<https://www.youtube.com/watch?v=tjtFkT97Xmc>

Una tabla Hash combina la habilidad de acceso aleatorio a los datos con el dinamismo de las listas enlazadas.

Esto significa que un buen diseño e implementación de una tabla hash conlleva a los siguientes resultados:

* La inserción de datos tiende a O(1)
* La eliminación de datos tiende a O(1)
* La consulta de datos en la tabla tiende a O(1)

Para obtener una mejor eficiencia, se crea una estructura de datos cuando al insertar datos en la estructura (Tabla Hash), **los mismos datos de entrada nos dan una pista en donde podemos encontrar los datos en sí mismo**, para luego ser consultada.

Una de las desventajas de las Tablas Hash es que no son eficientes al momento de ordenar los datos que ellas mismas contienen. Las Tablas Hash deben ser usadas si y solo si, en el planteamiento del problema en cuestión, no es de sumo interés que los datos en sí mismos, se encuentren ordenados dentro de la propia estructura de datos.

Una Tabla Hash está conformada por dos elementos esenciales:

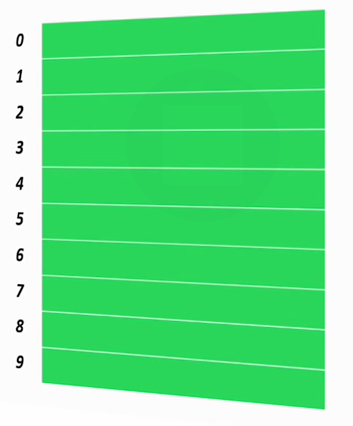
* Primeramente, una función hash, la cual retorna un numero entero no negativo llamado “código hash”
* Segundo, un arreglo capaz de guardar datos del tipo de datos especifico que deseamos guardar en la estructura de datos (ej: cadenas de caracteres)

La idea esencial de una Tabla Hash es que, en base a los datos de entrada, generamos un código hash, y luego guardar esa pieza de información (datos de entrada) en la posición del arreglo que representa el código hash generado con anterioridad.

Ejemplo:

Imagínese una Tabla Hash capaz de guardar 10 elementos,

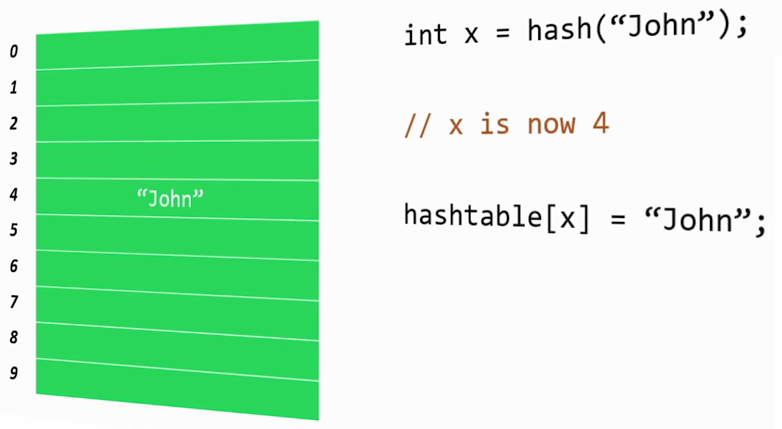
**string hashtable[10]**:



Y se tiene como datos de entrada, la cadena “John”, la cual es pasada como parámetro a una función hash (generadora de códigos hash):

**int x = hash(“John”);**

Siendo “John”, la cadena de caracteres que deseamos insertar en la Tabla hash, y supóngase, que el valor entero devuelto por la función hash para la cadena de caracteres “John”, es de x=4, esto significa que, la cadena de caracteres será guardada en la posición 4 de la Tabla Hash.



**¿Qué sucede si deseamos saber la posición en la cual “John” se encuentra en la Tabla Hash?**

Es en este momento donde se presenta la eficiencia de la estructura de datos conocida como “Tabla Hash”, para conocer la posición en la cual “John” se encuentra en la tabla, solo es necesario ejecutar la función hash pasando como parámetro la cadena “John”, y como resultado obtendremos su código hash, el numero entero que indica donde se encuentra dentro de la tabla. De este modo se evita recorrer cada uno de los elementos para conocer la posición de la cadena.

**¿Cómo definir una función hash (propiedades de una función hash)?**

Una función hash debería:

* No tener límites respecto a la cantidad de códigos hash que pueda generar esta función, lo que significa que, basándonos en el ejemplo anterior, toda cadena de caracteres que reciba como parámetro, debería generar como resultado un código hash
* Usar solo los datos que se han enviado como parámetros a la función hash
* Usar todos los datos (ej: toda la cadena de caracteres) a ser codificada
* Ser determinístico, lo que implica que, cuando se envíe como parámetro la misma cadena de caracteres, se debe de obtener siempre el mismo código hash.
* Los datos deben ser distribuidos uniformemente
* Generar códigos hash completamente distintos para datos de entrada que son similares (ej: “John” y “John1”)

**Ejemplo básico de una función hash**

unsigned int hash(char \*str){

int suma = 0;

for (int j=0; str[j] != ‘\0’; j++){

suma += (int)str[j];

}

return suma % HASH\_MAXIMOS;

}

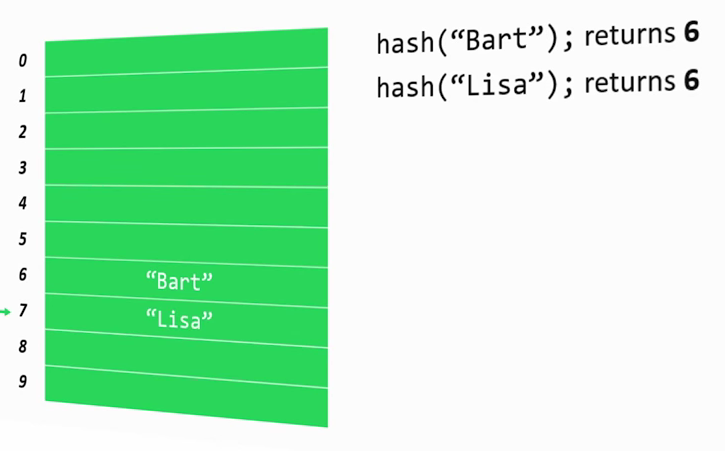
**Problemas comunes en las Tablas Hash:**

Una colisión ocurre cuando, dos piezas de datos, al ser pasadas a la función hash, retornan el mismo código hash aun cuando estas son distintas. Este caso debe ser evitado para no sobrescribir en posiciones que ya están siendo ocupadas por otros elementos.

La manera de resolver colisiones es por medio de “Linear Probing” (sondeo lineal). Por medio de este método, si se tiene una colisión, se intenta guardar el dato en la siguiente posición, y en caso de que esta se encuentre ocupada también por otro elemento, se mueve al siguiente y así sucesivamente hasta encontrar un índice vacío que pueda ocupar ese valor de entrada. De este modo, si no encontrásemos el valor en la posición indicada por su código hash generado por la función hash, al menos este dato se encontrará cerca de ese código hash.

Ejemplo:

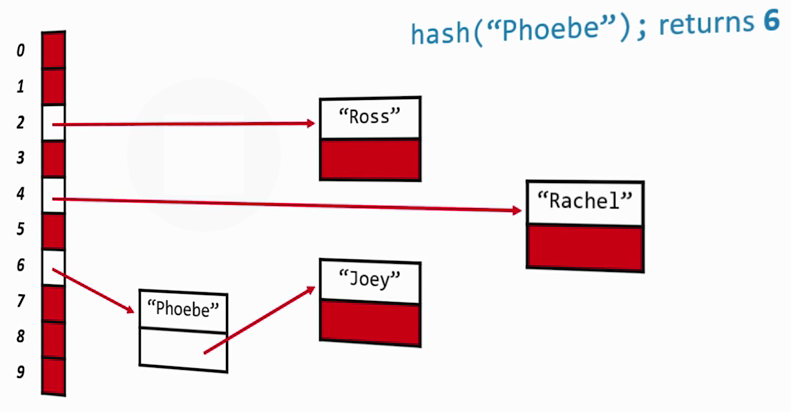
Supóngase que la función hash para la cadena de caracteres “Bart” devuelve un valor entero de x=6, lo que significa que se guardara en la posición 6 de la Tabla Hash. Entonces se vuelve a ejecutar la función hash para la cadena de caracteres “Lisa” y esta devuelve también el valor de x=6, lo que representa una colisión que será resuelta por medio de sondeo lineal, moviéndose a la siguiente posición que se encuentra debajo de “Bart” y guardando allí el valor de “Lisa” ya que no existe ningún otro elemento usando esta posición.



El sondeo lineal puede representar un problema a futuro ya que, volviendo al caso de que la función hash retorne repetidamente el valor de x=6, implica que el nuevo elemento a agregar a la Tabla Hash poco a poco, se encontrara más lejos de la posición original x=6 porque habrá más elementos que ya estén ocupando las posiciones contiguas a 6 (generando algo llamado “clustering”, agrupamiento), mientras, basándonos en el ejemplo anterior, aún existe el problema de que gracias a las colisiones entre valores de la Tabla Hash, se intente agregar el nuevo valor a una posición que no exista dentro de la tabla (que en esencia es, por el momento un arreglo de datos del tipo string).

Para resolver el problema de espacio y colisiones, en vez de mover el nuevo elemento que colisiona con otro elemento que ya se encuentra en esa posición, se usa listas enlazadas basándonos en la idea de que, en vez de guardar solo un elemento en esa posición, este elemento sea un puntero a una lista enlazada que permita guardar múltiples elementos en esa posición (por medio de la lista) representada por el código hash.

Retomando la idea anterior que las Tablas Hash son eficientes al momento de insertar, eliminar, buscar datos, es gracias a la combinación código hash + listas enlazadas ya que, si múltiples elementos ocupan la misma posición en la tabla hash, solo se necita recorrer la lista enlazada guardada en esa posición para poder encontrar el valor original de este modo, sin necesitar recorrer cada elemento de la tabla hash, ni cada elemento de todas las listas enlazadas que guarda esa tabla.

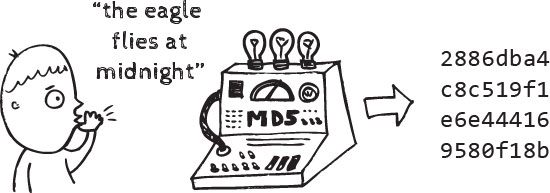


**Funciones Hash criptográficas, contraseñas y seguridad**

[**https://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/**](https://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/)

Anteriormente, el concepto de función hash fue definida como una función que dado datos de entrada (en este caso cadenas de caracteres), generaba un número entero no negativo que no solo servía para identificar la posición en la cual dicho elemento se encontraba en la tabla, sino que esta función retornaba el mismo valor entero para el mismo valor pasado como parámetro.





Pero qué ocurre si la información que será guardada en la tabla hash es información sensible, ya sea contraseña de cuentas de usuarios, números de tarjeta de crédito, etc. Si la información se guarda en la tabla en texto plano, esto implica un gran riesgo para la seguridad de los usuarios ya que, en caso de ser robada la información de la tabla, esta tabla se encuentra en un lenguaje visible para el atacante, lo que implica que, con facilidad podría saber las contraseñas originales sin ningún problema ya que son legibles para los humanos. En base a esta problemática surgen las funciones hash criptográficas cuyo objetivo no solo es retornar un identificador para esa cadena de texto, sino que también “obscurecer” la cadena original para que no pueda ser leída con facilidad por los humanos. Lo que significa que, tomando como ejemplo la cadena “John”, por medio de funciones hash criptográficas, seria traducido a:

**John = 61409aa1fd47d4a5332de23cbf59a36f**

Supóngase que los valores retornados por esta función hash son guardados como las “contraseñas” en una tabla de usuarios en vez de la cadena original que es la contraseña en si misma del usuario por motivos de seguridad en casos de ataque. ¿Cómo un usuario puede autenticarse al sistema?

Tomemos el siguiente caso de ejemplo:

El usuario John, tiene una cuenta en Facebook, siendo la contraseña de John **“password123”**. Usando una función hash criptográfica, la cadena “password123” seria traducido a:

**password123 = 482c811da5d5b4bc6d497ffa98491e38**

Siendo la cadena al lado derecho del signo = la que es guardada en la tabla de contraseñas de Facebook. Ahora el usuario John desea iniciar sesión con su cuenta de Facebook, ¿Cómo John se autentica en el sistema si lo guardado por Facebook es **482c811da5d5b4bc6d497ffa98491e38** y no **password123**?

Recordando una de las propiedades básicas de las funciones hash es que sea determinístico, lo que significa que, si se introduce la misma cadena de caracteres la función debería retornar siempre el mismo resultado. John por error escribe **password12345** como contraseña, entonces Facebook, envía como parámetro a la función hash criptográfica password12345, y esta cadena es traducida a:

**password12345 = 365d38c60c4e98ca5ca6dbc02d396e53**

Este valor es comparado con el valor guardado en la tabla de contraseñas, siendo ambas cadenas diferentes, lo que implica que John no ha escrito correctamente su contraseña:

**482c811da5d5b4bc6d497ffa98491e38 ¡= 365d38c60c4e98ca5ca6dbc02d396e53**

Ahora John, percatándose de su error y la alerta de Facebook de que la contraseña era inválida, procede a escribir su verdadera contraseña, password123, y este es encriptado y comparada con la contraseña guardada por Facebook, siendo ambas cadenas iguales, la introducida por el usuario y la guardada por Facebook, logrando iniciar sesión con éxito a su cuenta.

**365d38c60c4e98ca5ca6dbc02d396e53 = 365d38c60c4e98ca5ca6dbc02d396e53**

En base al ejemplo anterior, se comprueba la importancia de las funciones hash criptográficas en el guardado de contraseñas ya que:

* Permiten obscurecer la información que fue originalmente introducida por el usuario, de este modo siendo imposible para un atacante conocer la contraseña original a partir de la cadena de caracteres generada por la función.
* Permite el acceso al sistema si y solo si el usuario proporciona la cadena original que da como resultado la cadena encriptada en la tabla, que en este caso, representa la contraseña original del usuario solo que escrita de manera “no legible”

Una de las reglas de oro de las funciones has criptográficas es que estas funciones deben de ser programadas de tal modo que, la cadena retornada por esta tiene que ser irreversible, lo que significa que a partir de su resultado es imposible deducir la cadena original que genero dicha salida, de este modo garantizando para el ejemplo de las contraseñas, que es imposible conocer la contraseña original.

**Integridad de los datos**

[**https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.110).aspx**](https://msdn.microsoft.com/en-us/library/f9ax34y5(v=vs.110).aspx)

[**http://www.makeuseof.com/tag/md5-hash-stuff-means-technology-explained/**](http://www.makeuseof.com/tag/md5-hash-stuff-means-technology-explained/)

[**http://www.guidingtech.com/9800/what-is-md5-checksum-how-to-verify-it/**](http://www.guidingtech.com/9800/what-is-md5-checksum-how-to-verify-it/)

Integridad de los datos se refiere a la precisión y consistencia (validez) de los datos a través de su ciclo de vida de uso. La integridad de los datos puede ser comprometida a través de muchas maneras, ya sea por errores de hardware, errores de software, actividades maliciosas, o problemas en el envío de la información a través de redes/Internet, siendo el caso de interés para esta investigación, el último problema antes mencionado.

¿Cómo asegurar la integridad de los datos en el envío de información?

Cada vez que un usuario envía un archivo a través de una red o Internet, puede que la otra persona con el mismo archivo que el usuario original envió, existiendo la posibilidad de corrupción en los datos.

Las funciones hash criptográficas juegan un rol importante en asegurar la integridad de los datos que posee un archivo, basándose en la idea de que, dos archivos que no sean exactamente iguales no pueden tener el mismo valor hash generado por una función de este tipo. Supóngase que se desea enviar la siguiente imagen a través de internet a un cliente:



Basándose en el hecho que, las funciones hash, dado datos de entrada, generan un número o cadena de caracteres asociada a esa entrada, se puede usar las funciones hash criptográficas como “huellas digitales” que nos permitan verificar la integridad de un archivo enviado, sabiendo el hash adjunto de ese archivo, tomando como ejemplo la imagen anterior, si los datos que conforman la imagen fuesen enviados a la función, generaría el siguiente valor:

**5eb63bbbe01eeed093cb22bb8f5acdc3**

La imagen anterior es enviada al usuario final como también el hash asociado a esa imagen para futura verificación de integridad, pero por motivos externos al momento de transmitir la imagen al usuario, este termina con el siguiente archivo:



Entonces, el usuario final, por medio del hash enviado junto a esa imagen, procede a verificar si el archivo enviado es exactamente el mismo que fue enviado por medio de una función hash que reciba como entrada los datos que conforman la imagen recibida, y descubre que el valor generado es el siguiente:

**258fc86eae1446507da87a5d8abde1c4**

Al ser comparado con el hash enviado, se comprueba que ambos hash no son iguales, lo que significa que, la imagen recibida no es la misma que la enviada:

**5eb63bbbe01eeed093cb22bb8f5acdc3 ¡= 258fc86eae1446507da87a5d8abde1c4**

Comprobándose de este modo que, una función hash criptográfica puede servir como medio de seguridad que nos permita:

* Verificar si los archivos enviados a través de una red han llegado a su destino sin ser comprometidos o modificados en todo el trayecto recorrido hacia su destino
* Comprobar que no existieron perdidas de información ya sea por problemas de hardware, bugs en el software usado para la transmisión de información
* Sin importar cuan pequeña o grande sea la modificación que haya recibido el archivo a través de su recorrido, gracias a la regla de las funciones hash que generan valores completamente distintos aun cuando su entrada es muy parecida, es posible verificar con exactitud que el archivo es el mismo o es distinto del original