File Organization Term Project Spring-2024

Introduction

This project made for to organize, index and then analyze the passwords in a specific data set.

This indexing code is designed for fast search systems. Proceeding step by step. Every step of the way is explained in detail.

2. **Programming Language Python**

Python is used for writing the entire script due to extensive standard library which makes it suitable for file manipulation, hashing, and string operations.

Python Libraries

os: Used for interacting with the operating system. This includes creating directories, listing files in a directory, and moving files.Functions Used:

os.makedirs(): Creates directories.
os.path.exists(): Checks if a directory exists.
os.listdir(): Lists files in a directory.
os.path.join(): Joins directory paths.
os.rename(): Moves files from one directory to another.

hashlib:Provides hashing algorithms (MD5, SHA-1, SHA-256) to compute hash values for passwords.

Functions Used:
hashlib.md5(): Computes MD5 hash of a password.
hashlib.sha1(): Computes SHA-1 hash of a password.
hashlib.sha256(): Computes SHA-256 hash of a password.

time:Used for measuring the performance of password search operations.

Functions Used:
time.time(): Returns the current time, used to calculate the duration of search operations.

string:Provides constants for string operations, particularly to check if the initial character of a password is a letter.

random:Generates random passwords for performance testing.

Functions Used:
random.randint(): Generates a random integer within a specified range.
random.choices(): Generates a random sequence of characters from a specified pool of characters (letters and digits).

Tools:

Text editor and Visiual Studio for writing code.
CLI for running scripts and managing the file system.
The structured file system we created in the beginning to store and manage the project's data.

3.

```
def MakeFolder():
    folders = ['Unprocessed_Passwords', 'Processed', 'Code', 'Index']
    for folder in folders:
        if not os.path.exists(folder):
            os.makedirs(folder)
```

This function creates four folders: Unprocessed-Passwords, Processed, Code, and Index if they don't already exist.

'Unprocessed_Passwords' was used for downloaded raw text files.

'Processed' used for to store processed text files version of downloaded raw text files by program.

'Code' is a Placeholder for additional scripts, configuration files, and documentation related to the project

'Index' folder is where the script stores the indexed passwords. The passwords are organized into subfolders based on the initial character of each password to facilitate efficient searching.

4.Explaning The code (Indexing)

1.Making Necessary Folders:

```
def MakeFolder():
    folders = ['Unprocessed_Passwords', 'Processed', 'Code', 'Index']
    for folder in folders:
        if not os.path.exists(folder):
            os.makedirs(folder)
```

This function makes the directories Unprocessed_Passwords, Processed, Code, and Index if they do not already exist. These directories organize the workflow of password processing.

2.Indexing Passwords:

```
def IndexPass():
    IndexPath = 'Index'
    UnprocessedPath = 'Unprocessed_Passwords'

    for filename in os.listdir(UnprocessedPath):
        with open(os.path.join(UnprocessedPath, filename), 'r', encoding='utf-8', errors='ignore') as file:
            for line in file:
                password = line.strip()
                IndexFolder = password[0].lower()
                IndexFolder = IndexFolder if IndexFolder in string.ascii_lowercase else 'other'

                if not os.path.exists(os.path.join(IndexPath, IndexFolder)):
                    os.makedirs(os.path.join(IndexPath, IndexFolder))

                HashedPassword = hashlib.md5(password.encode()).hexdigest()
                sha128 = hashlib.sha1(password.encode()).hexdigest()
                sha256 = hashlib.sha256(password.encode()).hexdigest()
                #Using hash values(keys) from haslib library(sha128,sha256,md5) for hashmap
                IndexFile = os.path.join(IndexPath, IndexFolder, IndexFolder + '.txt')

                with open(IndexFile, 'a',encoding='utf-8', errors='ignore') as index:
                    index.write(f"{password}|{HashedPassword}|{sha128}|{sha256}|{filename}\n")
```

This function processes files containing passwords from the 'Unprocessed_Passwords' directory. It reads each password, sorts them into appropriate folders, and records hashes.

Reading Files: The function reads all filenames one by one in the 'Unprocessed_Passwords' directory.

Processing Each Password:

Opens each file and reads it line by line.(Reading passwords)

Removes any leading/trailing whitespace from each password.(Cleaning password)

Determines the folder based on the first character of the password. If the first character is a letter (a-z), it uses that letter as the folder name. Otherwise, it uses the folder named other.(Sorting)

Creates the necessary folders in the Index directory if they don't already exist.(

Calculates MD5, SHA-1, and SHA-256 hashes for each password.(Generating Hashes)

Appends the password and its hashes, along with the source filename, to a text file in the corresponding folder in the Index directory.(Recording)

3.Moving Processed Files From Unprocessed

```
def MoveToProcessed():
    UnprocessedPath = 'Unprocessed_Passwords'
    ProcessedPath = 'Processed'

    for filename in os.listdir(UnprocessedPath):
        source = os.path.join(UnprocessedPath, filename)
        destination = os.path.join(ProcessedPath, filename)
        os.rename(source, destination)
```

This function moves the processed password files from 'Unprocessed_Passwords' to the 'Processed' directory.
Reading Files: Lists all filenames in the 'Unprocessed_Passwords' directory.
Moving Files: Moves each file to the 'Processed'directory using os.rename.

4.Searching for Passwords

```
def SearchPassword(query):
    IndexPath = 'Index'
    query = query.strip().lower()
    IndexFolder = query[0].lower()
    IndexFolder = IndexFolder if IndexFolder in string.ascii_lowercase else 'other'

    IndexFile = os.path.join(IndexPath, IndexFolder, IndexFolder + '.txt')

    found = False
    result = []
```

```
with open(IndexFile, 'r', encoding='utf-8', errors='ignore') as index:
    for line in index:
        parts = line.strip().split('|')
        if parts[0] == query:
            result = parts
            found = True
            break
```

```
if not found:
    HashedPassword = hashlib.md5(query.encode()).hexdigest()
    sha128 = hashlib.sha1(query.encode()).hexdigest()
    sha256 = hashlib.sha256(query.encode()).hexdigest()
    result = [f"{query}|{HashedPassword}|{sha128}|{sha256}|search"]
    # Adding searched password to index for other searches
    with open(IndexFile, 'a', encoding='utf-8', errors='ignore') as index:
        index.write(f"{query}|{HashedPassword}|{sha128}|{sha256}|search\n")


return result
```

This function searches for a password in the indexed files. If found, it returns the password and its hashes. If not, it calculates the hashes, records them, and returns the data.

Prepare Search: Cleans the query and determines the index folder.
Open Index File: Opens the relevant index file.
Search for Password: Reads through the file line by line, splitting each line into components and checking if the password matches the query.
Return or Add Password: If found, returns the data. If not found, calculates the hashes, records the password in the index file, and returns the data.

5. Measuring Search Performance(Time):

```
def MeasureSearchTime():
    IndexPath = 'Index'
    totaltime = 0

    for _ in range(10):
        RandomPassword = GenerateRandomPassword()
        start = time.time()
        SearchPassword(RandomPassword)
        end = time.time()
        totaltime += end - start

    averagetime = totaltime / 10
    print("Average search time of 10 random passwords:", averagetime)
```

This function measures the average search time for ten random passwords.
Generate Random Passwords: Generates ten random passwords.
Measure Time: Measures and records the time taken to search for each password. Calculate Average: Calculates the average search time and prints it.

6.Generating Random Password:

```
def GenerateRandomPassword():
    import random
    import string
    length = random.randint(8, 16)
    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))
```

This function generates a random password of length between 8 to 16 characters.

7.Main Script:

```
if __name__ == "__main__":#Fixed code block
    MakeFolder()
    IndexPass()
    MoveToProcessed()
    MeasureSearchTime()
    query = input("Enter password for searching: ")
    start = time.time()
    result = SearchPassword(query)
    end = time.time()#End of the time
    print("Search result:", result)
    print("Search time:", end - start, "seconds")
```

This section runs the main functions to create folders, index passwords, move processed files, measure search performance, and search for a user-provided password.

5.Searching Function and Performance:

The search function, **`SearchPassword(query)`**, is implemented to locate a given password within a predefined index structure. Here's a detailed breakdown of how it works and how it processes user queries:

Determining the Index Folder:

The function determines the appropriate index folder based on the first character of the query. If the first character is a letter, the folder name will be that letter. If it's not a letter, it defaults to the 'other' folder.

Searching the Index File: The function constructs the path to the index file corresponding to the determined folder. It then opens the file in read mode and iterates through each line to find a match for the password.

Processing and returning the Results:

If a match is found, the function sets found to True and stores the result.

If no match is found, the function generates MD5, SHA-1, and SHA-256 hashes for the password, stores these hashes in a result list, and appends this information to the index file for future searches.

At last the function returns the result, which includes the password, its hashes, and the filename where it was found or "search" if it was newly added.

```python
def MeasureSearchTime():
    IndexPath = 'Index'
    totaltime = 0

    for _ in range(10):
        RandomPassword = GenerateRandomPassword()
        start = time.time()
        SearchPassword(RandomPassword)
        end = time.time()
        totaltime += end - start

    averagetime = totaltime / 10
    print("Average search time of 10 random passwords:", averagetime)
```

```python
def GenerateRandomPassword():
    import random
    import string
    length = random.randint(8, 16)
    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))
```

Generating and Searching Random Passwords:

A loop runs 10 times. In each iteration

A random password is generated using the `GenerateRandomPassword()` function, which creates a password of random length (between 8 and 16 characters) consisting of letters and digits.

The current time is recorded in `start`

The random password is searched using the `SearchPassword()` function.

The time taken for the search is recorded by subtracting start_time from the current time, resulting in `end - start`.

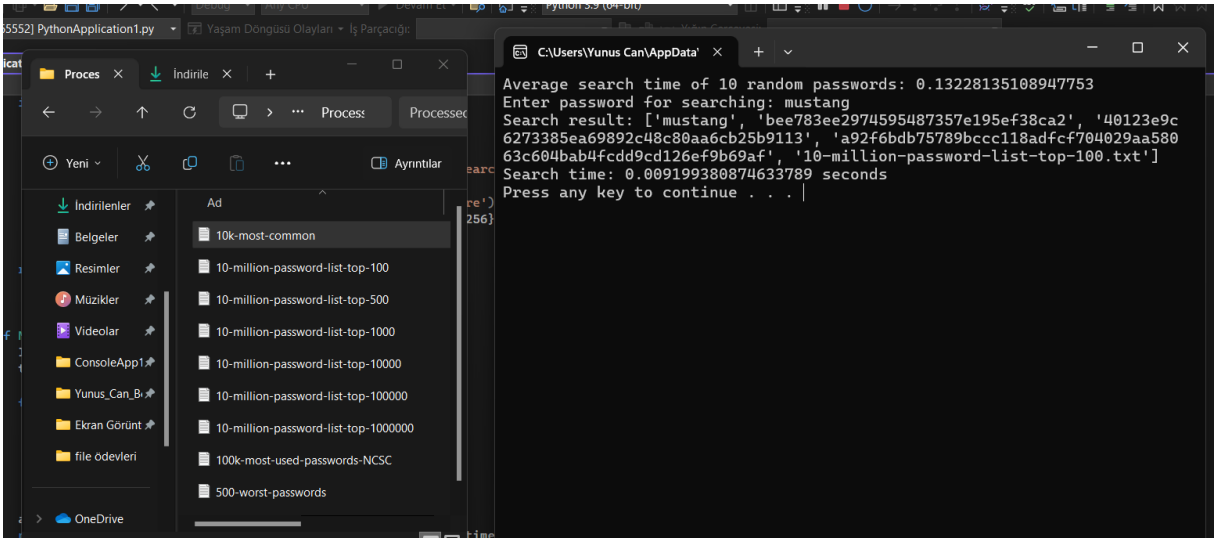
The elapsed time is added to `totaltime`.

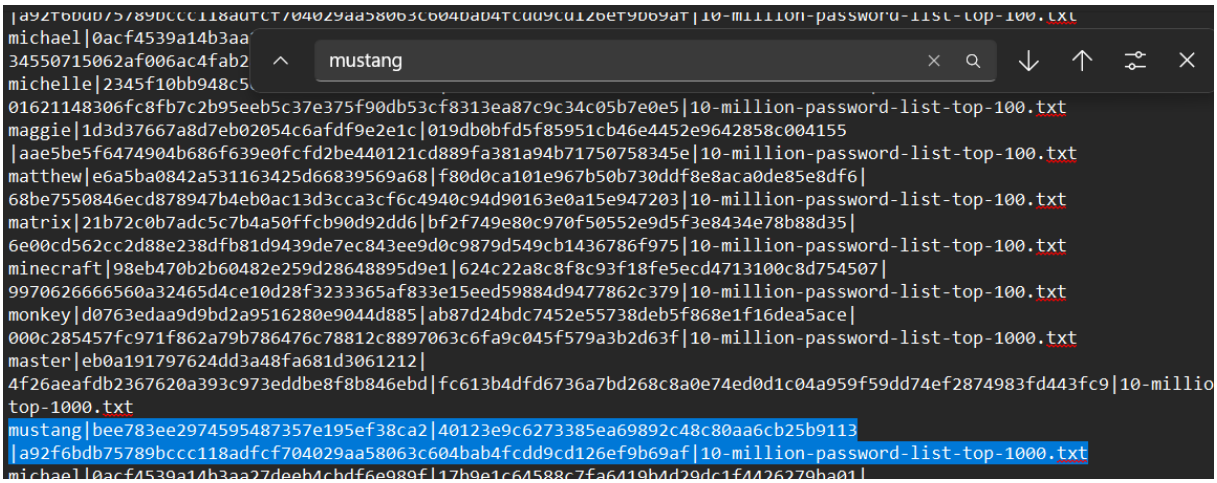
Calculating the Average Search Time:

After all iterations, the total search time is divided by 10 to calculate the average search time.

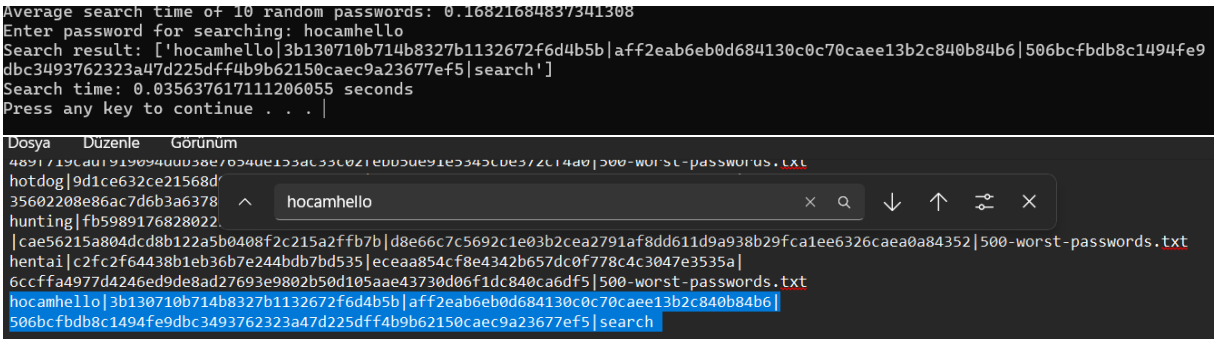
The average search time is printed.

Results:

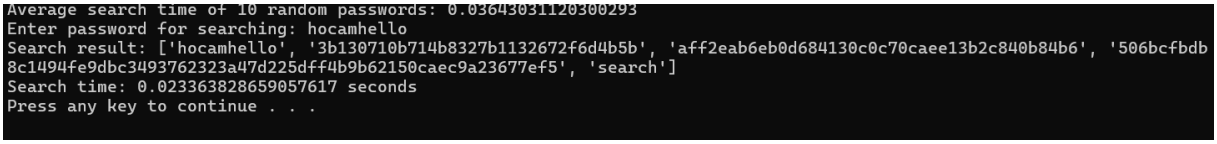




For a New Password:



After searching again new password:



7.Summary

The script initializes the folders, indexes the passwords, moves processed files, and measures search performance when run. It then prompts the user to enter a password to search for and prints the search result along with the search time.

The script has passed all tasks successfully.

The script only works for simple operations, security measures are not considered. Functional features can be added and security can be improved.

8.Referances

https://www.w3schools.com/python/module_os.asp
https://docs.python.org/3/library/hashlib.html
https://www.programiz.com/python-programming/time
https://www.w3schools.com/python/module_random.asp
https://www.simplilearn.com/tutorials/python-tutorial/index-in-python
https://stackoverflow.com/questions/419163/what-does-if-name-main-do

1306220069 Yunus Can Berber