UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

Practica 1

Técnicas de programación avanzada
Docente. Borja Monsalve Piqueras

Gerónimo Basso Sosa
Ángel Esquinas Puig

CURSO 2022-2023

Índice

Introducción

Debemos realizar un programa en java, con una clase ya definidas las cuales son Hash, Celda y Pruebas, todo ello para realizar la programación de nuestra propio tabla hash, donde nos encontraremos el estado, la clave y el valor. Para su realización utilizaremos direccionamiento abierto con doble hashing para la resolución de colisiones.

Marco teórico

Las tablas hash son estructuras de datos que se utilizan para almacenar un número elevado de datos sobre los que se necesitan operaciones de búsqueda e inserción muy eficientes. Una tabla hash almacena un conjunto de pares "(clave, valor)". La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor.

Un diccionario es un ejemplo de estructura que se puede implementar mediante una tabla hash. Para cada par, la clave es la palabra por buscar, y el valor contiene su significado. El uso de esta estructura de datos es tan común en el desarrollo de aplicaciones que algunos lenguajes las incluyen como tipos básicos.

La implementación de una tabla hash está basada en los siguientes elementos:

- Una tabla de un tamaño razonable para almacenar los pares (clave, valor).
- Una función "hash" que recibe la clave y devuelve un índice para acceder a una posición de la tabla.
- Un procedimiento para tratar los casos en los que la función anterior devuelve el mismo índice para dos claves distintas. Esta situación se conoce con el nombre de colisión.

El tamaño de una tabla hash es el otro parámetro que debe ser ajustado con cierto cuidado. Al igual que en el caso de la función de hash, se puede analizar cuáles son las consecuencias extremas de la elección de un tamaño de tabla inadecuado. Por ejemplo, si el tamaño es demasiado pequeño, digamos 1, de nuevo la tabla se comporta como una lista encadenada. En cambio, si el tamaño es enorme, el malgasto de memoria es evidente, puesto que habrá un elevado número de posiciones cuya lista de colisiones está vacía.

Cuando se utilizan listas de colisión se suele utilizar una técnica basada en un "factor de carga". Por factor de carga se entiende el valor al dividir el número de elementos en la tabla y su número de entradas. Si la función de hash produce una distribución uniforme de los elementos, un valor elevado del factor de carga denota una longitud excesiva de las listas de colisiones. Un valor de factor de carga muy reducido denota una longitud muy corta de las cadenas de colisión y por tanto una infrautilización de memoria.

Las implementaciones más complejas de tablas hash incluyen dos umbrales para el valor del factor de carga. Cada operación de inserción o borrado de un elemento consulta estos valores. Si el factor de carga supera el valor máximo, se reserva espacio para una tabla mayor, se borran todos los elementos de la tabla antigua y se insertan en la nueva (con factor de carga obviamente menor). Análogamente, si al borrar un elemento el factor de

carga es menor que el umbral inferior, se realiza una operación similar, pero con una nueva tabla más pequeña. De esta forma se consigue mantener de una forma transparente al exterior, el valor del factor de carga en límites razonables. La operación de redimensionado de la tabla es cara en tiempo de CPU, pero se supone que las operaciones futuras de búsqueda, al encontrar listas de colisión de la dimensión adecuada, rentabilizan su ejecución. (Madrid, 2023)

Descripción del código

Clase Hash

La clase hash es una clase creada con el objetivo de representar una tabla hash. Definiremos varios atributos dentro de esta, estos son:

- **contenedor: Celda<Valor>[]** donde almacenaremos el contenido de todas las celdas.
- **numElementos: int** variable que especifica la cantidad de elementos que hay dentro del contenedor, es importante diferenciar que numElementos no es el tamaño de la tabla, sino la cantidad de elementos que hay dentro de la tabla.
- **alfaMaximo: float** un valor entre cero y uno el cual establece hasta que porcentaje puede estar completa la tabla hash.

Dentro de la clase hash contábamos con varios métodos para poder emular el comportamiento de una tabla hash. Estos son:

Hash ()

Es uno de los varios constructores que hay dentro de la clase Hash, no tiene ningún parámetro de entrada, en caso de ser utilizado se inicializa una tabla hash con alfaMax 0.8 y tamaño 7.

```
public Hash(){
    this.contenedor = new Celda[7];
    this.alfaMaximo = 0.80f;
}
```

Hash (int capacidad)

Constructor de la clase hash que solo tiene como parámetro la capacidad de la tabla, el alfaMaximo será por defecto 0.8.

```
public Hash(int capacidad){
    this.alfaMaximo = 0.80f;
}
```

Hash (int capacidad, float alfaMax)

Constructor de la clase hash, tiene como parámetro la capacidad de la tabla, y el alfaMax.

```
public Hash(int capacidad, float alfaMax){
    this.contenedor = new Celda[capacidad];
    this.alfaMaximo = alfaMax;
}
```

insertar (int clave, Valor v): void

La función insertar representa todo lo que ocurre cuando queremos agregar un elemento a nuestra tabla hash. Es importante diferenciar dos casos en la función insertar, un insertar normal es aquel en donde puede haber colisiones, pero el tamaño de la tabla no cambia y un insertar con redimensionamiento es aquel en donde vamos a agrandar la tabla e insertar los elementos nuevamente.

Empezamos nuestro código calculando el factor de carga alfaAFuturo de la tabla hash, es importante calcular este alfa desde principio dado que nunca sabemos cuándo vamos a tener que hacer el redimensionamiento.
Definimos la variable "yaExiste" en falso, y continuamos verificando si la tabla hash no está vacía, se recorre cada celda del contenedor para verificar si la clave ya existe en la tabla. Si es así, se establece la variable "yaExiste" a verdadero y se muestra un mensaje de error, esto ocurre dado que no podemos insertar claves iguales en la tabla hash, por lo tanto, si ya tenemos la clave en nuestra tabla no haremos ningún insert.

Si "yaExiste" es falso, ósea si mi tabla no está vacía y tengo un elemento que quiero insertar verifico si el factor de carga alfaAFuturo supera el alfaMaximo. Si esto ocurre, llamo a la función "redimensionar" para realizar un redimensionamiento de mi tabla hash. Si no hay necesidad de redimensionamiento o si ya lo hice previamente, voy a insertar mi elemento, establecemos la variable "colisiones" a cero y calculamos el índice de la celda en la que se debe insertar el valor utilizando la función de hash "funcionHash" y el número de colisiones actual. Si hay una colisión en el índice calculado, se incrementa la variable "colisiones" y se recalcula el índice utilizando la función de hash.
Cuando se encuentra una celda vacía (sin colisiones) en el índice calculado, se crea una nueva celda con la clave y el valor y se establece su estado en 1 (significando celda ocupada). Por último, la celda se inserta en el contenedor en la posición del índice calculado y se incrementa el número de elementos.

```
public void insertar(int clave, Valor v){
    float alfaAFuturo =  ((float)( numElementos + 1 ) /  (float
)contenedor.length);
    boolean yaExiste = false;
    //Si la tabla hash no está vacía, comprobamos que el valor no
exista.
    if(numElementos >= 0){
        for (int i = 0; i < contenedor.length; i++) {
            if(contenedor[i] != null){
                if(contenedor[i].getClave() == clave &&
contenedor[i].getEstado() == 1){
                    yaExiste = true;
                    System.out.println("El valor ya existe en la tabla
hash.");
                }
            }
        }
    }
    if(!yaExiste){
        if( alfaAFuturo >= alfaMaximo) { //camino redimensionar la
hash table.
            redimensionar();
        }
        int colisiones = 0;
        int indice = funcionHash(clave, colisiones);
        while(hayColision(indice)){
```

```
            colisiones++;
            indice = funcionHash(clave, colisiones);
        }
        Celda<Valor> celda = new Celda(clave, v);
        celda.setEstado(1);
        contenedor[indice] = celda;
        numElementos++;
    }
}
```

borrar (int clave): Boolean

Esta función recibe como parámetro una clave de la función hash para poder borrarla luego.

Comenzamos verificando si la tabla hash está vacía utilizando el método "esVacia()". Si es así, se imprimirnos un mensaje diciendo que la tabla esta vacía y se devolvemos la variable "devolución" que sigue siendo false dado que no ocurrió nada que la cambiara dentro de la función.

Si la tabla no es vacía, se declara una variable de tipo entero llamada "colisiones" y se inicializa en 0. También se declara otra variable de tipo entero llamada "índice" que se obtiene a través de una función hash que utiliza la clave buscada y el número de colisiones. Estas dos variables nos van a permitir buscar la celda donde está el elemento que queremos borrar. Continuamos con un bucle while que comprueba si hay colisiones en la posición del índice calculado anteriormente. Si hay colisiones, se comprueba si la clave de la celda actual del contenedor es igual a la clave buscada y si su estado es igual a 1, significando que la celda está ocupada, si el estado fuera distinto de 1 no tendría sentido realizar un borrado. Entonces cambiamos el estado de la celda a -1, se decrementa el número de elementos en la tabla y se asigna a la variable "devolución" el valor true.
Si no se ha encontrado la clave buscada en la celda actual o si su estado es distinto de 1, se incrementa el número de colisiones y se recalcula el índice a través de la función hash nuevamente.

Nuestro bucle continúa hasta que se elimina la celda correspondiente a la clave buscada o hasta que se recorren todas las celdas del contenedor. Luego devolvemos el valor de devolución.

```
public boolean borrar(int clave){
    boolean devolucion = false;
    if(esVacia()){
        System.out.println("La tabla hash está vacía.");
        return devolucion;
    }

    int colisiones = 0;
    int indice = funcionHash(clave, colisiones);

    while(hayColision(indice) ){
        if(contenedor[indice].getClave() == clave &&
contenedor[indice].getEstado() == 1){
            contenedor[indice].setEstado(-1);
            numElementos--;
            devolucion = true;
        }
        colisiones++;
```

```
            indice = funcionHash(clave, colisiones);
        }
        return devolucion;
    }
```

get(int clave): Valor

Función que dada una clave de la tabla hash como argumento, me devuelve el valor que hay almacenado en la celda que tiene dicha clave.

Empezamos el código realizando una comprobación básica si la tabla hash está vacía utilizando el método "esVacia()". Si es así, se imprime un mensaje en la consola indicando que la tabla está vacía y se devuelve la variable "devolución" que sigue siendo null. En el caso que la tabla hash no sea vacía lo que haremos es un bucle while con el objetivo de buscar la celda que tiene la clave que estamos buscando, el bucle comprueba si hay colisiones en la posición del índice calculado anteriormente. Si hay colisiones, se comprueba si la clave de la celda actual del contenedor es igual a la clave buscada. Si es así, se asigna a la variable "devolución" el valor asociado a la clave actual.

Si no encontramos la clave buscada en la celda actual, se incrementa el número de colisiones y se recalcula el índice a través de la función hash.
Continuamos así hasta que se encuentra la celda correspondiente a la clave buscada o hasta que se recorren todas las celdas del contenedor. Finalmente, se devuelve la variable "devolución" que contiene el valor asociado a la clave buscada o null si no se encontró ninguna celda con la clave buscada.

```
public Valor get(int clave){
    Valor devolucion = null;

    if(esVacia()){
        System.out.println("La tabla hash está vacía.");
        return devolucion;
    }

    int colisiones = 0;
    int indice = funcionHash(clave, colisiones);

    while(hayColision(indice)){
        if(contenedor[indice].getClave() == clave){
            devolucion = contenedor[indice].getValor();
        }
        colisiones++;
        indice = funcionHash(clave, colisiones);
    }

    return devolucion;
}
```

esVacia (): boolean

Función que devuelve un true o false dependiendo de si la tabla hash esta vacía o no.

```
public boolean esVacia(){
    boolean devolucion = false;
    if(numElementos == 0){
        devolucion = true;
```

```
    }
    return devolucion;
}
```

factorCarga (): float

Dentro de la tabla hash es fundamental antes de realizar una inserción saber si nuestra tabla esta por pasar o no la cantidad de datos máxima que queremos que almacene. Para eso, debemos comprar el factor de carga con el alfaMax. En esta función lo que hacemos es devolver ese factor de carga actual, el cual lo obtenemos dividiendo la cantidad de elementos en la tabla sobre el tamaño.

```
public float factorCarga(){
    float devolucion = 0;
    devolucion = (float) (numElementos / contenedor.length);
    return devolucion;
}
```

hayColision (int index): boolean

Esta función nos devuelve un true o false si dado un índice, si la celda en la tabla hash de este índice está ocupada o no. Lo que hacemos es primero en el primer bloque if, es verificar si la tabla está vacía o no, si está vacía no hay celdas ocupadas. Si la tabla no está vacía, voy a entrar en el índice del contenedor y verificar si la celda está ocupada o no (estado 1 es ocupada). Si está ocupada de vuelo true significado que hay colisión, devuelvo false si no hay colisión. Está rodeado de un try catch para evitar excepciones de null pointer.

```
private boolean hayColision(int index){
    boolean devolucion = false;
    if(!esVacia()){
        try{
            if(contenedor[index].getEstado() == 1){
                devolucion = true;
            }
        }catch (NullPointerException e){
            devolucion = false;
        }
    }
    return devolucion;
}
```

funcionHash (int clave, int colisiones) : int

La clase hash tiene una función hash asociada para decidir en qué posición de la tabla se tiene que insertar un elemento, y en caso de haber colisiones, cual es la nueva posición obtenida tras resolver dicha colisión. La parte clave de esta función es la variable devolución, la cual obtenemos su resultado con los métodos hash1 y hash2 que explicamos más adelante. Una vez tengo el resultado lo que procedo a hacer es verificar si ese resultado es más grande que el tamaño de mi tabla, si lo es le hago el módulo con el tamaño de la tabla, si no lo es devuelvo su valor normal.

```
private int funcionHash(int clave, int colisiones){
    int devolucion = 0;
    devolucion = (hash1(clave) + hash2(clave, colisiones));

    if(devolucion > contenedor.length){
        devolucion = devolucion % contenedor.length;
```

```
    }
    return devolucion;
}
```

hash1 (int clave): int

Función que dada una clave, realiza unos cálculos básicos con el tamaño de la tabla y nos devuelve la primer parte de nuestra función hash.

```
private int hash1(int clave){
    int devolucion = 0;
    devolucion = clave % contenedor.length;
    return devolucion;
}
```

hash2 (int clave, int colisiones): int

Función que, dada una clave y una cantidad de colisiones, realiza unos cálculos básicos con el tamaño de la tabla y nos devuelve la primera parte de nuestra función hash.

```
private int hash2(int clave, int colisiones){
    int devolucion = 0;
    devolucion = (colisiones * (7 - (clave % 7)));
    return devolucion;
}
```

redimensionar (): void

Esta función es una de las más importantes dentro de nuestro código, se ejectua cuando vamos a insertar un elemento y ese elemento genera que el factor de carga sea mayor al alfaMax, por lo tanto, debemos crear una nueva tabla con un nuevo tamaño, reinsertar los elementos anteriores y por último insertar el elemento que genero el redimensionar.

Explicando un más acerca de que hace el redimensionar,

Comenzamos declarando una variable de tipo int llamada "nuevaCapacidad" que se inicializa con el valor del siguiente número primo después de multiplicar la longitud actual del contenedor por 2. Que el nuevo tamaño del hash sea un primo nos va a ayudar a tener menos colisiones y asegurar un tamaño optimo.
Utilizaremos un "contenedorProvisional" para pasar los valores de un contenedor a otro, aunque sabemos que no es la mejor técnica, es la forma que más intuitiva nos pareció y no nos complicaba tanto. El nuevo contenedor de tipo Celda<Valor> se inicializa con la misma longitud que el contenedor original, esto es todo previo a realizar el cambio al nuevo contenedor. Con el método "Arrays.copyOf()" pasamos a copiar todos los elementos del contenedor original al arreglo temporal "contenedorProvisional".

Remplazamos el contenedor original por un nuevo arreglo de tipo Celda<Valor> con la nueva capacidad calculada, el contendor por ahora esta vacío, empezaremos a llenarlo con los valores que ahora están almacenados en contenedorProvisional.
Para insertar los valores anteriores realizamos un bucle que itera sobre cada elemento del contenedor "contenedorProvisional" y se verifica si el elemento no es nulo.
Si el elemento no es nulo, se declara una variable "colisiones" inicializada en 0 y otra variable "indice" que calculamos a través de una función hash que utiliza la clave del elemento y el número de colisiones. Si hay una colisión, se vuelve a calcular el índice de la celda hasta que se encuentre una celda vacía.

Luego creamos una nueva instancia de Celda<Valor> con la clave y el valor del elemento actual del arreglo temporal. Establecemos el estado de la celda como 1 para indicar que la celda está ocupada. Por último, se agrega la celda a la posición correspondiente del nuevo contenedor utilizando el índice calculado previamente.

```
private void redimensionar(){
    int nuevaCapacidad = siguientePrimo(contenedor.length * 2);

    Celda<Valor>[] contenedorProvisional = new
Celda[contenedor.length];

    contenedorProvisional = Arrays.copyOf(contenedor,
contenedor.length);

    contenedor = new Celda[nuevaCapacidad];

    for (int i = 0; i < contenedorProvisional.length; i++) {
        if(contenedorProvisional[i] != null){
            int colisiones = 0;
            int indice =
funcionHash(contenedorProvisional[i].getClave(), colisiones);
            while(hayColision(indice)){
                indice =
funcionHash(contenedorProvisional[i].getClave(), colisiones);
                colisiones++;
            }
            Celda<Valor> celda = new
Celda(contenedorProvisional[i].getClave(),
contenedorProvisional[i].getValor());
            celda.setEstado(1);
            contenedor[indice] = celda;
        }
    }
}
```

esPrimo (int): boolean

Función que, dado un número, devolvemos true o false si el número es primo o no. Lo primero es verificar que el número no es menor o igual que 1 dado que estos números no son primos, si este no es el caso, utilizamos la propiedad de la raíz cuadrada que tiene los números primos para comprobar si este número la cumple o no.

```
private boolean esPrimo(int numero){
    if (numero <= 1) {
        return false;
    }
    int i = 2;
    while (i <= Math.sqrt(numero)) {
        if (numero % i == 0) {
            return false;
        }
        i++;
    }
    return true;
}
```

siguientePrimo (int): int

Función que devuelve el siguiente número primo a partir de un numero dado. El objetivo de esta función es para cuando hacemos el redimensionar poder obtener el nuevo tamaño de la tabla como un número primo. Lo que hacemos en la función es un bucle llamado a la función esPrimo en la condición, si no se cumple le sumo uno al número hasta que sea primo, luego devuelvo su valor.

```
private int siguientePrimo(int numero){
    while(!esPrimo(numero)){
        numero++;
    }
    return numero;
}
```

toString (): String

Método el cual me imprime el contendió que hay dentro de mi tabla hash en un formato entendible y fácil de observar. Lo que hacemos es recorrer el contendor e ir agregando los valores del contenedor al string devolución, si el contenedor tiene información en ese índice devuelvo el contenido, sino lo tiene lo devuelvo como null, esto lo hacemos así para que sea más fácil observar todo el contenido dentro de la tabla hash.

```
public String toString(){
    String devolucion = "";
    for (int i = 0; i < contenedor.length; i++) {
        if(contenedor[i] != null){
            devolucion += "Indice: " + i  + "    |" + " Estado: " +
contenedor[i].getEstado() + "           " + "Clave: " +
contenedor[i].getClave() + "       " + "Valor: " +
contenedor[i].getValor() + "\n";
        }else{
            devolucion += "Indice: " + i  + "    |" + " Estado: " +
"null       " + " " + "Clave: " +  "null     " + "Valor: " +  "null" +
"\n";
        }
    }
    return devolucion;
}
```

Funciones pequeñas:

Por último, tenemos varias funciones que son importantes y su código no es muy extenso, estas funciones sirven para setear y acceder a la información de la clase.
getAlfaMax (): float devuelve el valor del alfamax.
setAlfaMax (float): void establece el nuevo valor del alfamax.
getNumElementos (): int devuelve la cantidad de elementos en el contenedor.

Clase Celda

Valores de la clase:

- (int)Clave -> clave que tiene la celda.
- (Valor)valor -> valor de esa celda asociado a una clave

- (int)estado -> variable la cual puede ser -1 si la celda ha sido borrada, 0 si está vacía y 1 si está ocupada.

Esta clase cuenta con su propio constructor en la que le pasamos como valores de entrada la clave y el valor, dentro de ella asocia estos dos valores a las variables y además el estado lo pone a 0 por defecto.

Las diferentes funciones que podemos encontrar en esta clase son los métodos setClave, setValor, getEstado, getClave y getValor. Además, contamos con una función llamada equals que recibe un Objeto y sirve para comparar dos celdas, devolviendo true si las celdas son iguales o false en caso contrario.

Pruebas

Para comprobar que el código que realizamos es correcto, vamos a realizar varias pruebas sobre el TAD Hash para poder confirmar que realizamos una implementación correcta. Para esto, vamos a definir varias hash tables en esta sección las cuales calcularemos nosotros manualmente sus inserciones y luego comprobaremos si nuestro programa las realiza de igual forma. Cada hash table creado a continuación manejara las colisiones con direccionamiento abierto y con función hash:

- H (clave, colisiones) = H1 (clave) + H2 (clave, colisiones)
- H1 (clave) = clave mod N
- H2 (clave, colisiones) = colisiones * (7 – (clave mod 7))

Hash 1:
Tamaño 11, ósea N=11.
Alfa máxima = 0,75.
Creamos la tabla vacía con 3 columnas, Índice, calve y valor.

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Insertamos (11,” Geronimo”)
Primer intento de inserción:
Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{1}{11} = 0{,}09$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

- H (11, 0) = H1 (11) + H2 (11, 0)
- H1 (11) = 11 mod 11 = 0
- H2 (11, 0) = 0 * (7 – (11 mod 7)) = 0

Obtenemos que:

- H (11, 0) = H1 (11) + H2 (11, 0) = 0 + 0 = 0

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Insertamos (13," Angel")

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{2}{11} = 0{,}18$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (13, 0) = H1 (13) + H2 (13, 0)
- H1 (13) = 13 mod 11 = 2
- H2 (13, 0) = 0 * (7 – (13 mod 7)) = 0

Obtenemos que:

- H (13, 0) = H1 (13) + H2 (13, 0) = 2 + 0 = 2

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel
3		
4		
5		
6		
7		
8		
9		
10		

Insertamos (15," Borja")

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{3}{11} = 0{,}27$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (15, 0) = H1 (15) + H2 (15, 0)
- H1 (15) = 15 mod 11 = 4
- H2 (15, 0) = 0 * (7 – (15 mod 7)) = 0

Obtenemos que:

- H (15, 0) = H1 (15) + H2 (15, 0) = 4 + 0 = 4

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel
3		
4	15	Borja
5		
6		
7		
8		
9		
10		

Insertamos (22,” David”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{4}{11} = 0{,}36$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (22, 0) = H1 (22) + H2 (22, 0)
- H1 (22) = 22 mod 11 = 0
- H2 (22, 0) = 0 * (7 – (22 mod 7)) = 0

Obtenemos que:

- H (22, 0) = H1 (22) + H2 (22, 0) = 0 + 0 = 0 ¡Colisión!

Segundo intento de inserción:

- H (22, 1) = H1 (22) + H2 (22, 1)
- H1 (22) = 22 mod 11 = 0
- H2 (22, 1) = 1 * (7 – (22 mod 7)) = 1 * (7 - 1) = 6

Obtenemos que:

- H (22, 1) = H1 (22) + H2 (22, 1) = 0 + 1 = 6

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel

3		
4	15	Borja
5		
6	22	David
7		
8		
9		
10		

Insertamos (33,” Gustavo”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{5}{11} = 0{,}45$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (33, 0) = H1 (33) + H2 (33, 0)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 0) = 0 * (7 – (33 mod 7)) = 0

Obtenemos que:

- H (33, 0) = H1 (33) + H2 (33, 0) = 0 + 0 = 0 ¡Colisión!

Segundo intento de inserción:

- H (33, 1) = H1 (33) + H2 (33, 1)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 1) = 1 * (7 – (33 mod 7)) = 1 * (7 - 5) = 2 ¡Colisión!

Tercer intento de inserción:

- H (33, 2) = H1 (33) + H2 (33, 2)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 2) = 2 * (7 – (33 mod 7)) = 2 * (7 - 5) = 2 * 2 = 4 ¡Colisión!

Cuarto intento de inserción:

- H (33, 3) = H1 (33) + H2 (33, 3)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 3) = 3 * (7 – (33 mod 7)) = 3 * (7 - 5) = 3 * 2 = 6 ¡Colisión!

Quinto intento de inserción:

- H (33, 4) = H1 (33) + H2 (33, 4)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 4) = 4 * (7 – (33 mod 7)) = 4 * (7 - 5) = 4 * 2 = 8 ¡Libre!

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	**Clave**	**Valor**
0	11	Geronimo
1		
2	13	Angel

3		
4	15	Borja
5		
6	22	David
7		
8	33	Gustavo
9		
10		

Hash 2:
Tamaño 11, ósea N=11.
Alfa máxima = 0,5.
Creamos la tabla vacía con 3 columnas, Índice, calve y valor.

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Insertamos (11,” Geronimo”)
Primer intento de inserción:
Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{1}{11} = 0{,}09$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

- H (11, 0) = H1 (11) + H2 (11, 0)
- H1 (11) = 11 mod 11 = 0
- H2 (11, 0) = 0 * (7 – (11 mod 7)) = 0

Obtenemos que:

- H (11, 0) = H1 (11) + H2 (11, 0) = 0 + 0 = 0

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2		
3		
4		
5		
6		
7		

8		
9		
10		

Insertamos (13,” Angel”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{2}{11} = 0{,}18$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (13, 0) = H1 (13) + H2 (13, 0)
- H1 (13) = 13 mod 11 = 2
- H2 (13, 0) = 0 * (7 – (13 mod 7)) = 0

Obtenemos que:

- H (13, 0) = H1 (13) + H2 (13, 0) = 2 + 0 = 2

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel
3		
4		
5		
6		
7		
8		
9		
10		

Insertamos (15,” Borja”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{3}{11} = 0{,}27$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (15, 0) = H1 (15) + H2 (15, 0)
- H1 (15) = 15 mod 11 = 4
- H2 (15, 0) = 0 * (7 – (15 mod 7)) = 0

Obtenemos que:

- H (15, 0) = H1 (15) + H2 (15, 0) = 4 + 0 = 4

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel

3		
4	15	Borja
5		
6		
7		
8		
9		
10		

Insertamos (22,” David”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{4}{11} = 0{,}36$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (22, 0) = H1 (22) + H2 (22, 0)
- H1 (22) = 22 mod 11 = 0
- H2 (22, 0) = 0 * (7 – (22 mod 7)) = 0

Obtenemos que:

- H (22, 0) = H1 (22) + H2 (22, 0) = 0 + 0 = 0 ¡Colisión!

Segundo intento de inserción:

- H (22, 1) = H1 (22) + H2 (22, 1)
- H1 (22) = 22 mod 11 = 0
- H2 (22, 1) = 1 * (7 – (22 mod 7)) = 1 * (7 - 1) = 6

Obtenemos que:

- H (22, 1) = H1 (22) + H2 (22, 1) = 0 + 1 = 6

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel
3		
4	15	Borja
5		
6	22	David
7		
8		
9		
10		

Insertamos (33,” Gustavo”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{5}{11} = 0{,}45$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (33, 0) = H1 (33) + H2 (33, 0)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 0) = 0 * (7 – (33 mod 7)) = 0

Obtenemos que:

- H (33, 0) = H1 (33) + H2 (33, 0) = 0 + 0 = 0 ¡Colisión!

Segundo intento de inserción:

- H (33, 1) = H1 (33) + H2 (33, 1)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 1) = 1 * (7 – (33 mod 7)) = 1 * (7 - 5) = 2 ¡Colisión!

Tercer intento de inserción:

- H (33, 2) = H1 (33) + H2 (33, 2)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 2) = 2 * (7 – (33 mod 7)) = 2 * (7 - 5) = 2 * 2 = 4 ¡Colisión!

Cuarto intento de inserción:

- H (33, 3) = H1 (33) + H2 (33, 3)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 3) = 3 * (7 – (33 mod 7)) = 3 * (7 - 5) = 3 * 2 = 6 ¡Colisión!

Quinto intento de inserción:

- H (33, 4) = H1 (33) + H2 (33, 4)
- H1 (33) = 33 mod 11 = 0
- H2 (33, 4) = 4 * (7 – (33 mod 7)) = 4 * (7 - 5) = 4 * 2 = 8 ¡Libre!

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0	11	Geronimo
1		
2	13	Angel
3		
4	15	Borja
5		
6	22	David
7		
8	33	Gustavo
9		
10		

Insertamos (19,” Facundo”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{6}{11} = 0{,}54$. Como es mayor que el alfa máxima tenemos que redimensionar nuestra tabla hash. Por lo tanto, empezamos encontrando el nuevo tamaño de nuestra nueva tabla hash.

Nuevo tamaño = 2 * tamaño anterior lo cual es 22 = 11*2, pero no termina ahí el proceso, para el nuevo tamaño debemos encontrar el próximo número primo a 22, el cual en este caso es 23. Por lo tanto, en nuevo tamaño de nuestra tabla es 23.

A continuación, mostramos la tabla:

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		

Por lo tanto, ahora debemos volver a insertar todos los elementos que insertamos previamente, pero tomando en cuenta el tamaño de esta nueva tabla.

Insertamos (11,” Geronimo”)
Primer intento de inserción:
Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{1}{23} = 0{,}043$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

- H (11, 0) = H1 (11) + H2 (11, 0)
- H1 (11) = 11 mod 23 = 11
- H2 (11, 0) = 0 * (7 – (11 mod 7)) = 0

Obtenemos que:

- H (11, 0) = H1 (11) + H2 (11, 0) = 11 + 0 = 11

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0		

1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	11	Geronimo
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		

Insertamos (13,” Angel”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{2}{23} = 0{,}087$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (13, 0) = H1 (13) + H2 (13, 0)
- H1 (13) = 13 mod 23 = 13
- H2 (13, 0) = 0 * (7 – (13 mod 7)) = 0

Obtenemos que:

- H (13, 0) = H1 (13) + H2 (13, 0) = 13 + 0 = 13

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	**Clave**	**Valor**
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	11	Geronimo

12		
13	13	Angel
14		
15		
16		
17		
18		
19		
20		
21		
22		

Insertamos (15,” Borja”)

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{3}{23} = 0{,}13$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (15, 0) = H1 (15) + H2 (15, 0)
- H1 (15) = 15 mod 23 = 15
- H2 (15, 0) = 0 * (7 – (15 mod 7)) = 0

Obtenemos que:

- H (15, 0) = H1 (15) + H2 (15, 0) = 15 + 0 = 15

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	**Clave**	**Valor**
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	11	Geronimo
12		
13	13	Angel
14		
15	15	Borja
16		
17		
18		
19		
20		
21		
22		

Insertamos (22," David")

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{4}{23} = 0{,}17$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (22, 0) = H1 (22) + H2 (22, 0)
- H1 (22) = 22 mod 23 = 22
- H2 (22, 0) = 0 * (7 – (22 mod 7)) = 0

Obtenemos que:

- H (22, 0) = H1 (22) + H2 (22, 0) = 22 + 0 = 22

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	11	Geronimo
12		
13	13	Angel
14		
15	15	Borja
16		
17		
18		
19		
20		
21		
22	22	David

Insertamos (33," Gustavo")

Nuestro factor de carga si insertamos este número seria: $\alpha = \frac{5}{23} = 0{,}217$. Como es menor que el alfa máxima no hay necesidad de redimensionamiento por ahora.

Primer intento de inserción:

- H (33, 0) = H1 (33) + H2 (33, 0)
- H1 (33) = 33 mod 23 = 10
- H2 (33, 0) = 0 * (7 – (33 mod 7)) = 0

Obtenemos que:

- H (33, 0) = H1 (33) + H2 (33, 0) = 10 + 0 = 10

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10	33	Gustavo
11	11	Geronimo
12		
13	13	Angel
14		
15	15	Borja
16		
17		
18		
19		
20		
21		
22	22	David

Como ya terminamos de insertar todos los elementos anteriores, ahora insertamos el que hizo redimensionar la tabla:
Insertamos (19,” Facundo”)
Primer intento de inserción:

- H (19, 0) = H1 (19) + H2 (19, 0)
- H1 (19) = 19 mod 23 = 19
- H2 (19, 0) = 0 * (7 – (19 mod 7)) = 0

Obtenemos que:

- H (19, 0) = H1 (19) + H2 (19, 0) = 19 + 0 = 19

Esta posición esta libre en nuestra tabla, por lo que podemos insertar el dato. Obteniendo que:

Índice	Clave	Valor
0		
1		
2		
3		
4		
5		
6		
7		
8		

9		
10	33	Gustavo
11	11	Geronimo
12		
13	13	Angel
14		
15	15	Borja
16		
17		
18		
19	19	Facundo
20		
21		
22	22	David

Ahora que tenemos estas dos tablas hash creadas, estamos en condiciones de poder realizar pruebas en nuestro código y poder contrastar si lo que estamos haciendo está realmente bien. Hemos realizado un total de siete tests diferentes, a continuación, procederemos a explicar lo que hace cada uno:

Test1()

Insertamos 5 elementos a la tabla Hash, con la intención de comprobar que ocurrirá si alguno de los elementos tiene varias colisiones. Lo hacemos a través de esta pequeña comprobación.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");

if (hash.get(11).equals("Geronimo") && hash.get(13).equals("Angel") &&
hash.get(15).equals("Borja") &&
        hash.get(22).equals("David") && hash.get(33).equals("Gustavo")
&& hash.get(1) == null &&
        hash.get(100) == null && hash.getAlfaMax() == 0.75f &&
hash.getNumElementos() == 5 ) {
    resultado = true;
}

if(resultado){
    System.out.println("Test 1: OK☑");
}else{
    System.out.println("Test 1: FAIL❌");
}
Terminal:
Test 1: OK✅
```

Test2()

Insertamos 5 elementos a la tabla Hash, para después insertar uno más que provoque el redimensionamiento de la tabla y los coloque en la nueva tabla, con el último elemento bien insertado.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.5f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");

//Elemento que fuerza el redimensionamiento.
hash.insertar(19, "Facundo");

if (hash.get(11).equals("Geronimo") && hash.get(13).equals("Angel") &&
hash.get(15).equals("Borja") &&
        hash.get(22).equals("David") && hash.get(33).equals("Gustavo")
&& hash.get(19).equals("Facundo") && hash.get(1) == null &&
        hash.get(100) == null && hash.getAlfaMax() == 0.5f &&
hash.getNumElementos() == 6){
    resultado = true;
}
if(resultado){
    System.out.println("Test 2: OK☑");
}else{
    System.out.println("Test 2: FAIL❌");
}
```

```
Terminal:
Test 2: OK ✅
```

Test3()

Borrado de varios elementos.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");


hash.borrar(11);
hash.borrar(13);
hash.borrar(15);
hash.borrar(22);

if( hash.get(33).equals("Gustavo") && hash.getNumElementos() == 1){
    resultado = true;
}
```

```
if(resultado){
    System.out.println("Test 3: OK☑");
}else{
    System.out.println("Test 3: FAIL❌");
}

Terminal:
Test 3: OK ✅
```

Test4()

Borrado e inserción de un elemento con la misma clave, pero con diferente valor.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");

hash.borrar(11);

hash.insertar(11, "Susana");

if(hash.get(11).equals("Susana") && hash.getNumElementos() == 5){
    resultado = true;
}
if(resultado){
    System.out.println("Test 4: OK☑");
}else{
    System.out.println("Test 4: FAIL❌");
}

Terminal:
Test 4: OK ✅
```

Test5()

Borrado e inserción de un elemento que sufre de colisión el cual tiene la misma clave, pero con diferente valor.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");

hash.borrar(22);

hash.insertar(22, "Alejandro");
```

```
if(hash.get(22).equals("Alejandro") && hash.getNumElementos() == 5){
    resultado = true;
}
if(resultado){
    System.out.println("Test 5: OK☑");
}else{
    System.out.println("Test 5: FAIL❌");
}
```

```
Terminal:
Test 5: OK ✅
```

Test6()

Insertado en la tabla Hash de varios elementos y después intentamos la inserción de un elemento con la misma clave que uno ya insertado previamente.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");
hash.insertar(22, "Alejandro"); //esto no se debería poder hacer, ya
existe un elemento con la clave 22.

if(hash.get(22).equals("David") && hash.getNumElementos() == 5){
    resultado = true;
}
if(resultado){
    System.out.println("Test 6: OK☑");
}else{
    System.out.println("Test 6: FAIL❌");
}
```

```
Terminal:
El valor ya existe en la tabla hash.
Test 6: OK ✅
```

Test7()

Borrado de un elemento, el cual se volverá a pedir que se borre otra vez.

```
boolean resultado = false;
Hash<String> hash = new Hash<>(11, 0.75f);
hash.insertar(11,"Geronimo");
hash.insertar(13,"Angel");
hash.insertar(15,"Borja");
hash.insertar(22,"David");
hash.insertar(33,"Gustavo");

hash.borrar(22);
```

```
if(hash.borrar(22) == false && hash.getNumElementos() == 4){
    resultado = true;
}
if(resultado){
    System.out.println("Test 7: OK☑");
}else{
    System.out.println("Test 7: FAIL❌");
}

Terminal:
El valor no se encuentra en la tabla
Test 7: OK✅
```

Conclusión

Por último, nos gustaría concluir que la entrega fue muy positiva para aprender como implementar una tabla hash en código, y también para reforzar conceptos sobre las tablas hash. Por último, nos gustaría comentar que donde más encontramos dificultad en la tarea fue con las funciones insert y redimensionar, las dos funciones supusieron un reto ya que al principio estábamos muy perdidos y no sabíamos por donde tirar, luego nos dimos cuenta de que hay varios casos excepcionales, como cuando la tabla esta vacía, o cuando quieren insertar una clave repetida. En el redimensionar lo que nos costó fue que los elementos que volviésemos a insertar se insertar correctamente y no eviten las colisiones o sobrescriban a otros.

Bibliografía y referencias

Madrid, U. C. (14 de Abril de 2023). *Arquitectura de sistemas UC3M*. Obtenido de Tablas hash: https://www.it.uc3m.es/pbasanta/asng/course_notes/ch07.html