Data Structures

Fall 2020, Programming Assignment #2

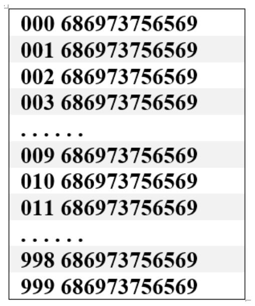
In computer systems, normally passwords are stored as hash values. For a hacker to conduct an attack, the common way is to hash every possible passwords and store their hash values in a file. If an attacker obtains a hash value, the attacker can compare this value against the hash values in the file. If a match is found, one immediately knows which password the hash value corresponds to. Then the password is recovered. In this programming assignment, the goal is to retrieve a password from a hash value. For a file with only 100 passwords, only 100 hash values would be generated. It is very easy to search 100 hash values. To increase the difficulty of such a dictionary attack, password ”*salt*” is introduced. The salt is a random value that we attach to a plaintext password. The hash value is calculated on the salted password (i.e. **H(salt ∥ password)**). The salt (i.e. the random value) adds the uncertainty to the password. For each password, the number of possible hash values is equivalent to the number of possible combinations of salt.

In this assignment, you will use a three-digit salt. This means the salt is a decimal number and each digit can be one of 10 possible numbers, namely 0-9. Since a salt has three digits, the number of all possible salt values is $10 \times 10 \times 10 = 1000$. For one password, there are 1000 possible hash values. For a file of 100 passwords, the total hash values are $100 \times 1000 = 100,000$. In other words, the search space for the dictionary attack is 100,000. One way to conduct the dictionary attack for the salted password is to attach every possible salt to a password and then hash the salted password (i.e. password with salt attached). Then we store all the hash values in a dictionary file. When we obtain a hash value for an unknown password, we search this value in the dictionary file. If we find a match, then the password is recovered (since the dictionary file should list which password the hash value corresponds to). With the salt, the size of the dictionary file increases 1000 times in this example.

In this assignment, each password has 6 uppercase letters. To calculate hash, those letters would be converted to numeric values. Here we use ASCII value to represent a letter. ASCII values are widely used in computer to store character. The ASCII table can be found at *https://www.asciitable.com/* . The uppercase letters have values ranging from 65 to 90. If a password is DEIKAE, you would first lay out the ASCII value of every letter side by side. So for the password of DEIKAE, you would generate 686973756569. Then you would insert the salt in front of this number sequence. For example, if a salt is 109, you would generate 109686973756569. Then apply the following hash function to this number sequence

$$Hash\ function = ((243 \times left) + right)\%85767489$$

Where, left is the left 8 digits of the number sequence. In this example, it is 10968697, which is treated as a long integer number. And right is the rest 7 digits of the number sequence. In this example. it is 3756569, which is treated as a long integer number. The result of the hash function is the hash value for this salted password. If we want to generate all possible hash values for this password, we need attach all possible values of salt to the password one by one. For the password DEIKAE (whose ASCII string is 686973756569). We should have 1000 salted password possibilities for this password as shown below. Then the hash values for those 1000 salted passwords will be stored in the dictionary file.



000 686973756569
001 686973756569
002 686973756569
003 686973756569
. . . . . .
009 686973756569
010 686973756569
011 686973756569
. . . . . .
998 686973756569
999 686973756569

1 Implementation

- **Inputs**.
  As mentioned above, we will provide a password file, which contains 100 passwords. Your program will ask the user to enter the name of the password file. Then the program will ask the user to enter a hash value. Your program will try to recover the password based on this hash value. Note this value should be treated as long integer.

- **Dictionary File.**
  Based on the input password file, you will generate a dictionary file which contains all possible hash values for all password. As mentioned above, for each password the possible hash values are 1000. Therefore, the dictionary file will have $100 \times 1000 = 100,000$ entries. Your program will save all entries in a text file. You are required to generate this text file. The entries of this dictionary file has the following format: [*password salt hash-value*]. For example, the first few entries of dictionary file should be similar to the following:

```
XEGUOQ 000 23411352
XEGUOQ 001 47711352
XEGUOQ 002 72011352
XEGUOQ 003 10543863
XEGUOQ 004 34843863
```

- **Outputs**
  Once the user enters the name of password file, your program can generate the dictionary file as described above. Now given a hash value, you can recover password by matching the given hash value to one of the hash values in the dictionary file. You will ask the user to enter a hash value as the target hash value. Then you program searches the dictionary file for this value. If a match was found, your program should print out indication that the password is recovered. Then the program should print out the recovered password (which is the password corresponds to the hash value) and the number of entries has been searched. If the user-entered hash value does not match any hash value in the dictionary file. You should also print out a line indicating the password is not found and number of entries has been searched. In this case, the number of entries searched should be 100,000.

- **Procedure**
  The basic procedure and requirements for this assignment are listed below:
  - Step 1: Prompt the user to enter the name of password file
  - Step 2: Generate the dictionary file
    Create a dictionary file and for each password
    - Step 2.1: Generate all possible 3-digit salt
    - Step 2.2: Attach each salt to the password
    - Step 2.3: Calculate hash value using the hash function
    - Step 2.4: Write password, salt and hash value to the file
    - Step 2.5: Repeat Step 2.2-2.4 for all salts
    - Step 2.6: Repeat Step 2.1-2.5 for all passwords
  - Step 3: Prompt the user to enter a hash value.
  - Step 4: Search the dictionary file for a match of this value. If a match is found, print out a success message and show the recovered password, salt value and number of entries have been searched. If no match is found, print out a failure message and show the number of entries haven been searched.