David Morales

Lab 1 – Option B

CS2302 Data Structures

September 6th, 2019

Introduction:

In this lab, the problem was to brute force hashed passwords from the textfile provided to us. The constraints of the lab were to use recursive methods and *at most* two nested for loops. The hashed passwords were between 3 to 7 characters long, and were digits only. A method that creates hashes was provided to us.

Proposed Solution Design and Implementations:

My main outline that I drew up was to first extract the information from the textfile into a 2D string array (lists in Python), and use a loop to feed the information to a method that will do generate the passwords. After that, I was to write the recursive method that will generate the passwords and makes the comparisons to the hashed passwords in the textfile. A for-loop within the recursive method was to be used to generate the numbers and make the comparisons. Then use recursion to add another character to the brute force if it wasn’t found in the certain number of characters.

The actual implementation began when I created a method to takes in a salt name (as a string), a hashed password (as a string), and the number of characters (as an integer) to use the brute forcing.

A for-loop was used to generate all the numbers within that number of characters. If the password was not found, the method will call itself again, and add one to the character number to go into the next set of numbers. If the password is found, then it returns the cracked password as a string.

I used a separate recursive method to get the highest number within a certain numbers of characters (Ex: 99, 999, 9999). This method returns a string that will later be in converted in the password generator method.

In the main method, a for-loop is used to insert the information from the 2D string array extracted from the textfile into the password generator method. It goes through line by line of data (where each user information is stored). It prints out the user and the password if the program finds the password. If the program is unable to find the password (password not all digits, incomplete hash, or bad input, etc), it will display an error that the password was not found.

Experimental Results:

The program behaved and ran as expected. Not much changes were made from the initial implementation since there is very little to do to speed up and reduce time complexity of brute forcing hashed passwords with the current design and constraints. The program runs reasonably well (I ran this on an older laptop). The runtime ranged between 6-8 minutes on my machine using the password list (100 users) provided to us.

All passwords from the textfile was cracked successfully with my algorithm. I ran separate tests (using the hash method with the cracked passwords and salts, and compared manually to the hashes from the textfile) to see the passwords were the actual passwords, and everything seems to be accurate and true.

Here is the cracked password list generated from my program:

User0 password is: 2419445

User1 password is: 6682

User2 password is: 949228

User3 password is: 6140

User4 password is: 942

User5 password is: 2671660

User6 password is: 3875

User7 password is: 39666

User8 password is: 7041617

User9 password is: 536596

User10 password is: 95300

User11 password is: 0254985

User12 password is: 9963

User13 password is: 80104

User14 password is: 87955

User15 password is: 960

User16 password is: 55409

User17 password is: 0008

User18 password is: 5094

User19 password is: 4202784

User20 password is: 1071180

User21 password is: 113

User22 password is: 133488

User23 password is: 079312

User24 password is: 965847

User25 password is: 492

User26 password is: 367

User27 password is: 144

User28 password is: 882676

User29 password is: 027037

User30 password is: 79612

User31 password is: 496

User32 password is: 7846812

User33 password is: 66477

User34 password is: 073282

User35 password is: 8510

User36 password is: 1881145

User37 password is: 9562452

User38 password is: 294

User39 password is: 1498538

User40 password is: 674

User41 password is: 3682

User42 password is: 42273

User43 password is: 185

User44 password is: 104034

User45 password is: 904

User46 password is: 04288

User47 password is: 1130

User48 password is: 65708

User49 password is: 601

User50 password is: 4264

User51 password is: 2043256

User52 password is: 548

User53 password is: 0750

User54 password is: 10649

User55 password is: 2466480

User56 password is: 4290

User57 password is: 150629

User58 password is: 8707532

User59 password is: 4770

User60 password is: 1720

User61 password is: 1440267

User62 password is: 2442002

User63 password is: 441

User64 password is: 118709

User65 password is: 9861315

User66 password is: 6319171

User67 password is: 9611

User68 password is: 4646

User69 password is: 725259

User70 password is: 707289

User71 password is: 017

User72 password is: 7804

User73 password is: 758866

User74 password is: 8287736

User75 password is: 34995

User76 password is: 62381

User77 password is: 7699450

User78 password is: 632

User79 password is: 9038

User80 password is: 2440

User81 password is: 8174401

User82 password is: 3118952

User83 password is: 984802

User84 password is: 853

User85 password is: 003757

User86 password is: 56660

User87 password is: 0259384

User88 password is: 497213

User89 password is: 7017

User90 password is: 7579036

User91 password is: 3244694

User92 password is: 562

User93 password is: 93499

User94 password is: 6675953

User95 password is: 9601

User96 password is: 12614

User97 password is: 7738

User98 password is: 1089

User99 password is: 378

Conclusions:

This project I’ve learned about recursive methods and algorithms, and how to implement them into the projects better. It was a good refresher on this concept, and gave me more a challenge instead of simple of recursive methods. I also learned a bit about hashes.

Appendix:

Source Code:

import hashlib

def hash\_with\_sha256(str):

hash\_object = hashlib.sha256(str.encode('utf-8'))

hex\_dig = hash\_object.hexdigest()

return hex\_dig

def max\_num(char\_num): #This method gets the max number of a certain amount of digits using recursion. Ex: 999, 9999, 99999...

if (char\_num == 0):

return ''

return '9' + max\_num(char\_num - 1)

def password\_gen(salt\_name, char\_num, hash\_pass):

max\_digit = int(max\_num(char\_num)) #Converts the string into a integer from the max\_num method.

if (char\_num >= 8):

print("Password not found. Not in the range of 3 to 7. This program only does digits within that range. Anything else is too much for me.")

return "Error. Password not found."

for x in range (0, max\_digit): #This for loop generates the numbers, adds them to the salt string, and uses the hashing method. Then compares string to the hashed password.

pass\_iter = str(x)

hex\_dig = hash\_with\_sha256(pass\_iter.zfill(char\_num) + salt\_name) #zfill is used to pad left side with zeros. Ex: 001, 00243, 02

if (hex\_dig == hash\_pass):

return pass\_iter.zfill(char\_num)

return password\_gen(salt\_name, char\_num + 1, hash\_pass) #This is the recursive part. Just adds one to char\_num to signify to add another character to the brute forcing.

def main():

print("Input textfile name: ")

filename = input()

with open(filename) as textFile:

textFile\_lines = [line.split(',') for line in textFile]

print("Brute force in progress! \n")

char\_num = 3

for x in range (len(textFile\_lines)): #This for loop feeds the password\_gen method with relevant information like the salt and hash from the textfile. Prints out password (if found).

hash\_pass = textFile\_lines[x][2]

hex\_dig = password\_gen(textFile\_lines[x][1], char\_num, hash\_pass.rstrip('\n'))

print(textFile\_lines[x][0] + ' password is: ' + hex\_dig)

main()