PassBreach

Psuedo code - Initialize vars:

Initialize targetHash as the hash to be cracked

Initialize characters as the set of possible characters

Initialize numThreads as the number of threads to use

Initialize queue as a concurrent queue for string combinations

Initialize visited as a concurrent dictionary for visited combinations

Psuedo code - Seed Initial Queue data structure:

For each character in characters:

Enqueue the character to the queue

Psuedo code - CrackPassword function:

Function CrackPassword(queue, visited, targetHash):

While queue is not empty:

Dequeue currentString from queue

If CheckHash(currentString, targetHash) is true:

Print "Password found: " + currentString

Terminate all threads and return

For each character in characters:

newString = currentString + character

If newString is not in visited:

Add newString to visited

Enqueue newString to queue

Psuedo code - CheckHash function:

Function CheckHash(input, targetHash):

Compute hash of input

If computed hash equals targetHash:

Return true

Else:

Return false

Psuedo code - Main function:

Function Main():

Initialize targetHash, characters, numThreads, queue, and visited

Seed the queue with initial character strings

Create numThreads threads:

For each thread:

Start thread to run CrackPassword with queue, visited, and targetHash

Wait for all threads to complete

Print "Password not found."

Psuedo code - Full System overview:

# Initialize Variables

targetHash = "your\_target\_hash\_here"

characters = "abcdefghijklmnopqrstuvwxyz"

numThreads = 4

queue = ConcurrentQueue()

visited = ConcurrentDictionary()

# Seed Initial Queue

For each character in characters:

queue.enqueue(character)

# Define CrackPassword Function

Function CrackPassword(queue, visited, targetHash):

While queue is not empty:

currentString = queue.dequeue()

If CheckHash(currentString, targetHash):

Print "Password found: " + currentString

Terminate all threads and return

For each character in characters:

newString = currentString + character

If visited.tryAdd(newString, true):

queue.enqueue(newString)

# Define CheckHash Function

Function CheckHash(input, targetHash):

hash = ComputeHash(input) # Use SHA256 or another hashing algorithm

If hash equals targetHash:

Return true

Else:

Return false

# Main Function

Function Main():

# Initialize variables

targetHash = "your\_target\_hash\_here"

characters = "abcdefghijklmnopqrstuvwxyz"

numThreads = 4

queue = ConcurrentQueue()

visited = ConcurrentDictionary()

# Seed the initial queue with character strings

For each character in characters:

queue.enqueue(character)

# Create and start threads

For i from 0 to numThreads - 1:

Start thread to run CrackPassword(queue, visited, targetHash)

# Wait for all threads to complete

Wait for all threads to finish

Print "Password not found."

--------------------------------------------------------------------------------------

Psuedo code - Reworked system for reusability and testability:

--------------------------------------------------------------------------------------

Psuedo code - Interfaces and Factory methods:

# Define IHashAlgorithm Interface

Interface IHashAlgorithm:

Method ComputeHash(input: string): string

# Define MD5HashAlgorithm Class

Class MD5HashAlgorithm Implements IHashAlgorithm:

Method ComputeHash(input: string): string

# Define SHA256HashAlgorithm Class

Class SHA256HashAlgorithm Implements IHashAlgorithm:

Method ComputeHash(input: string): string

# Define HashAlgorithmFactory

Class HashAlgorithmFactory:

Method CreateHashAlgorithm(type: string): IHashAlgorithm

If type == "MD5":

Return new MD5HashAlgorithm()

Else If type == "SHA256":

Return new SHA256HashAlgorithm()

Else:

Throw Exception("Unsupported hash algorithm")

# Define IQueueManager Interface

Interface IQueueManager:

Method Enqueue(string: string): void

Method Dequeue(): string

# Define ConcurrentQueueManager Class

Class ConcurrentQueueManager Implements IQueueManager:

Method Enqueue(string: string): void

Method Dequeue(): string

# Define IHashCracker Interface

Interface IHashCracker:

Method CrackPassword(): void

Psuedo code - Implement classes:

# Implement MD5HashAlgorithm Class

Class MD5HashAlgorithm Implements IHashAlgorithm:

Method ComputeHash(input: string): string

# Compute and return MD5 hash

# Implement SHA256HashAlgorithm Class

Class SHA256HashAlgorithm Implements IHashAlgorithm:

Method ComputeHash(input: string): string

# Compute and return SHA-256 hash

# Implement ConcurrentQueueManager Class

Class ConcurrentQueueManager Implements IQueueManager:

ConcurrentQueue queue = new ConcurrentQueue()

Method Enqueue(string: string): void

queue.Enqueue(string)

Method Dequeue(): string

Return queue.TryDequeue(out string result) ? result : null

# Implement HashCracker Class

Class HashCracker Implements IHashCracker:

Constructor(queueManager: IQueueManager, hashAlgorithm: IHashAlgorithm, targetHash: string, characters: char[])

# Initialize fields

Method CrackPassword(): void

While queueManager is not empty:

currentString = queueManager.Dequeue()

If hashAlgorithm.ComputeHash(currentString) == targetHash:

Print "Password found: " + currentString

Exit

For each character in characters:

newString = currentString + character

queueManager.Enqueue(newString)

Psuedo code - Main function:

# Main Function

Function Main():

# Initialize variables

targetHash = "your\_target\_hash\_here"

characters = "abcdefghijklmnopqrstuvwxyz"

numThreads = 4

hashType = "SHA256" # or "MD5", based on user input

# Create instances

hashAlgorithmFactory = new HashAlgorithmFactory()

hashAlgorithm = hashAlgorithmFactory.CreateHashAlgorithm(hashType)

queueManager = new ConcurrentQueueManager()

# Seed initial queue with character strings

For each character in characters:

queueManager.Enqueue(character.ToString())

# Create and start threads

For i from 0 to numThreads - 1:

Start thread to run new HashCracker(queueManager, hashAlgorithm, targetHash, characters).CrackPassword()

# Wait for all threads to complete

Wait for all threads to finish

Print "Password not found."

Psuedo code - Summary:

Interfaces:

* Ensure consistent functionality and enable easy extension for different hashing algorithms (IHashAlgorithm), queue management (IQueueManager), and password cracking (IHashCracker).

Factory Pattern:

* Simplifies the instantiation of hash algorithms based on user input.

Concurrent Processing:

* Utilizes ConcurrentQueueManager for thread-safe queue operations.

Reusability and Readability:

* Modular design with clear separation of concerns, making the codebase easy to understand, maintain, and extend.