

## Lab 1 Report

Data Structures (CS2302)

Hugo Rico - 80531055

Instructor: Diego Aguirre

Sept. 19<sup>th</sup>, 2018

**Introduction:** The purpose of this lab was to use recursion to develop a brute force password generator in order to determine the real passwords of various system accounts given only each accounts' hashed password and salt value. Each real password was said to be composed of only integers (0-9) and could range from 3 to 7 characters in length. Yet, our password generator should be able to generate all possible passwords between any given set of lengths. We were allowed to use at most 2 nested loops inside of our recursive method, but less than two was perfectly fine as well.

**Design & Implementation:** The way I approached the problem was through the lens of a potential user. I began by focusing on the extraction of the data in the file containing the system accounts. In this module I used a series of 'while' loops, 'try' blocks and 'if' statements to properly receive user input regarding the name of the file and the minimum/maximum password lengths. The program handles erroneous input appropriately using the while loops and 'exception' blocks. Within this module, I also traverse the file line by line, splitting each piece of information per line and assigning them to an array called "fields". The program then calls the password generator method every time it reads a new line. I chose to solve the problem this way so that the passwords are calculated and printed on the screen in the order of the users in the list.

The next module involved in my program is the actual password generator, which receives 4 parameters: a saltValue, hashedPassword, minLength and maxLength. The method begins by generating the 'upperbound' for that respective method cycle ( $9 * \text{the minimum password length}$ ) and the first password to check ( $0 * \text{the minimum password length}$ ). I made sure to implement these generations as string multiplication and not integer multiplication. The method then proceeds with concatenating the initial password with the saltValue passed to the method in order to call the hash\_with\_sha256 method; this develops our first hashed password. The method then proceeds to compare the hashedPassword passed to the passGenerator method with the generated hashed password, and if a match is found, then the initial all-digit string is returned. Otherwise, the method proceeds into a while loop to generate the rest of the possible passwords by adding 1 to the password string of integers. An 'if' statement is used to correctly determine and format any leading 0's for certain passwords (i.e. 001 or 00013) using the '.zfill' function. Every time a new password string is generated, it is concatenated with the passed salt value to

generate a new hashed password. This hashed password is then compared to the hashed password passed to the method to determine a possible match. If a match is found, the method returns the original password string of integers. The recursive step takes place every time the 'upperbound' is reached by the generated password (i.e. 999 for the first cycle, 9999 for the second cycle, and so on) until reaching 9999999 (for password lengths of 3 and 7, respectively). When the recursive call is made, the only difference is the addition of 1 to the minLength value, meaning the next set of passwords to check are those with an additional character. The method runs until either a password match is found, or until reaching its base case. The base case would be the 'minLength' being greater than the 'maxLength'. If this base case is reached, the method prints out a statement informing a user that a password for a certain system account was not found.

The only other module involved in my program is the hash\_with\_sha256 method, which is used to generate the hashed passwords. This method was provided in the initial code at the beginning of the lab.

After traversing each line in the given file, the program ends by closing the file, printing a 'Goodbye!' statement, and printing its execution time.

## Experimental Results:

**Initial Test:** I began my testing by using the .txt file of system accounts provided to us, which contains 100 users all with real hashed passwords. My program was able to determine all of the user's passwords in just under 11 minutes (See below for partial terminal output of this case)

```
Success!
User92 password is: 562

Success!
User93 password is: 93499

Success!
User94 password is: 6675953

Success!
User95 password is: 9601

Success!
User96 password is: 12614

Success!
User97 password is: 7738

Success!
User98 password is: 1089

Success!
User99 password is: 378

Goodbye!
--- Execution Time: 644.8976156711578 seconds ---
```

Execution time: 10 minutes 45 seconds

### Test Cases for Various Input Sizes

Then, in order to test the speed of my program, I copied all 100 system accounts in the file and created 2 new .txt files. The first one had only the initial 50 users, while the second had all 100 users twice (for a total of 200 users. I chose to test my program this way so that I could observe how the size of the input affected the execution time of my program. Results of each test are below:

#### Test 1 – Execution Time: 4 minutes 24 seconds

```
Success!
User45 password is: 904

Success!
User46 password is: 04288

Success!
User47 password is: 1130

Success!
User48 password is: 65708

Success!
User49 password is: 601

Success!
User50 password is: 4264

Goodbye!
--- Execution Time: 263.4969627857208 seconds ---
```

## Test 2 – Execution Time: 23 minutes 12 seconds

```
Success!
User95 password is: 9601

Success!
User96 password is: 12614

Success!
User97 password is: 7738

Success!
User98 password is: 1089

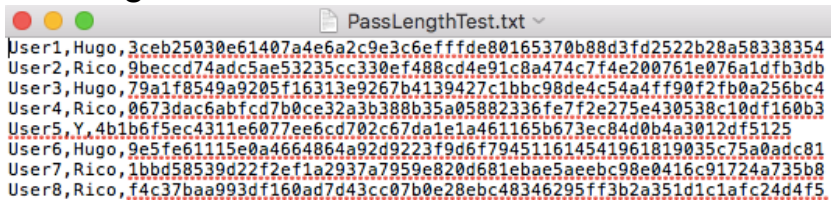
Success!
User99 password is: 378

Goodbye!
--- Execution Time: 1394.0020790100098 seconds ---
```

## Test Case for Various Password Lengths

I then proceeded to test my program using passwords of different lengths to see how these variables affected the execution time of my program. I used the `hash_with_sha256` method to develop a separate program called 'HashedPasswordGenerator' that receives a given password and salt value then outputs the generated hashed password. Using this program, I generated 8 system accounts of password lengths 1-8, respectively, and tested my program using this file. The results of this test are below:

## PassLengthsTest File: Execution Time: 9.5 minutes



```
User1,Hugo,3ceb25030e61407a4e6a2c9e3c6efffde80165370b88d3fd2522b28a58338354
User2,Rico,9beccd74adc5ae53235cc330ef488cd4e91c8a474c7f4e200761e076a1dfb3db
User3,Hugo,79a1f8549a9205f16313e9267b4139427c1bbc98de4c54a4ff90f2fb0a256bc4
User4,Rico,0673dac6abfcd7b0ce32a3b388b35a05882336fe7f2e275e430538c10df160b3
User5,Y,4b1b6f5ec4311e6077ee6cd702c67da1e1a461165b673ec84d0b4a3012df5125
User6,Hugo,9e5fe61115e0a4664864a92d9223f0d6f794511614541961819035c75a0adc81
User7,Rico,1bbd58539d22f2ef1a2937a7959e820d681ebae5aeebc98e0416c91724a735b8
User8,Rico,f4c37baa993df160ad7d43cc07b0e28ebc48346295ff3b2a351d1c1afc24d4f5
```

```
Welcome to THE PASSWORD BREAKER!

Enter the name of the file containing the system accounts:

PassLengthTest
Enter the minimum length of each password:

1
Enter the maximum length of each password:

8
Success!
User1 password is: 5

Success!
User2 password is: 22

Success!
User3 password is: 333

Success!
User4 password is: 4444

Success!
User5 password is: 54546

Success!
User6 password is: 666777

Success!
User7 password is: 7338908

Success!
User8 password is: 88877722

Goodbye!
--- Execution Time: 568.2966310977936 seconds ---
```

## Test Case for Erroneous Input & Non-Matching Salt Value to Hashed Password

Last but not least, I tested how my program handles erroneous input and nonmatching salt values to hashed passwords by creating a .txt file with one user account with a random salt value and randomly developed hash password. I entered a minimum length value of 1 and maximum length value of 5. The results of this test are below:

```
Welcome to THE PASSWORD BREAKER!
```

```
Enter the name of the file containing the system accounts:
```

```
WrongValuesTest
```

```
Enter the minimum length of each password:
```

```
0
```

```
Enter the maximum length of each password:
```

```
5
```

```
Lengths cannot be 0! Try Again.
```

```
Enter the minimum length of each password:
```

```
1
```

```
Enter the maximum length of each password:
```

```
1
```

```
Minimum length must be less than Maximum length! Try Again.
```

```
Enter the minimum length of each password:
```

```
1
```

```
Enter the maximum length of each password:
```

```
A
```

```
Lengths should be in integer format! Try Again.
```

```
Enter the minimum length of each password:
```

```
1
```

```
Enter the maximum length of each password:
```

```
5
```

```
ERROR: Password not found. Check that user has correctly formatted password and try again.
```

```
User1 password is: NOT FOUND
```

```
Goodbye!
```

```
--- Execution Time: 0.45112180709838867 seconds ---
```

**Conclusions:**

By solving this lab, I learned not only about hashing passwords with salt values using the `hash_with_sha256` method (used widely in industry), but also about various program functions using the Python programming language. For example, before completing this lab, I did not know how to read file line by line in Python. I also learned how to format integer values with leading zero's in Python, and how to implement 'try' and 'except' blocks in Python. This program also opened my eyes to the usability of recursive methods.

By analyzing the execution times of my program tests, I can estimate that the Big-Oh time complexity of my recursive method is  $O(n)$  where  $n$  is the number of users in the system accounts file.

Overall, this was a fun, yet challenging lab. Aside from having to learn how to perform various tasks in Python, figuring out the appropriate implementation of the solution to the problem presented while considering various types of input was arduous. Nonetheless, I enjoyed working on this lab assignment and look forward to future labs.

## Appendix:

```
# Lab 1: Password Cracking (Option C)
# Author: Hugo Rico (UTEP ID#:80531055 )
# Course: Data Structures (CS2302)
# Instructor: Diego Aguirre
# Program uses a recursive method to generate all possible passwords for a list
# of system accounts, determining each account's password in the process.
# Program receives a .txt file containing a list of records, where each record
# contains a username, a salt value, and a hashed password.
# -----
# Last Modified: September 19th, 2018

import hashlib
import time

def hash_with_sha256(str): #hash_with_sha256 method to create hashed passwords
    hash_object = hashlib.sha256(str.encode('utf-8'))
    hex_dig = hash_object.hexdigest()
    return hex_dig

#Password Generator works recursively to generate all possible passwords between given minimum and
maximum lengths
def passGenerator(saltValue,hashPassword,minLength,maxLength):
    if int(minLength) > int(maxLength):
        print('ERROR: Password not found. Check that user has correctly formatted password and try again.')
        passWord = 'NOT FOUND'
        return passWord
    upperBound = '9' * int(minLength) #Last password to check in a cycle
    passWord = '0' * int(minLength) #Initial password to check
    hex_dig = hash_with_sha256(passWord+saltValue)
    if hashPassword == hex_dig:
        print('Success!')
        return passWord
    else:
        while int(passWord) < (int(upperBound)+1):
            passWord = int(passWord) + 1
            passWord = str(passWord)

            #Used to format leading zero's
            if int(minLength) > 1:
                if int(passWord) <= int((int(minLength)-1)*'9'):
                    passWord = passWord.zfill(int(minLength))

            hex_dig = hash_with_sha256(passWord+saltValue)
            if hashPassword == hex_dig:
                print('Success!')
                return passWord
        newMinLength = int(minLength) + 1
        return passGenerator(saltValue,hashPassword,str(newMinLength),maxLength)

def main():
    class Error(Exception):
        #Base class for other exceptions
        pass

    class ZeroValueError(Error):
        #Raised when input value is 0
```

pass

```
class LengthValueError(Error):
```

```
#Raised when inputted minimum length value is greater than or equal to maximum length value
```

```
pass
```

```
print('Welcome to THE PASSWORD BREAKER!\n')
```

```
while True:
```

```
    print('Enter the name of the file containing the system accounts:')
```

```
    filename= input()
```

```
    try:
```

```
        while True:
```

```
            try:
```

```
                print('Enter the minimum length of each password:')
```

```
                minLength = int(input())
```

```
                print('Enter the maximum length of each password:')
```

```
                maxLength = int(input())
```

```
                if minLength == 0:
```

```
                    raise ZeroValueError
```

```
                if maxLength <= minLength:
```

```
                    raise LengthValueError
```

```
                break
```

```
            except ValueError:
```

```
                print('Lengths should be in integer format! Try Again.')
```

```
            except ZeroValueError:
```

```
                print('Lengths cannot be 0! Try Again.')
```

```
            except LengthValueError:
```

```
                print('Minimum length must be less than Maximum length! Try Again. ')
```

```
#Reading of .txt file with system accounts, line by line
```

```
startTime = time.time()
```

```
file = open(filename + '.txt','r')
```

```
while True:
```

```
    line = file.readline()
```

```
    fields = line.split(",")
```

```
#Extracting data:
```

```
userName = fields[0]
```

```
#End of File Reached
```

```
if userName == "":
```

```
    break
```

```
saltValue = fields[1]
```

```
hashPassword = fields[2].split('\n')[0]
```

```
passWord = passGenerator(saltValue,hashPassword,minLength,maxLength)
```

```
print(userName + ' password is: ' + passWord)
```

```
print("")
```

```
if not line:
```



```
        break
    file.close()

    print('Goodbye!')
    print('--- Execution Time: %s seconds ---' % (time.time() - startTime))
    break

except FileNotFoundError:
    print('Oops! ' + filename + ' was not found! Try again.')

main()
```

## **Academic Honesty Certification**

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

*Hugo Rico*