = sha256(sentAry[k]);

k++;

}

}



}



}
```

```
class Dictionary{
private String input[];


public Dictionary(){
input = load("./first-names.txt");
}
public int getSize(){
return input.length;
}
public String getWord(int n){
return input[n];
}
private String[] load(String file) {
File aFile = new File(file);
StringBuffer contents = new StringBuffer();
BufferedReader input = null;
try {
input = new BufferedReader( new FileReader(aFile) );
String line = null;
int i = 0;
while (( line = input.readLine()) != null){
contents.append(line);
i++;
contents.append(System.getProperty("line.separator"));
}
}catch (FileNotFoundException ex){
```

```
System.out.println("Can't find the file - are you sure the file is in this location: "+file);
ex.printStackTrace();
}catch (IOException ex){
System.out.println("Input output exception while processing file");
ex.printStackTrace();
}finally{
try {
if (input!= null) {
input.close();
}
}catch (IOException ex){
System.out.println("Input output exception while processing file");
ex.printStackTrace();
}
}
String[] array = contents.toString().split("\n");
/*for(String s: array){
s.trim();
}*/
for (int i=0; i<array.length; i++)
{
array[i] = array[i].trim();
}


return array;
}
}
```