

**Ingeniería en Ciencias de la Computación**

**Programación**

Exposición: Hashing

Tercer Parcial

|  |  |
| --- | --- |
| **Equipo:** | **Nombre del Maestro:** |
| 5 | Ricardo Blanco Vega |
| **Matrícula:** | **Nombre del Alumno:** |
| 338832 | Juan Angel Cepeda Fernández |
| 338729 | Ivan Bejarano Chaine |
| 339028 | Placido Alberto Gonzalez Castillo |
| 338856 | Sebastián Martínez Ávila |

Fecha de entrega: **11/11/2019**

**Tabla de contenidos.**

[**Objetivo**](#_g6i1vya7cpgf) **3**

[**Desarrollo**](#_hf81d4ithxbm) **3**

[**Qué es el hashing?**](#_w471thqv48ql) **3**

[Transformación de claves](#_upmchijvavte) 4

[Funciones Hash](#_48q1o6cdzmui) 5

[Solución de colisiones](#_sneldu2ccm7p) 7

[Propiedades](#_9yxxhinwdhiz) 8

[Ejemplos de Hashing](#_5cajre54ydou) 9

[Aplicaciones](#_5zngqcfpt85k) 9

[**Práctica guiada**](#_th66tmpy5zqz) **10**

[**Conclusiones**](#_idzjaxil7n9c) **15**

[**Recomendaciones**](#_7py6qm108hkg) **15**

[**Referencias**](#_etgq7slk4k4n) **15**

### Objetivo

Exponer a los compañeros del grupo de introducción a la programación qué es, cómo se usa y para qué sirve el hashing dentro de los ambientes de programación.

Proporcionar una práctica guiada para facilitar la comprensión del hasing e involucrar a los compañeros en el tema.

### Desarrollo

#### Qué es el hashing?

A las funciones resumen también se les llama funciones hash o funciones digest. Una función hash H es una función computable mediante un algoritmo tal que:

**H: U -> M**

**x -> h(x)**

Tiene como entrada un conjunto de elementos, que suelen ser cadenas, y los convierte en un rango de salida finito, normalmente cadenas de longitud fija. Es decir, la función actúa como una proyección del conjunto U sobre el conjunto M.

Hay que tener en cuenta que M puede ser un conjunto definido de enteros. En este caso podemos considerar que la longitud es fija si el conjunto es un rango de números de enteros ya que podemos considerar que la longitud fija es la del número con mayor número de cifras. Todos los números se pueden convertir al número especificado de cifras simplemente anteponiendo ceros.

Normalmente el conjunto U tiene un número elevado de elementos y M es un conjunto de cadenas con un número más o menos pequeño de símbolos. La idea básica de un valor hash es que sirva como una representación compacta de la cadena de entrada.

Por esta razón se dice que estas funciones resumen datos del conjunto dominio.

#### **Transformación de claves**

Una tabla hash es una colección de ítems que se almacenan de tal manera que sea más fácil encontrarlos más tarde. Cada posición de la tabla hash, a menudo llamada una ranura, puede contener un ítem y se llama por un valor entero comenzando en 0. Por ejemplo, tendremos una ranura llamada 0, una ranura llamada 1, una ranura llamada 2, y así sucesivamente. Inicialmente, la tabla hash no contiene ningún ítem por lo que cada ranura está vacía.

La Figura muestra una tabla hash de tamaño de la tabla *m*=11

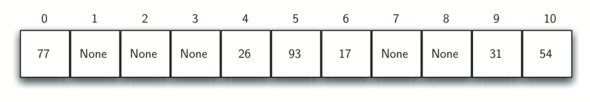
En otras palabras, hay *m* ranuras en la tabla, con nombres de 0 a 10.

### 

La correspondencia entre un ítem y la ranura a donde pertenece ese ítem en la tabla hash se denomina la función hash. La función hash tomará cualquier ítem de la colección y devolverá un número entero en el rango de nombres de las ranuras, entre 0 y m-1. Supongamos que tenemos el conjunto de ítems enteros 54, 26, 93, 17, 77 y 31. Nuestra primera función hash, a veces denominada “método del residuo”, simplemente toma un ítem y lo divide por el tamaño de la tabla, devolviendo el residuo como su valor hash ( h(item)=item%11 ).

La Tabla da todos los valores hash para nuestros ítems de ejemplo. Tenga en cuenta que este método del residuo (módulo aritmético) estará típicamente presente en alguna forma en todas las funciones hash, ya que el resultado debe estar en el rango de nombres de las ranura*s.*

Ahora, cuando queramos buscar un ítem, simplemente usamos la función hash para calcular el nombre de la ranura para el ítem y luego verificamos la tabla hash para ver si está presente. Esta operación de búsqueda es O(1), ya que se requiere una cantidad de tiempo constante para calcular el valor hash y luego indexar la tabla hash en esa ubicación. Si todo está donde debería estar, hemos encontrado un algoritmo de búsqueda de tiempo constante.

**

Esta técnica sólo va a funcionar si a cada ítem le corresponde una ubicación exclusiva en la tabla hash. Por ejemplo, si el ítem 44 hubiera sido el siguiente ítem de nuestra colección, tendría un valor hash de 0 ( 44%11=0). Dado que 77 también tenía un valor hash de 0, tendríamos un problema. Según la función hash, dos o más ítems necesitan estar en la misma ranura.

Esto se conoce como colisión (también se puede llamar un “choque”). Evidentemente, las colisiones crean un problema para la técn*ica de* búsqueda por transformación de claves*.*

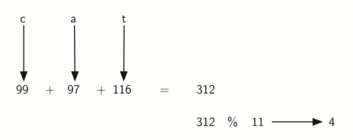
#### **Funciones Hash**

Dada una colección de ítems, una función hash que asigna cada ítem en una ranura única se conoce como una función hash perfecta. Si conociéramos los ítems y la colección nunca cambiara, entonces sería posible construir una función hash perfecta. Desafortunadamente, dada una colección arbitraria de ítems, no existe una forma sistemática de construir una función hash perfecta.

Una forma de obtener una función hash perfecta es aumentar el número de ranuras dentro de la tabla, pero esto gastaría una enorme cantidad de recursos informáticos si los datos a almacenar fueran matrículas de 50 estudiantes o de números de registros poblacionales.

El objetivos es crear una función hash que disminuya significativamente el número de colisiones, sea fácil de calcular y pueda distribuir la cantidad de datos uniformemente dentro de las tablas hash.

Existen varios métodos para ello, uno es el método de plegado que consiste en sumar las partes del mensaje luego utilizar el método del residuo. el segundo método sería el método del cuadrado que consiste en elevar un número al cuadrado, extraer dígitos dentro de su cuadrado y luego aplicar el método del residuo.

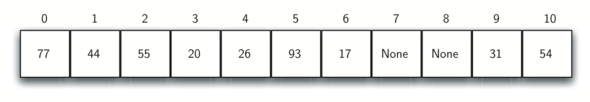


En la imagen anterior representa una cadena de caracteres “cat” se separan las letras y se les da el valor que poseen en el código ASCII, sus valores se suman y nos da como resultado 312, aplicando el método del residuo en una tabla con 11 ranuras a la cadena “Cat” le corresponde la ranura 4 o el valor hash 4.

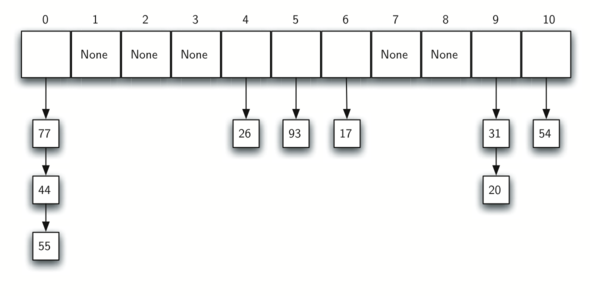
#### Solución de colisiones

Ahora regresamos al problema de las colisiones. Cuando a dos ítems se les asigna la misma ranura, debemos contar con un método sistemático para colocar el segundo ítem en la tabla hash. Este proceso se denomina solución de colisiones. Como dijimos anteriormente, si la función hash es perfecta, nunca se producirán colisiones. Sin embargo, como esto no suele ser posible, la solución de colisiones se convierte en una parte muy importante de la transformación de claves.

Un método para solucionar las colisiones examina la tabla hash e intenta encontrar otra ranura disponible para contener el ítem que causó la colisión. Una manera sencilla de hacerlo es comenzar en la posición del valor hash original y luego moverse secuencialmente a lo largo de las ranuras hasta encontrar la primera que esté vacía. Tenga en cuenta que es posible que necesite volver a la primera ranura (circularmente) para cubrir toda la tabla hash. Este proceso de solución de colisiones se conoce como direccionamiento abierto ya que intenta encontrar la siguiente ranura o dirección disponible (“abierta”) en la tabla hash. Al visitar sistemáticamente cada una de las ranuras, de una en una, estamos ejecutando una técnica de direccionamiento abierto llamada prueba línea.



Encadenamiento



#### Propiedades

**Bajo Costo**

Calcular el valor hash es de bajo costo en recursos computacionales y en almacenamiento

**Compresión**

Una función *hash* comprime datos si puede mapear un dominio con datos de longitud muy grande a datos con longitud más pequeña

**Blindaje contra colisiones**

Las funciones hash deben de tener un proceso por el cual eviten colisiones de datos

**Asociativas**

#### 

#### **Ejemplos de Hashing**

Funciones hash distintas producen outputs o salidas que difieren de la extensión, pero la salida siempre será la misma, será constante, Por ejemplo el algoritmo SHA-256 solo podrá producir salidas de 256 bits, mientras que SHA-1 solo podrá producir una salida o digest de 160 bits.

Para ilustrarlo, procesaremos las palabras "Binance" y "binance" mediante el algoritmo hashing SHA-256 (que es el utilizado por Bitcoin)

|  |  |
| --- | --- |
| Input | Output (256 bits) |
| Binance | f1624fcc63b615ac0e95daf9ab78434ec2e8ffe402144dc631b055f711225191 |
| binance | 59bba357145ca539dcd1ac957abc1ec5833319ddcae7f5e8b5da0c36624784b2 |

Véase como un cambio menor (la mayúscula de la primera letra) produce como resultado un valor de hash radicalmente distinto. Pero dado que estamos empleando el SHA-256, los outputs tendrán siempre una extensión fija de 256 bits (o 64 caracteres) -independientemente del tamaño del input. Asimismo, no importará la cantidad de veces que se procesen las dos palabras a través del algoritmo, los dos outputs se mantendrán inalterables.

#### **Aplicaciones**

Las funciones hash convencionales presentan una amplia variedad de casos de uso, entre los que se incluyen lookups de bases de datos, análisis de grandes archivos y gestión de datos. Por el contrario, las funciones hash criptográficas se utilizan ampliamente en aplicaciones de seguridad de la información, tales como la autenticación de mensajes y las huellas digitales. En lo que respecta a Bitcoin, las funciones hash criptográficas constituyen una parte esencial del proceso de minado, y también juegan un papel destacado en la generación de nuevas direcciones y claves.

Pero el auténtico potencial del hashing se revela cuando es necesario lidiar con cantidades enormes de información. Por ejemplo, es posible procesar un archivo grande o un conjunto de datos a través de una función hash, y a continuación utilizar su output para rápidamente verificar la exactitud e integridad de los datos. Esto es posible debido a la naturaleza determinística de las funciones hash: el input producirá siempre un output simplificado y condensado (hash). Dicha técnica elimina la necesidad de almacenar y recordar grandes cantidades de datos.

El hashing es particularmente útil en el contexto de la tecnología blockchain. La blockchain de Bitcoin lleva a cabo diversas operaciones que conllevan hashing, la mayoría de ellas en el marco del proceso de minado. De hecho, casi todos los protocolos de criptomonedas dependen del hashing para condensar grupos de transacciones en bloques, así como para producir enlaces criptográficos entre cada uno de estos últimos -lo que, de forma efectiva, acaba generando una blockchain

### Práctica guiada

#include<stdlib.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define size 100

//Estructura que contendra cada celda de la tabla guardando los datos del

//estudiante que se introduciran

// por medio del teclado

struct infoalumno

{

char nombre[50];

char carnet[10];

char telefono[9];

char direccion[50];

int esAlta;

};

typedef struct infoalumno alumno;

//Declaracion de la tabla hash

struct tipotabla

{

alumno \*tabla[size];

int elementos;

double factorcarga;

//declaramos un factor de carga puesto que una tabla hash se vuelve ineficiente cuando sobrepasa el 50%

};

typedef struct tipotabla tabladispersa;

// Declaracion de funciones

void CrearTabla(tabladispersa \*);

void insertar(tabladispersa\*, alumno);

alumno \*buscar(tabladispersa\*, char \*);

int eliminar(tabladispersa \*, char \*);

long transforma(char \*);

int direccion(tabladispersa \*, char \*);

void flush();

main()

{

int op, res, i, p;

char clave[50];

alumno datos, \*prt;

tabladispersa tabla[size], n;

CrearTabla(tabla);

system("color 05");

system("cls");

while(1)

{

printf("MENU");

printf("Seleccione la operacion a realizar:");

printf("1.Insertar Elementont2.Buscar un elemento");

printf("3.Eliminar un elementont-Salirnt");

scanf("%d",&op);

switch(op)

{

case 1:

system("cls");

puts("nNombre del estudiante");

flush();

gets(datos.nombre);

puts("nCarnet del estudiante:");

flush();

gets(datos.carnet);

puts("nNo de telefono del estudiante");

flush();

gets(datos.telefono);

puts("nDireccion del estudiante:");

flush();

gets(datos.direccion);

p = direccion(tabla , datos.carnet);

printf("nnLa clave hash generada es: %dn",p);

puts("nInsercion exitosa!n");

insertar(tabla, datos);

break;

case 2:

system("cls");

puts("nIntroduzca el carnet del estudiante que desea buscar");

flush();

scanf("%s",&clave);

prt = buscar(tabla, clave);

if(prt)

{

system("cls");

puts("ntDatos del estudiante:n");

printf("nNombre: %sn",prt->nombre);

printf("nCarnet %sn",prt->carnet);

printf("nNo de telefono %sn",prt->telefono);

printf("nDireccion: %sn",prt->direccion);

}

else

puts("nNo existe ese estudiante en este registronn");

break;

case 3:

system("cls");

puts("nDigite el dato a eliminar de la tabla:");

flush();

scanf("%s",&clave);

i = eliminar(tabla, clave);

if(i!=1)

puts("nEliminacion con exiton");

else

puts("nError en la eliminacionn");

break;

case 4:

exit(0);

break;

}

}

getch();

return 0;

}

// Crea una tabla hash para trabajar

void CrearTabla(tabladispersa \*t)

{

int j;

for(j=0;j<size;j++)

{

t->tabla[j] == NULL;

}

t->elementos = 0;

t->factorcarga = 0.0;

}

// transforma los caracteres de la clave en valores enteros

long transforma(char \*clave)

{

int j;

long d = 0;

for(j=0;j<strlen(clave);j++)

{

d = d \* 27 + clave[j];

}

return ((d>=0) ? d: -d);

// dirección recibe la tabladispersa y la clave para colocar esta ultima en la tabla

}

int direccion(tabladispersa \*t, char \*clave)

{

int i=0;

long p, d;

d = transforma(clave);

p = d % size;

while(t->tabla[p] != NULL && strcmp(t->tabla[p]->carnet, clave) !=0)

{

i++;

p = p + i \* i;

p = p % size;

}

return (int)p;

}

// Inserta los datos que representa la clave en la tabla hash

void insertar(tabladispersa \*t, alumno r)

{

alumno \*pr;

int posicion;

pr = (alumno\*)malloc(sizeof(alumno));

strcpy(pr->nombre, r.nombre);

strcpy(pr->carnet, r.carnet);

strcpy(pr->telefono, r.telefono);

strcpy(pr->direccion, r.direccion);

pr->esAlta = 1;

posicion = direccion(t, r.carnet);

t->tabla[posicion] = pr;

t->elementos++;

t->factorcarga=(t->elementos)/size;

if(t->factorcarga > 0.5)

{

puts("nFactor de Carga supera el 50% de la tabla");

}

}

// Busca el elemento en la tabla e imprime si lo encuentra o no

alumno \*buscar(tabladispersa \*t, char \*clave)

{

alumno \*pr;

int posicion;

posicion = direccion(t, clave);

pr = t->tabla[posicion];

if(pr != NULL)

{

if(!(pr->esAlta))

{

pr = NULL;

}

}

return pr;

}

// Elimina el elemento de la tabla hash

int eliminar(tabladispersa \*t, char \* clave)

{

int posicion;

posicion = direccion(t, clave);

if(t->tabla[posicion] !=NULL)

{

t->tabla[posicion] -> esAlta = 0;

}

else

return 1;

}

void flush()

{

fflush(stdin);

fflush(stdout);

}

### Conclusiones

Como conclusión podemos ver que las funciones hash sirven para representar la huella digital de distintos documentos, haciendo cada uno de estos únicos. También pueden utilizarse las funciones hash para temas de criptografía o seguridad.

### Recomendaciones

* Tomar en cuenta los caracteres introducidos, por ejemplo el código de “Brian” es 75c450c3f963befb912ee79f0b63e563652780f0, mientras qué el codigo de “Brain” es 8b9248a4e0b64bbccf82e7723a3734279bf9bbc4, esto quiere decir que al generar un código hash cualquier cambio por más minúsculo que sea puede devolver un hash completamente diferente.

### Referencias

Chapman, S. J. (2018). *Fortran for Scientists and Engineers.* Mcgraw-Hill.