Trabajo cuatro

Aplicaciones de las tablas Hash

Simón Cuartas Rendón
C.C. 1.037.670.103

Universidad Nacional de Colombia
Facultad de Minas
Estructuras de datos
Medellín
2021

1. Usos de las tablas Hash en la actualidad

Las tablas hash son una de las estructuras de datos más utilizadas por los diseñadores de software y programadores en general gracias a la versatilidad que ofrecen en su implementación y la altas eficiencias que ofrecen en promedio para operaciones básicas como inserción, eliminación o recuperación de datos. En ese sentido, se va a indagar sobre algunas de las aplicaciones más frecuentes que éstas encuentran en la actualidad.

Un ejemplo básico pero ampliamente usado es la correlación entre dos variables que están mutuamente ligadas, pues con una se puede acceder a la otra; es decir, se puede pensar como información tabular en solo dos columnas, de manera que una de ellas tiene la llave que permite acceder al valor que está asociado a la otra columna. Ejemplos de lo anterior incluyen el acceso a información básica de ciudadanos mediante su documento de identidad, a través del cual se pueden obtener los datos relacionados como el nombre del ciudadano, su residencia o su correo electrónico. También se tiene la gestión de usuarios registrados en un sitio web, pues con la contraseña que permite a un usuario loguearse en dicha web, se la toma como la clave que permite acceder al valor asociado, que en este caso sería el seudónimo o el correo electrónico del usuario, dando autorización para acceder a la página en caso de que coincidan con lo que está almacenado [1][2].

Por otro lado, se tiene que la criptografía es una de las áreas donde las tablas hash encuentran aplicaciones con más frecuencia, puesto que las funciones hash que se emplean facilitan la generación de códigos, generalmente únicos cuando dichas funciones son robustas, de manera que permiten identificar a un determinado string de caracteres o el permiso de acceso a algún archivo que va a ser encriptado. Así pues, si aquello que ha sido encriptado es tratado de ser modificado, esto podrá ser detectado. En particular, se puede pensar en la necesidad de guardar algún documento en algún servidor en la nube sin acceso a terceros, permitiendo al punto de acceso local almacenar la llave de acceso al documento, la cual es la clave en la tabla hash asociada, y así, al descargar el documento, se verificará si las llaves almacenadas en el punto remoto y el punto local desde el cual se intenta acceder al archivo son las mismas, con lo cual autorizará o rechazará el ingreso al archivo [2].

Otro elemento interesante es la identificación de patrones de algunas palabras clave que son frecuentemente empleadas en servicios de escritura de texto o código, de manera que se pueda autocompletar con rapidez cuando se

identifica la escritura de algún segmento de la llave de alguna de las palabras frecuentemente usadas [2].

Adicionalmente, se puede mencionar el caso de *Shazam*, pues fue de las primeras compañías en ofrecer un servicio de reconocimientos de canciones mediante la grabación de un segmento de esta, lo cual era posible gracias a la conversión de los sonidos recibidos en señales, basados en variables como la frecuencia y amplitud de las ondas de sonido percibidas, de manera que pudiesen ser codificadas mediante procesamientos matemáticas como la transformada de Fourier y así poder contrastadas con las llaves disponibles en una tabla hash que contiene las llaves codificadas de miles de canciones para poder hacer la búsqueda de canciones, de manera que se pueda recuperar el nombre de la canción y el artista que quieren ser hallados [3].

Por otro lado, dentro del mismo contexto de la programación, se tiene que diversos compiladores de código apelan a las tablas hash para poder llevar un registro de *'cosas'* las variables, funciones o métodos, clases, objetos e interfaces, entre otros, que va leyendo el compilador conforme va procesando el código, de manera que cuando se tope nuevamente con alguna de las *'cosas'* que ya han sido registradas, pueda identificarlo rápidamente en tiempo de ejecución y poder ser empleado en el contexto del código [4].

Y continuando con la programación, se puede mencionar que diversos lenguajes dinámicos como JavaScript, Perl y Ruby hacen uso de las tablas hash para poder implementar objetos, de manera que las llaves están relacionados con nombres de las instancias de un tipo de objeto determinado, así como a los métodos asociados a dichas instancias, mientras que los valores se constituyen como punteros para la instancia o método correspondiente [5].

Asimismo, las tablas hash pueden ser empleadas para la implementación de antememoria o caché, de forma que se pueda acceder de forma rápida a información auxiliar que ayuden a acelerar la ejecución de otros procesos que se apoyan precisamente en la memoria caché pero que tal vez no pueden incorporar por sí mismas las tablas hash dada la forma en que están estructurados sus datos [6].

Y así, se puede evidenciar que las tablas hash son especialmente útiles para aplicaciones que requieren enlazar datos que están mutuamente ligados, pues la manipulación de sus valores se puede dar fácilmente en tiempo de ejecución, y muestra de su versatilidad se da en el amplio rango de aplicaciones que pueden encontrar las tablas hash en la vida real para solucionar problemas o crear productos o servicios competitivos.

2. Usos de las tablas Hash en la actualidad

Se va a hacer el uso de tablas hash para implementar de forma sencillo la autorización para el logueo de los usuarios de una página web X.

Esta ofrece al usuario tres opciones: registrarse, loguearse o salir del aplicativo. Dado que la idea es simplemente mostrar la aplicación de las tablas hash en este contexto, al ejecutarse las opciones ***uno*** o ***dos*** se retorna al menú principal hasta que se salga.

```
¡Bienvenido a la página X!
Ingrese el número de la opción que desee:
1. Registrarme.
2. Ingresar.
3. Salir.
```

Este se basa en una clase denominada `usuario`, la cual tiene dos atributos que son de tipo String: `nombre`, que corresponde al seudónimo que emplea el usuario para registrarse y posteriormente loguearse y `clave`, que es la contraseña con la cual se va a loguear. Además, hay un atributo estático que es una tabla **hash,** implementada en Java como un `HashMap` y que se llama `usuarios`, donde tanto la clave como el valor son strings y corresponden al seudónimo y la contraseña del usuario respectivamente. Esta tabla hash va a ser útil para poder incorporar todos los usuarios que se han registrado en la página. Con esto, se tiene un constructor para esta para poder relacionar el seudónimo y la contraseña asociada, así como agregar estos dos elementos a la tabla de `usuarios`.

```java
public class usuario {

    static HashMap<String, String> usuarios = new HashMap<>();

    String nombre;
    String clave;

    public usuario(String nombre, String clave) {
        this.nombre = nombre;
        this.clave = clave;
        usuarios.put(nombre, clave);
    }
```

De esta manera, se crea el método de registro de usuarios llamado `nuevoUsuario()`, el cual solicita al usuario elegir su alias, y con esto se ejecuta un `if` que chequea si ese seudónimo está disponible, de manera que informe al usuario si debe cambiarlo en caso de estar ocupado o si puede continuar consultándole la contraseña que va a emplear para poder finalizar un registro utilizando el constructor descrito previamente. Finalmente se le confirma al usuario la contraseña y el alias ingresados.

El código de este método es el que se ve a continuación:

```
public static void nuevoUsuario() {
    System.out.println("Bienvenido. Por favor, elije un seudónimo:");
    String nombre = entrada.next();

    if (usuarios.containsKey(nombre)) {
        System.out.println("El nombre de usuario ya está registrado. Intente con otro diferente");
    }

    else {
        System.out.println("Elije la clave que vas a usar");
        String clave = entrada.next();

        usuario nuevo = new usuario(nombre, clave);
        System.out.println("¡Se ha realizado el registro con éxito!");
        System.out.println("Usted es el usuario con seudónimo " + nuevo.nombre + " y contraseña " + nuevo.clave);
        System.out.println("\n");
    }
}
```

Ahora bien, supongamos que un usuario se quiere registrar con el nombre de ***coco***, y que éste está disponible, por lo que vería lo siguiente en consola:

```
Ingrese el número de la opción que desee:
1. Registrarme.
2. Ingresar.
3. Salir.
1
Bienvenido. Por favor, elije un seudónimo:
coco
Elije la clave que vas a usar
Amist@de$567
¡Se ha realizado el registro con éxito!
Usted es el usuario con seudónimo coco y contraseña Amist@de$567
```

Y más adelante un usuario diferente decide registrarse en la página intentado usar el seudónimo ***coco***, que como ya se vio está ahora ocupado, por lo que su mensaje sería el que se ve en la primera imagen de la siguiente página:

```
¡Bienvenido a la página X!
Ingrese el número de la opción que desee:
1. Registrarme.
2. Ingresar.
3. Salir.
1
Bienvenido. Por favor, elije un seudónimo:
coco
El nombre de usuario ya está registrado. Intente con otro diferente
```

Por otro lado, el método que se encarga del logueo de usuarios es denominado `logueo()`, y requiere al usuario que ingrese el seudónimo y la contraseña asociada para poder ingresar a la página. Hecho esto, se tiene un `if` que confirma que en la tabla hash que registra a todos los usuarios en efecto exista un usuario con el seudónimo ingresado. Si lo anterior no sucede, se le informa al usuario y se le invita a intentarlo de nuevo y lo retorna al menú principal. En la situación opuesta, se verifica que la contraseña ingresada sea la correcta, caso en el cual se le informa al usuario que fue logueado exitosamente; en el caso contrario, se le informa al usuario que la clave es errónea y se le envía al menú principal. El código que permite lo recién descrito es el siguiente:

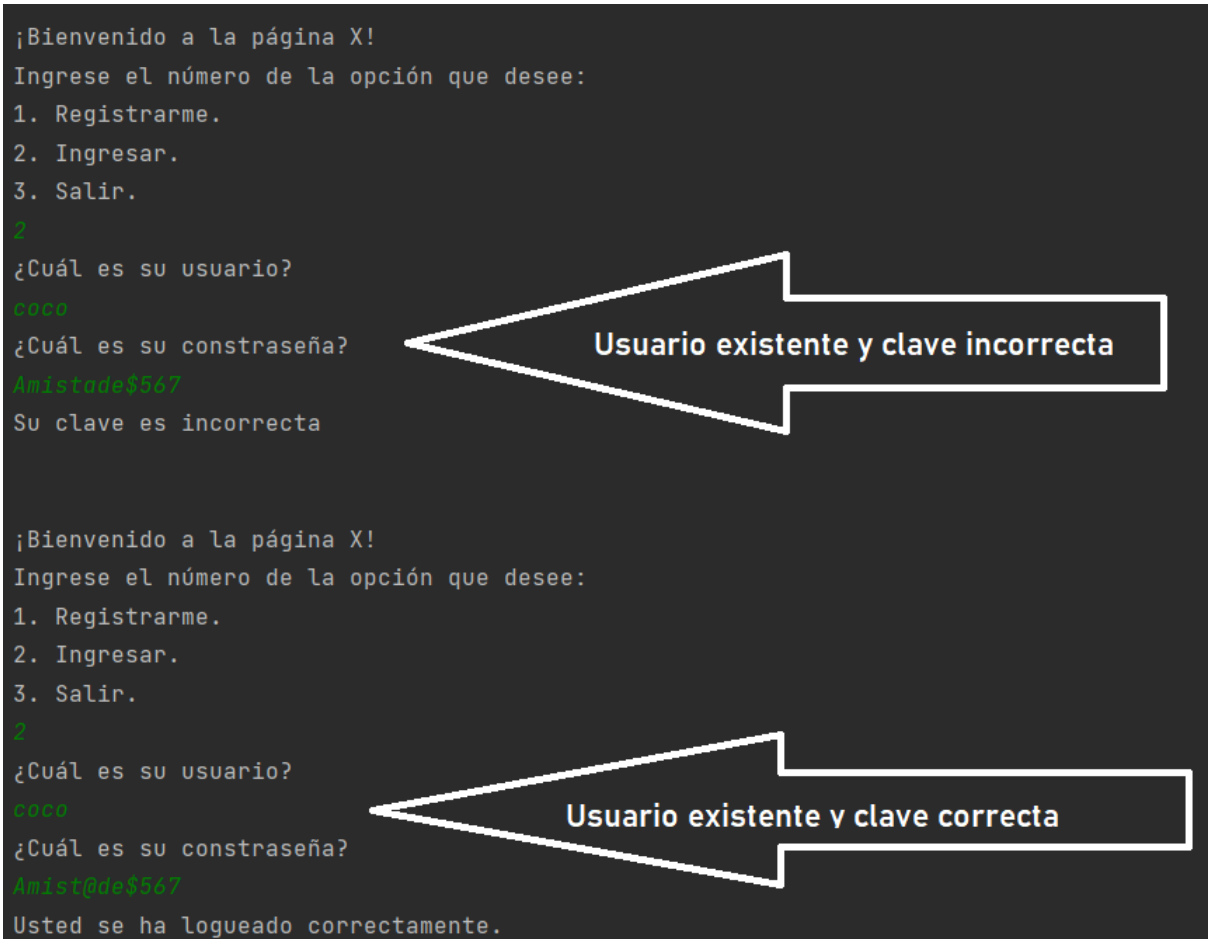
```java
public static void logueo() {
    System.out.println("¿Cuál es su usuario?");
    String nombre = entrada.next();
    System.out.println("¿Cuál es su constraseña?");
    String clave = entrada.next();

    if (!usuarios.containsKey(nombre)) {
        System.out.println("No existe un usuario con el seudónimo ingresado. Inténtelo de nuevo.");
    }
    else if (usuarios.containsKey(nombre)) {
        if (clave.equals(usuarios.get(nombre))) {
            System.out.println("Usted se ha logueado correctamente.");
        }
        else {
            System.out.println("Su clave es incorrecta");
        }
    }
    System.out.println("\n");
}
```

Con esto claro, veamos un ejemplo: de momento, solo existe el usuario ***coco***, por lo que al tratar de ingresar con el seudónimo ***coco123*** se retorna al menú principal, como se puede ver a continuación:

```
¡Bienvenido a la página X!
Ingrese el número de la opción que desee:
1. Registrarme.
2. Ingresar.
3. Salir.
2
¿Cuál es su usuario?
coco123
¿Cuál es su constraseña?
Amist@de$567
No existe un usuario con el seudónimo ingresado. Inténtelo de nuevo.
```

Sin embargo, si el usuario sí existe, se ejecuta un nuevo `if` para verificar que la contraseña dada por el usuario sí corresponda con la asociada al seudónimo ingresado. Si lo anterior ocurre, se le informa al usuario que ha sido logueado correctamente; de lo contrario, se le informa que la clave es errónea. La apariencia de esto se aprecia en la primera imagen de la siguiente imagen.



Adicionalmente, se incorporó una opción secreta al usuario según la cual, si se elige la opción cero (0), se imprime por consola todos los usuarios registrados y sus respectivas contraseñas (en orden no secuencial), lo cual se logra el iterar a través de la tabla hash `usuarios`. Esto se realiza para efectos pedagógicos de evidenciar la implementación de este método, y su resultado se puede ver en la segunda imagen de la siguiente página.

```
¡Bienvenido a la página X!
Ingrese el número de la opción que desee:
1. Registrarme.
2. Ingresar.
3. Salir.
0
Usuario: palomaValencia --- Contraseña: uribeTeAmo
Usuario: konichiwa --- Contraseña: tokyo2020
Usuario: coco --- Contraseña: Amist@de$567
Usuario: HolaSoyGerman --- Contraseña: teApuestoUnaPizza
```

Para finalizar, se presenta el código completo a continuación:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class usuario {

    static HashMap<String, String> usuarios = new HashMap<>();

    String nombre;
    String clave;

    public usuario(String nombre, String clave) {
        this.nombre = nombre;
        this.clave = clave;
        usuarios.put(nombre, clave);
    }

    public static void nuevoUsuario() {
        System.out.println("Bienvenido. Por favor, elige un seudónimo:");
        String nombre = entrada.next();

        if (usuarios.containsKey(nombre)) {
            System.out.println("El nombre de usuario ya está registrado. Intente con otro
diferente");
        }

        else {
            System.out.println("Elige la clave que vas a usar");
            String clave = entrada.next();

            usuario nuevo = new usuario(nombre, clave);
            System.out.println("¡Se ha realizado el registro con éxito!");
```

```
            System.out.println("Usted es el usuario con seudónimo " + nuevo.nombre + " y
contraseña " + nuevo.clave);
            System.out.println("\n");
        }
    }

    public static void logueo() {
        System.out.println("¿Cuál es su usuario?");
        String nombre = entrada.next();
        System.out.println("¿Cuál es su constraseña?");
        String clave = entrada.next();

        if (!usuarios.containsKey(nombre)) {
            System.out.println("No existe un usuario con el seudónimo ingresado. Inténtelo de
nuevo.");
            System.out.println("\n");
        }
        else {
            if (clave.equals(usuarios.get(nombre))) {
                System.out.println("Usted se ha logueado correctamente.");
            }
            else {
                System.out.println("Su clave es incorrecta");
            }
        }
        System.out.println("\n");
    }

    public static void baseDatos() {
        for (Map.Entry<String, String> stringStringEntry : usuarios.entrySet()) {
            System.out.println("Usuario: " + ((Map.Entry) stringStringEntry).getKey() + " --- " +
"Contraseña: " + ((Map.Entry) stringStringEntry).getValue());
        }
        System.out.println("\n");
    }

    static Scanner entrada = new Scanner(System.in);

    public static void main(String[] args) {

        int opcion; // Entero para el número de opción a usar en el menú
        do {
            System.out.println("¡Bienvenido a la página X!");
            System.out.println("Ingrese el número de la opción que desee:");
            System.out.println("1. Registrarme.");
            System.out.println("2. Ingresar.");
            System.out.println("3. Salir.");

            opcion = entrada.nextInt();

            switch (opcion) {
                case 0 -> baseDatos();
                case 1 -> nuevoUsuario();
                case 2 -> logueo();
                case 3 -> System.out.println("¡Hasta luego!");
            }
        }
        while (opcion != 3);

    }
}
```

En cualquier caso, es posible ver y descargar el código en el siguiente enlace:

https://github.com/scuartasr/logueoED

Este repositorio es público, por lo que no se requiere invitación para acceder a él.

2.1. Análisis algorítmico del código

A continuación se da un análisis algorítmico para el constructor y los tres métodos creados y descritos anteriormente.

2.1.1. Constructor `usuario(String nombre, String clave)`

Este constructor únicamente se compone de dos líneas, de las cuales dos se encargan de asignar los atributos `nombre` y `clave` propios de cada instancia de la clase `usuario`, de manera que suman dos unidades a la función de eficacia de este constructor. Por otro lado, se emplea un método propio de las tablas hash que es `put`, con el cual se agrega el usuario creado a la tabla hash que relaciona a todos los usuarios que tienen cuenta en la página X, y se sabe de la teoría vista en clase que esta tiene una complejidad constante en promedio. Así,

$$f(n) = 1 + 1 + 1$$
$$f(n) = 3$$

O(1) - constante

2.1.2. Creación de usuarios nuevos con `nuevoUsuario()`

Este método comienza con un mensaje que solicita al usuario el seudónimo bajo el cual va a realizar el registro y luego hay un input para dicho seudónimo, por lo que se comienzan con dos acciones que se realizan en tiempo constante. Enseguida se tiene el `if` que verifica que el seudónimo sea nuevo y en caso de que esto suceda se le informa esto al usuario, por lo que se realizan dos actividades en tiempo constante, pero se debe tener en cuenta que en el peor de los casos se ejecuta el `else` que se describirá a continuación, por lo que se suma solo una unidad a la función de eficiencia algorítmica. De esta manera, si se tiene al `else` que permite concluir el registro al requerir la contraseña al usuario, con lo cual se puede ejecutar el constructor descrito en el punto 2.1.1. y que tiene $f(n) = 3$, y se concluye con la impresión de tres cadenas de caracteres con información al usuario y una nueva línea para separar los textos, que se dan en tiempo constante. Con todo lo anterior, se llega a que la función de eficiencia algorítmica de este método está dada por:

$f(n) = 1 + 1 + 1 + 1 + 1 + 3 + 1 + 1 + 1$

$f(n) = 11$

O(1) - constante

2.1.3. Logueo de usuarios con `logueo()`

El método `logueo()` método tiene inicialmente dos impresiones por consola seguidas cada una por dos solicitudes de input que corresponden al usuario el primero y a la contraseña el segundo, los cuales tienen complejidad constante. Enseguida se tiene al `if` de verificación de existencia del seudónimo ingresado. En el peor de los casos, este `if` retorna un `false` y se debe continuar al siguiente if que chequea si la contraseña ingresada coincide con la ingresada inicialmente por el usuario cuando realizó su registro y que fue almacenada en la tabla hash, para lo cual se apela al método `get`, propio de los HashMaps en Java y que según lo viste en clase tiene complejidad constante. Entonces, de suceder esto, se imprime el mensaje por pantalla de que el logueo fue exitoso y, de no ser así, se informa que la clave es incorrecta. Se debe notar que sin importar si la contraseña es correcta o no. Así, la función de complejidad está dada por:

$f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1$

$f(n) = 7$

o(1) - constante

2.1.3. Usuarios registrados y sus contraseñas con `baseDatos()`.

Este método, que como ya se explicó anteriormente está inicialmente oculto al usuario por el menú principal simplemente se encarga de iterar a lo largo de la tabla hash para retornar todos los métodos que se tienen. En cada ciclo se obtiene el valor de la clave (seudónimo) y el valor asociado (contraseña), que se consigue en tiempo constante, por lo que se aporta un *2n* a la función de eficiencia algorítmica. Además, se finaliza con una impresión de una nueva línea por consola, lo cual es constante. De aquí se sigue que:

$f(n) = 2n + 1$

O(n) - lineal

De lo anterior es importante notar que salvo la función `baseDatos()`, que es meramente orientativa, se tienen **eficiencias algorítmicas constantes**, lo cual es sumamente importante para la página web X, pues si se utilizara algún otro tipo de estructura de datos le va a resultar complejo poder acceder a los datos de sus usuarios de forma ágil a medida que la página se haga más popular y más usuarios se registren en ella, lo cual puede desmotivar a muchos de ellos a seguir usándola, lo cual podría tener un perjuicio económico para la empresa (y de reputación para el programador). Es con esto que se evidencia la gran importancia y utilidad que tienen las tablas hash en aplicaciones básicas como la presentada.

3. Referencias

[1] R. Sedgewick y K. Wayne, *Algorithms*. Boston, MA, Estados Unidos de América: Addison Wesley, 2011.

[2] A. Shekhar, "Applications of Hash Table", *AfterAcademy*, 25-sep-2020. [En línea]. Disponible en: https://afteracademy.com/blog/applications-of-hash-table. [Accedido: 02-sep-2020].

[3] J. Jovanovic, "Shazam! Reconomiento de algoritmos de música, huellas dactilares y procesamiento", *Developers*. [En línea]. Disponible en: https://www.toptal.com/algorithms/shazam-reconocimiento-de-algoritmos-de-musica-huellas-dactilares-y-procesamiento. [Accedido: 02-sep-2021].

[4] TutorialPoint, "Compiler design. Symble table", *TutorialsPoint*, 27-dic-2014. [En línea]. Disponible en: https://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm. [Accedido: 03-sep-2021].

[5] B. Shriram Krishnamurthi, "Objects: Interpretation and types", *Programming and programming languages*, 31-dic-2016. [En línea]. Disponible en: https://papl.cs.brown.edu/2016/objects.html. [Accedido: 12-sep-2021].

[6] L. Chen, "Algorithms 7 - Hash/Cache and Memory", *Code by Case*, 26-ene-2021. [En línea]. Disponible en: https://www.codebycase.com/algorithms/a07-hash-cache-memory.html. [Accedido: 12-sep-2021].