Implementación del Hash del cuco para búsqueda y encriptación en una base de datos

Anthony Bautista[1], Cristopher García[2], Rosa Limachi[3] y Julio Rosales[4]
[1 2 3] Ciencias de la Computación [4] Matemática
Universidad Nacional de Ingeniería, Lima, Perú.
{[1]abautistal, [3]rlimachip}@uni.pe {[2]crisebas100, [4]julio845}@gmail.com

***Resumen*—En este informe se propone una forma de obtener una búsqueda de usuarios de una base de datos en tiempo constaste, así como asegurar sus constraseñas, empleando el lenguaje python para la implementación y el algoritmo de hash del cuco para la búsqueda y la encriptación.**

I. Introducción

Para un administrador de la base de datos de una institución siempre surge el problema de identificar de una manera rápida quién y cómo se usa los recursos del sistema. Una solución ha este problema sería la creación de un sistema de autenticación de usuarios donde se genera y almacena un nombre de usuario y su respectiva contraseña. Aunque esta solución es válida sigue habiendo el problema de la suplantación de usuarios, por el acceso no autorizado de las contraseñas en la base de datos donde se almacenaba, y el aumento de tiempo en la búsqueda de usuarios a medida que estos aumenten. Por ello se tendría que encriptar las contraseñas de una forma que no se pueda obtener la real a partir de la encriptada y emplear un algoritmo que encuentre a los usuarios mediante una operación constante.

II. Métodos

A. Base de datos
Colección de información organizada de un modo específico para que su contenido pueda ser tratado y analizado de manera rápida.

B. Diccionario
Estructura de datos para almacenar un conjunto de elementos que admite tres operaciones básicas: Búsqueda(x) que devuelve verdadero si x está en el conjunto actual, y falso en caso contrario; Insertar(x) que agrega el elemento x al conjunto actual si aún no está presente; Eliminar(x) que elimina x del conjunto actual si está presente [4].

C. Función Hash
Función que a partir de una entrada que suele ser una cadena, genera una salida (cadena) de longitud fija, esta tiene información. La colisión hash se produce cuando entradas distintas generan el mismo valor en una función hash. Ver Algoritmo 1.

Algorithm 1: funcion_hash(funcion,clave)

```
if cont == n then
    aux.append(clave)
for j = 0 to ver do
    suma = suma + ord(i)
switch funcion do
    case 1 do
        return suma %n
    case 2 do
        return n − (suma %n) − 1
```

D. Tabla Hash
Estructura de datos que relaciona claves y valores para cada elemento que guarda. Utilizaremos una función hash para transformar la clave de un dato e identificar el lugar que ocupará en la tabla.

E. Colisión Hash
Situación donde al aplicar una función hash a dos entradas distintas generan igual valor.

F. Hash del cuco
Algoritmo que permite resolver las colisiones hash ofreciendo para un valor x la posibilidad de tomar la posición $h_1(x)$ o $h_2(x)$. Esto nos permitirá buscar un elemento observando solo dos posiciones en la matriz. Al insertar un nuevo elemento x, por supuesto, todavía puede ocurrir que no haya espacio, ya que tanto la posición $h_1(x)$ como la posición $h_2(x)$ pueden ocuparse. Esto se resuelve imitando los hábitos de anidamiento del cuco europeo, desplazando el ocupante actual de la posición $h_1(x)$ para dejar espacio. Si la posición alternativa $h_1(x)$ está disponible, el valor es ingresado. De lo

contrario, $h_1(x)$ repite el comportamiento de x y desplaza al ocupante. Esto se continúa hasta que el procedimiento encuentra una posición disponible o ha tardado demasiado. En este último caso, se eligen nuevas funciones hash y se reconstruye toda la estructura de datos [4].

G. Filtro de Bloom

Es una estructura de datos probabilística tipo Monte Carlo donde las búsquedas pueden cometer errores. La búsqueda solo retorna un booleano que indica cuando la clave existe o no dentro del diccionario.

III. Estado del Arte

1) Cuckoo Hashing
   Presentamos un diccionario simple con el tiempo de búsqueda constante en el peor de los casos, que iguala el rendimiento teórico del esquema de hash dinámico clásico de Dietzfelbinger, siendo este más complicado de implementar. Además, se menciona algunos retos a superar. Como el no haberse encontrado una familia de función hash explícita que sea probablemente buena para el esquema y si aumentar tablas mejorara el rendimiento de la memoria.[1]
2) Hashing: Técnicas y Hash para la Protección de Datos
   Se compara las diversas técnicas de hashing, donde cada una de estas puede presentar colisiones, con un costo computacional alto. Además, muestra sus diversos usos como en el cifrado de contraseñas, en creación de certificados digitales, cuando ocurre un ataque de base de datos .[2]
3) Algoritmo hash y vulnerabilidad a ataques
   Explica el problemas del ciframiento de los datos al aplicar encriptación por hash, donde se propone como solución emplear dos algoritmos como el SHA-1 y RIPEND-160.[3]
4) Cuckoo Filter: Practically Better Than Bloom
   Se propone una nueva estructura de datos llamada Cuckoo filter que puede reemplazar los Bloom filters para las pruebas de pertenencia a conjuntos aproximados. Los filtros de cuco son compatibles con la inserción y eliminación de elementos dinámicamente, mientras se logra un rendimiento aún mayor que los filtros Bloom.[5]

IV. Diseño del experimento

El experimento se realizara con el lenguaje python creando 4 módulos:

1) Main
   Módulo principal donde se muestra las opciones de registro, validación de usuario y salir. Basta con ejecutar esta función para que el programa funcione.
2) Datos
   Almacena las funciones que permiten saber si el usuario que se registra existe, así como validar su contraseña.
3) Cuckoo
   Posee la función colocar que se usa tanto para desordenar las contraseñas ingresadas en forma de una sucesión de carácteres, como para almacenar las cuentas de usuarios en las tablas hash, mediante el algoritmo del hash del cuckoo, como se muestra en Algoritmo 2. Ver Figura 1.

Algorithm 2: Colocar(clave, tabla, cont, n)

```
if cont == n then
    aux.append(clave)
for j = 0 to ver do
    pos[i] = funcion_hash(i + 1, clave)
    if hashTable[tabla][pos[tabla]]! = 0 then
        save = hashTable[tabla][pos[tabla]]
        hashTable[tabla][pos[tabla]] = clave
        colocar(save, (tabla + 1) %ver, cont +
         1, n)
    else
        hashTable[tabla][pos[tabla]] = clave
```

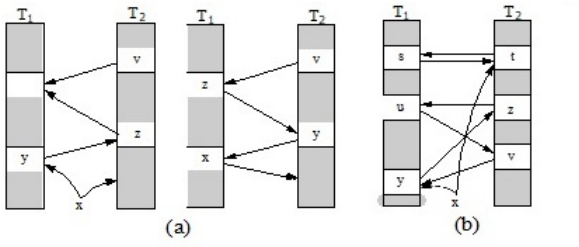

Figura 1. Ejemplos de inserción de cuckoo Hashing. Las flechas muestran posibilidades para mover las teclas. (a) La clave x se inserta con éxito moviendo las teclas z e y de una tabla a la otra. (b) No se puede acomodar la llave x y es necesario un refrito.

4) Recepcion_datos
   Permite el almacenamiento de usuario y contraseñas en archivos cvs, encriptando la contraseña mediante el hash del cuco. Además, se verifica si el usuario con su respectiva contraseña existe.

V. Anexo

Ver el cuaderno de jupyter en el siguiente enlace: https://github.com/crisebas/Cuckoo-Hashing/blob/master/experimentacion.ipynb

Referencias

[1] Rasmus, Pagh., Flemming Friche Rodler. (2001) Cuckoo Hashing. Journal of Algorithms.

[2] Samuel, Sánchez., Pablo, Domínguez., Luis Velásquez. (2009) Hashing: Técnicas y Hash para la Protección de Datos. Universidad Tecnológica de Panamá.

[3] Mena Miranda, Yerko. (2009) Algoritmos HASH y vulnerabilidad a ataques. Universidad Mayor De San Andrés.

[4] Rasmus, Pagh. (2006) Cuckoo Hashing for Undergraduates. IT University of Copenhagen.

[5] Fan, Andersen, Kaminsky, Mitzenmacher(2014) Cuckoo Filter: Practically Better Than Bloom. CoNEXT '14