

# Estructuras de Datos

---

Universidad Nacional de San Antonio de Areco

# Unidad Didáctica 3. Hashing (o Dispersión).

- Definición de Hashing(Dispersión)
- Atributos.
- Limitaciones
- Tipos de Dispersión.
  - Hashing con espacio de direccionamiento estático
  - Hashing con espacio de direccionamiento dinámico
- Parámetros del Hash
  - Función de Hash.
  - Tamaño de cada nodo (cubeta) de almacenamiento.
  - Densidad de empaquetamiento.
  - Métodos de tratamientos de desbordes (overflow).

# Unidad Didáctica 3. Hashing (o Dispersión).

- Colisiones
- Resolución de Colisiones con overflow.
  - Saturación Progresiva
  - Saturación Progresiva Encadenada
  - Doble Dispersión
  - Área de desbordes por separado
- Hash asistido por tabla
- El proceso de eliminación
- Hash con espacio de direccionamiento dinámico
- Hash Extensible


# Dispersión. Introducción.

Es común al crear estructuras de datos y luego trabajar sobre las mismas, tener la necesidad de realizar búsquedas en forma más frecuente que la necesidad de realizar inserciones.

Por ejemplo si tenemos una lista de personas podríamos querer saber si cierta persona está ingresada, o saber información acerca de la misma. Para estos casos en que queremos realizar consultas sobre la estructura de datos puede ser útil trabajar con una estructura de datos que permita realizar búsquedas rápidas (de orden pequeño).



# Dispersión. Introducción.

- Necesitamos un mecanismo de acceso a registros con una lectura solamente.
- Hasta el momento tenemos:
  - Secuencial:  $N/2$  accesos promedio
  - Ordenado:  $\log_2 N$
  - Árboles: 3 o 4 accesos
- Problema  Solución
- ¿Cómo lo logramos?

# Dispersión. Definiciones.

## Dispersión o Hashing

- Técnica para generar una dirección base única para una **llave (clave)** dada. La dispersión se usa cuando se requiere acceso rápido a una llave.
- Técnica que convierte la **llave asociada a un registro de datos** en un número aleatorio, el que sirve después para determinar dónde se almacenará el registro.
- Técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en direcciones de almacenamiento en memoria secundaria.



# Dispersión. Atributos.

La técnica de dispersión presenta una serie de **atributos** o **propiedades**:

- **No se requiere almacenamiento adicional**, es el archivo de datos el que resulta disperso. Esta dispersión se genera a partir de la clave primaria de un registro de información del archivo , que al aplicarle la función de hash, se determina la dirección de memoria resultante dónde se almacenará.
- **Facilita** la inserción y eliminación rápida de registros.
- **Localiza** registros dentro de un archivo con un **sólo acceso a disco**.
- La idea básica consiste en transformar las claves en direcciones de memoria mediante una **función de transformación**. Estas direcciones pueden hacer referencia a memoria primaria, en cuyo caso tendríamos un array, pero en general las tablas de dispersión **aprovechan su potencial para la implementación en memoria secundaria**.

# Dispersión. Limitaciones.

El método de dispersión presenta algunas **limitaciones**, casos en donde no es posible aplicarlo.

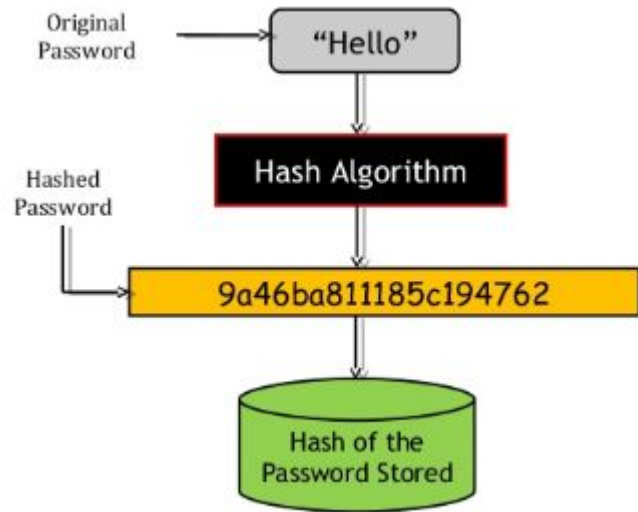
- No podemos usar registros de longitud variable, esto se debe a que cada dirección física obtenida debe tener la capacidad de almacenar un registro de tamaño conocido. No puede pasar que el registro no entre en dicho espacio.
- No puede haber orden físico de datos. Los registros son esparcidos dentro del archivo de datos, de acuerdo a la dirección de memoria que se obtiene con la aplicación de la función de hash sobre la clave.
- No permite llaves duplicadas, por lo tanto, el hash no puede usarse sobre claves secundarias.



# Tipos de Dispersión.

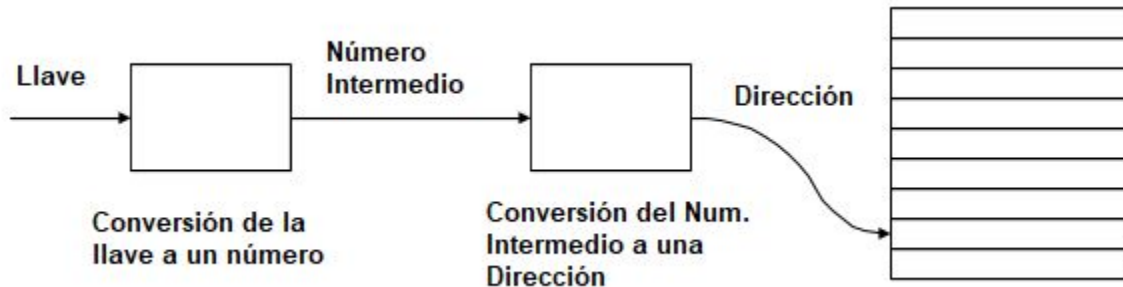
El método de hashing presenta dos alternativas (políticas) para su implementación:

- **Hashing con espacio de direccionamiento estático:** el **espacio disponible** para dispersar los registros dentro del archivo de datos está **fijado previamente**. Así, la función de hash aplicada a una clave da como resultado una dirección física posible dentro del espacio disponible para el archivo.



# Tipos de Dispersión.

- **Hashing con espacio de direccionamiento dinámico:** el **espacio disponible** para dispersar los registros dentro del archivo de datos **aumenta o disminuye en función de las necesidades** de espacio que en cada momento tiene el archivo. Así, la función de hash aplicada a una clave da como resultado un valor intermedio, que será utilizado para obtener una dirección física posible para el archivo. Estas direcciones no están establecidas a priori y son generadas de manera dinámica.



# Dispersión. Parámetros.

El método de dispersión, cuando utiliza **espacio de direccionamiento estático**, presenta **cuatro parámetros** que definen su comportamiento:

1. Función de Hash.
2. Tamaño de cada nodo (cubeta) de almacenamiento.
3. Densidad de empaquetamiento.
4. Métodos de tratamientos de desbordes (overflow).

# Dispersión. Parámetros.

## Función de Hash.

- Esta **función de hash** puede verse como una **caja negra** que recibe como entrada una clave, y produce una dirección de memoria donde almacenar el registro asociado a la clave en el archivo de datos.
- Una **función de hash o dispersión** es una función que transforma un valor, que representa la llave primaria de un registro, en otro valor, dentro de un determinado rango, que se utiliza como dirección física para insertar un registro en un archivo de datos.
- Una **llave o clave primaria**, es un valor que identifica unívocamente a ese registro de datos de los demás registros. Por ejemplo, si en los registros almacenamos personas, el DNI, es una clave primaria.

# Dispersión. Parámetros.

**Función de Hash.** Se presenta un inconveniente:

Supongamos que tenemos el siguiente algoritmo de función de hash, que dada una clave, devuelve un valor en función de el código ASCII de sus caracteres:

```
function hash_ascii ( llave: string, tamaño: integer ) : integer;  
var valor, i : integer;  
begin  
    valor := ASCII( llave[1] );  
    for i = 2 to tamaño  
        valor := valor + ASCII( llave[i] );  
    hash_ascii := valor / numero_direcciones_fisicas;  
end;
```

# Dispersión. Parámetros.

**Función de Hash.** Se presenta un inconveniente:

Supongamos ahora que tenemos las claves DACA y CADA, si se aplica la función anterior, el valor ASCII resultante será el mismo, por lo tanto, los registros de datos asociados a ambas claves, deberían almacenarse en el mismo lugar de memoria.

Esta situación se denomina **colisión** entre ambos registros.

Y ambas claves se denominan **sinónimos**. Se dice que dos claves distintas  $x_1$  y  $x_2$  son **sinónimos** para una función de transformación  $H(x)$  dada, si  $H(x_1) = H(x_2)$ .

# Dispersión. Colisiones.

Las **colisiones** ocasionan problemas, **no es posible almacenar dos registros en el mismo espacio físico**. Es necesario encontrar una solución. Tenemos dos alternativas:

1. **Elegir un algoritmo de dispersión perfecto, que no genere colisiones.** Este tipo de algoritmo debe asegurar que dadas dos claves diferentes siempre se obtendrán dos direcciones físicas diferentes. Se ha demostrado que obtener este tipo de funciones resulta en extremo muy difícil.

# Dispersión. Colisiones.

2. **Minimizar el número de colisiones a una cantidad aceptable**, y de esta manera tratar a una colisión como una condición excepcional. Existen diferentes formas de reducir el número de colisiones:

- a. Distribuir los registros de la forma más aleatoria posible, es decir que cada clave tenga una dirección distinta.
- b. Utilizar más espacio de disco que el necesario. Si tenemos 10 claves y usamos 10 lugares, significa que tenemos un lugar para cada registro y la posibilidad de colisión entre las claves es mayor. Si utilizamos 100 espacios, tenemos 10 lugares posibles para cada registro. La desventaja es el desperdicio de espacio.
- c. Ubicar o almacenar más de un registro por cada dirección de memoria. Esto significa que las claves sinónimos podrán albergarse en la misma dirección física. Si a pesar de la capacidad del nodo o cubeta, ya no entran más claves sinónimos, se produce una **saturación** o **desborde**.



# Dispersión. Parámetros.

## Tamaño del nodo (o cubeta) de almacenamiento.

La **capacidad del tamaño del nodo** queda determinada por la posibilidad de transferencia de información en cada operación de entrada/salida desde RAM hacia el disco, y viceversa.

## Densidad de Empaquetamiento.

Se define la **densidad de empaquetamiento (DE)** como la relación entre el espacio disponible para el archivo de datos y la cantidad de registros que integran dicho archivo.

# Dispersión. Parámetros.

## Densidad de Empaquetamiento.

La siguiente fórmula describe la DE como la razón entre la cantidad de registros que componen un archivo (**r**) y el espacio disponible para almacenar dicho archivo.

El espacio disponible se define como la cantidad de nodos direccionables (**n**) por la función de hash, y la cantidad de registros que cada nodo puede almacenar, **registros por nodo (RPN)**.

$$DE = \frac{r}{RPN * n}$$

# Dispersión. Parámetros.

## Densidad de Empaquetamiento.

Si se debieran esparcir 30 registros entre 10 direcciones con capacidad de 5 registros por dirección, la DE sería de 0,6 o 60%.

$$DE = \frac{r}{RPN * n} \quad \longrightarrow \quad DE = \frac{30}{5 * 10} = 0.6$$

# Dispersión. Parámetros.

## **Densidad de Empaquetamiento.**

- Cuanto mayor es la DE, mayor será la posibilidad de colisiones, dado que se dispone de menos espacio para esparcir los registros.
- Si la DE de empaquetamiento se mantiene baja, se dispone de mayor espacio para esparcir los registros y, por ende, disminuye la probabilidad de colisiones.
- Cuando la DE se mantiene baja, se desperdicia espacio, dado que se utiliza menos espacio que el reservado, generando fragmentación del archivo.

# Dispersión. Parámetros.

## Métodos de tratamiento de desbordes (overflow)

Un **desborde** o **overflow** ocurre cuando un registro es direccionado a un nodo que no tiene capacidad para almacenarlo. Se realizan dos acciones:

1. encontrar un lugar para el registro en otra dirección
2. y asegurarse de que el registro posteriormente sea encontrado en esa nueva dirección.

# Dispersión. Resolución de Colisiones con overflow.

Aunque la función de hash sea eficiente y aún con DE relativamente baja, es probable que las colisiones produzcan overflow o saturación. Por lo tanto, se debe contar con algún método para reubicar a aquellos registros que no pueden ser almacenados en la dirección base obtenida a partir de la función de hash.

Tenemos cuatro métodos:

- Saturación Progresiva
- Saturación Progresiva Encadenada
- Doble Dispersión
- Área de desbordes por separado

# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva

El método consiste en almacenar el registro en la dirección siguiente más próxima al nodo donde se produce la saturación.

Ejemplo: tenemos nodos con capacidad 2.

En el nodo 50, están los registros Álvarez y González. Se inserta un nuevo registro Pérez, y la función de hash retorna como dirección base el nodo 50. Que como está lleno genera **overflow**. La primera dirección del nodo posterior capaz de almacenar el registro es el nodo 52. Por lo tanto, Pérez, se **inserta** en ese lugar.

	.....	.....
50	Álvarez	González
51	Abarca	Zurita
52	Hernández	
53	.....	.....

# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva

El proceso de **búsqueda** de información debe sufrir algún cambio. Si se busca a Pérez, la dirección base que determina la función de hash sigue siendo 50. Como el elemento no se encuentra en dicho lugar, y el nodo está completo, se debe continuar la búsqueda en los nodos subsiguientes hasta encontrar el elemento, o hasta encontrar un nodo que no esté completo.

Pérez no se encuentra en el nodo 50, ni en el 51, recién se ubica en el nodo 52. Fueron necesarios 3 accesos para encontrar el registro.

	.....	.....
50	Álvarez	González
51	Abarca	Zurita
52	Hernández	Pérez
53	.....	.....



# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva

Suponga ahora que se busca al registro con clave Rodríguez, la dirección base dada por la función de hash es 50. Se busca el elemento en el nodo 50, luego en el 51, luego en el 52, hasta que, cuando se busca en el nodo 53, se encuentra un nodo que no está completo. Por lo tanto, se puede decir, que esa clave no está en el archivo.

	.....	.....
50	Álvarez	González
51	Abarca	
52	Hernández	Pérez
53	.....	.....

El método podría requerir chequear todas las direcciones disponibles en un caso extremo, para poder localizar un registro.

# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva

Esta técnica necesita una **condición adicional excepcional**. Suponga que se elimina el registro correspondiente a la clave Zurita. En este caso, se intenta localizar a Pérez, y se presenta un nuevo inconveniente. Al chequear la dirección 51, no está completa, por lo que la búsqueda se detendría, sin permitir localizar a Pérez.

El método necesita indicar si una dirección estuvo completa anteriormente, la cual debe ser marcada, como dirección ya saturada, para no impedir la búsqueda de registros.

Se podría utilizar por ejemplo, un carácter especial como # para marcar, que estuvo saturado.

	.....	.....
50	Álvarez	González
51	Abarca	
52	Hernández	Pérez
53	.....	.....

	.....	.....
50	Álvarez	González
51	Abarca	#
52	Hernández	Pérez
53	.....	.....

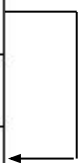
# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva Encadenada

Este método presenta otra variante al tratamiento de saturación. En líneas generales funciona igual que la saturación progresiva. Un elemento que se intenta ubicar en una dirección completa, es direccionado a la dirección inmediata siguiente con espacio disponible. **La diferencia radica en que, una vez localizada la nueva dirección, ésta se encadena o enlaza con la dirección base inicial, generando una cadena de búsqueda de elementos.**

	.....	.....	-1
50	Álvarez	González	-1
51	Abarca	Zurita	-1
52	Hernández		-1
53	.....	.....	-1

	.....	.....	-1
50	Álvarez	González	52
51	Abarca	Zurita	-1
52	Hernández	Pérez	-1
53	.....	.....	-1



# Dispersión. Resolución de Colisiones con overflow.

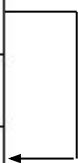
## Saturación Progresiva Encadenada

Con la saturación progresiva fueron necesarios realizar tres accesos para encontrar el registro de la clave Pérez, en cambio con la saturación progresiva encadenada se requieren sólo dos accesos.

Si bien la performance final de cada método dependerá del orden de llegada de las claves, en líneas generales puede decirse que este método presenta mejoras con respecto al anterior.

	.....	.....	-1
50	Álvarez	González	-1
51	Abarca	Zurita	-1
52	Hernández		-1
53	.....	.....	-1

	.....	.....	-1
50	Álvarez	González	52
51	Abarca	Zurita	-1
52	Hernández	Pérez	-1
53	.....	.....	-1




# Dispersión. Resolución de Colisiones con overflow.

## Saturación Progresiva Encadenada

Sin embargo, requiere que cada nodo manipule información extra: la **dirección del nodo siguiente**.

El enlace entre los nodos, sirve tanto para búsquedas, inserciones y eliminaciones. Por ejemplo, se desea insertar otro registro, cuya clave con la función de hash da como resultado el nodo 50, al estar lleno, se sigue el enlace, pero el nodo 52 también está lleno, pero como en su enlace tiene -1, se busca el nodo siguiente más próximo con espacio, se almacena allí y se actualizan los enlaces.

	.....	.....	-1
50	Álvarez	González	52
51	Abarca	Zurita	-1
52	Hernández	Pérez	-1
53	.....	.....	-1



	.....	.....	-1
50	Álvarez	González	52
51	Abarca	Zurita	-1
52	Hernández	Pérez	53
53	Domínguez	.....	-1

# Dispersión. Resolución de Colisiones con overflow.

## Doble Dispersión

El problema que presenta la saturación progresiva es que, a medida que se producen saturaciones, los registros tienden a esparcirse en nodos cercanos. Esto puede provocar un **exceso de saturación sectorizada**.

- El método de la **Doble Dispersión** consiste en tener **dos funciones de hash**.
- Esta segunda función no retorna una dirección, sino que su resultado es un **desplazamiento**.
- Este **desplazamiento** se **suma** a la **dirección base** obtenida en la primera función, y se genera la **nueva dirección** donde se insertará el registro.
- En caso de producirse otro overflow, se deberá sumar reiteradamente el desplazamiento obtenido, y así encontrar una dirección con espacio suficiente para almacenar al nuevo registro.

# Dispersión. Resolución de Colisiones con overflow.

## Doble Dispersión

- La doble dispersión tiende a esparcir los registros en saturación a lo largo del archivo de datos, pero con un efecto lateral importante.
- Los registros en overflow tienden a **ubicarse “lejos”** de sus direcciones de base, lo cual produce un mayor desplazamiento de la cabeza de lectura/escritura del disco rígido, aumentando el tiempo de respuesta ante una búsqueda.



# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

En las técnicas anteriores:

- Cuando ocurre overflow, los registros son dispersados en nodos que no se corresponden con su dirección base original.
- A medida que se completa el archivo, hay muchos registros ocupando direcciones que originalmente no le corresponden.
- Esto disminuye la performance del método de hashing utilizado. (más colisiones).
- Como alternativa se sugiere el uso de **área de desbordes por separado**.



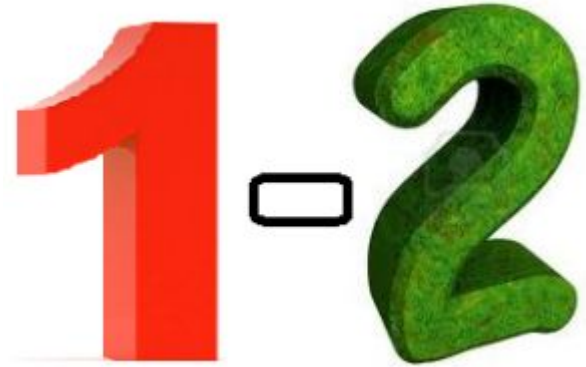


# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

Se distinguen dos tipos de nodos:

- Aquellos **direccionables por la función de hash**.
- Y **aquellos de reserva**, que sólo podrán ser utilizados en caso de saturación o desborde, pero que no son alcanzables por la función de hash.



# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

Continuando con el ejemplo anterior, con cubetas (nodos) de capacidad para dos registros, las claves Alfa y Delta, direccionan a la cubeta 50.

	.....	.....	-1
50	Alfa	Delta	-1
51			-1
52			-1
53			-1

**Nodos**

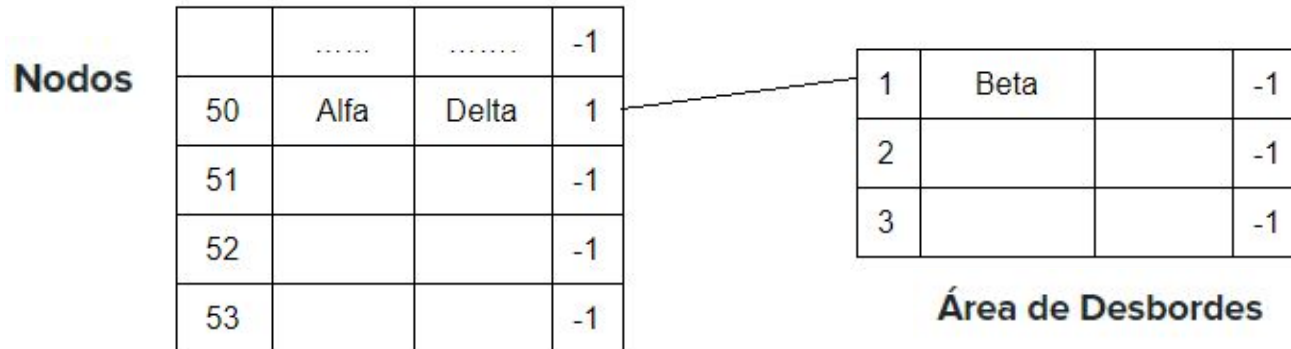
1			-1
2			-1
3			-1

**Área de Desbordes**

# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

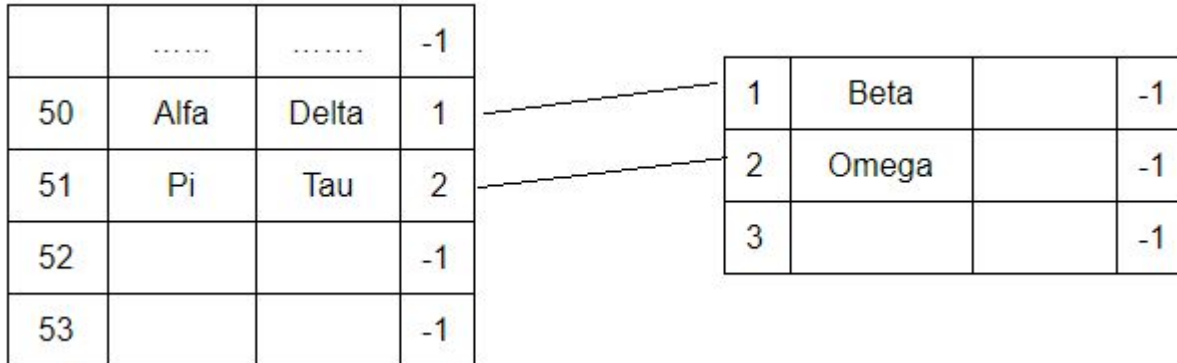
- Luego llega la clave Beta, al dispersarla, la dirección base obtenida a partir de la función de hash es nuevamente el nodo 50.
- Se produce saturación, pues la cubeta 50 está llena, y el registro es reubicado en la primera dirección disponible dentro del área de desbordes separada.
- La dirección base original se encadena con la dirección de reserva.



# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

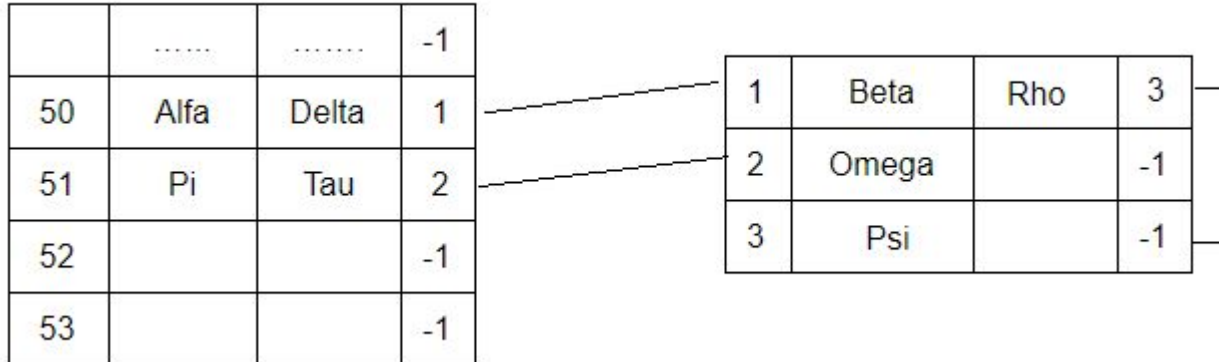
- Luego se dispersan las claves Pi y Tau, con una dirección de base 51, por lo que no se produce desborde.
- Si llega la clave Omega al mismo nodo 51, se produce overflow.
- Se utiliza otra dirección del área de desbordes, la 2 en este caso.



# Dispersión. Resolución de Colisiones con overflow.

## Área de Desbordes por Separado

Por último, si llegaran dos claves más, Rho y Psi, con una dirección base 50, la primera de ellas, Rho, se ubica en la dirección 1 de desbordes, a segunda, Psi, no cabe. Por lo tanto, se redirecciona a un nuevo nodo, el tercero del área de desbordes, el cual se enlaza con el primero.



# Hash asistido por Tabla

- El método de hash resulta ser el más eficiente en términos de recuperación de información, permite conseguir un acceso para recuperar un dato en más de 99,9 % de los casos, cuando la DE es del 75%.
- Las operaciones de alta y de baja se comportan con el mismo nivel de eficiencia para los mismos porcentajes.
- Sin embargo, hay situaciones donde se necesitan más de un acceso para localizar un dato.
- Con los casos anteriores, si bien son alternativas eficientes, a medida que se generan situaciones de saturación, el número de accesos requeridos aumenta.
- Existe otra alternativa, también para **almacenamiento estático**, que asegura tener un sólo acceso para un registro de datos dado. **Se llama Hash asistido por Tabla.**
- Para ser implementada, necesita que una propiedad del hash no se cumpla, es decir, **necesita almacenamiento adicional.**

# Hash asistido por Tabla

Utiliza **tres funciones**:

1. La primera función **FH1** retorna la dirección física del nodo a donde el registro debería almacenarse.
2. La segunda función **FH2**, retorna un desplazamiento (similar al método de doble dispersión).
3. La tercera función **FH3**, retorna una secuencia de bits que **NO** pueden ser todos unos.

**El método comienza con una tabla (estructura adicional) con tantas entradas como direcciones de nodos se tengan disponibles. Cada entrada tendrá todos sus bits en uno.**

# Hash asistido por Tabla

**Ejemplo:** para fines prácticos **FH3** retorna solamente 4 bits. El tamaño de la cubeta es de dos registros. La siguiente tabla muestra el resultado de aplicar las tres funciones de hash a todos los registros del archivo.

Clave	FH1(Clave)	FH2(Clave)	FH3(Clave)
Alfa	50	3	0001
Beta	51	4	0011
Gamma	52	7	0101
Delta	50	3	1000
Epsilon	52	3	0110
Rho	51	5	0100
Pi	50	2	0010
Tau	53	2	1110
Psi	50	9	1010
Omega	50	4	0000



# Hash asistido por Tabla

Veamos ahora la llegada de las claves Alfa, Beta, Gamma, Delta, Epsilon y Rho.

**Tabla en memoria**

....	1111
50	1111
51	1111
52	1111
53	1111
54	1111
...	1111

**Nodos en disco**

50	Alfa	Delta
51	Beta	Rho
52	Gamma	Epsilon

# Hash asistido por Tabla

La siguiente clave a insertar es Pi, que debería insertarse en la dirección 50. Como esta dirección está completa, el procedimiento es el siguiente:

1. Se obtiene el valor de FH3 para todos los registros del nodo 50.
2. Se determina la clave que genera el mayor valor, en este caso, Delta.
3. Se escribe en disco el nodo 50 con todos los registros, menos el de mayor valor.
4. En la tabla de memoria, para la posición 50 queda el valor de FH3 de Delta.
5. Para la clave seleccionada en el paso 2, en esta caso Delta, se obtiene la FH2.
6. Se suma, la FH1 de Delta, más el desplazamiento de FH2. (nodo 53 -->  $50 + 3$ )
7. Se intenta insertar Delta en la dirección resultante, si no hay saturación se inserta. Si hay saturación, se comienza nuevamente desde el paso 1, para la nueva cubeta.

# Hash asistido por Tabla

**Tabla en memoria**

....	1111
50	1000
51	1111
52	1111
53	1111
54	1111
...	1111

**Nodos en disco**

50	Alfa	Pi
51	Beta	Rho
52	Gamma	Epsilon
53	Delta	

# Hash asistido por Tabla

**¿Cómo localizamos a la clave Delta?** El proceso debe asegurar un sólo acceso a disco.

Para localizar a una clave hacemos lo siguiente:

1. Se genera la FH1 para la clave a buscar.
2. Se genera la FH3 para la misma clave.
3. Se chequea el contenido de la tabla en memoria en la posición indicada por FH1, comparándolo con el resultado de FH3:
  - a. Si el resultado de de FH3 es menor que el de la tabla, se busca la clave en cuestión en el nodo indicado por FH1 y el proceso termina.
  - b. Si el resultado de de FH3 es mayor o igual que el de la tabla, el proceso continúa en el paso 4.
4. Se obtiene el valor de FH2 y se suma al valor de FH1 (de la clave buscada), y se obtiene la nueva posición.
5. Se comienza nuevamente desde el paso 3, utilizando ahora como posición el resultado de FH1 + FH2.

# Hash asistido por Tabla

Ahora busquemos a Delta:

1 y 2. Calculamos  $FH1(\text{Delta}) = 50$ ,  $FH3(\text{Delta}) = 1000$ ,

3. al comparar 1000 con el de la tabla para la posición 50, vemos que es igual, por lo tanto,

4. se calcula  $FH2(\text{Delta}) = 3$  y sumamos  $FH1(\text{Delta}) + FH2(\text{Delta}) = 53$

Se comienza nuevamente desde el paso 3,  $FH3(\text{Delta})$  es menor que el valor de la tabla para el nodo 53, por lo tanto Delta, tiene que estar almacenada en esa cubeta.

Se hace una búsqueda secuencial en el nodo 53, que anteriormente es traído a memoria principal. **Y cómo se puede ver sólo tuvimos un acceso a disco, todo lo demás ha sucedido en memoria principal.**

# Hash asistido por Tabla

## ¿Qué pasa si buscamos una clave que no está en el archivo?

Queremos encontrar a la clave Psi, que aún no ha sido almacenada.

Calculamos  $FH1(Psi) = 50$  y  $FH3(Psi) = 1010$ , como el valor 1010 es mayor que el valor 1000 de la tabla para el nodo 50, la clave buscada no está en el nodo 50.

Se obtienen  $FH2(Psi) = 2$ , se suma  $FH1(Psi) + FH2(Psi) = 50 + 2 = 52$  y el valor de  $FH3(Psi) = 1010$  es menor que el valor de la tabla para el nodo 52, que es 1111. Se accede al nodo 52, se hace una búsqueda secuencial y Psi no está, por lo que podemos decir que la clave Psi, no está en el archivo.

# Hash asistido por Tabla

Debemos insertar las claves que faltan: la clave Tau va al nodo 53 y no produce overflow.

**Tabla en memoria**

....	1111
50	1000
51	1111
52	1111
53	1111
54	1111
...	1111

**Nodos en disco**

50	Alfa	Pi
51	Beta	Rho
52	Gamma	Epsilon
53	Delta	Tau

# Hash asistido por Tabla

La clave Psi se debería almacenar en el nodo 50, pero se produce overflow. Se calcula FH3 para Alfa, Pi y Psi; y al ser el valor de Psi el mayor, esta será la clave que se quite del nodo 50.

En este caso como el valor FH3(Psi) es mayor que el valor que está en la tabla para la dirección 50, el valor de FH3(Delta), este valor no debe ser cambiado para que la búsqueda pueda realizarse sin problemas.

El valor de  $FH2(Psi) = 9$ , por lo que  $FH1(Psi) + FH2(Psi) = 50 + 9 = 59$ , por lo que la clave Psi se almacenará en el nodo 59.

**Tabla en memoria**

....	1111
50	1000
51	1111
52	1111
53	1111
54	1111
...	1111
59	1111
...	1111

**Nodos en disco**

50	Alfa	Pi
51	Beta	Rho
52	Gamma	Epsilon
53	Delta	Tau
59	Psi	



# Hash asistido por Tabla

Por último, llega la clave Omega, que también debe ser almacenada en el nodo 50. Se produce overflow, y entre las claves del nodo 50, Alfa, Pi y Omega, la que tiene el valor de FH3 mayor es Pi, por lo tanto, es la que debe salir de dicho nodo.

En el nodo 50 quedan Alfa y Omega, y como  $FH3(Pi)$  es menor que  $FH3(Delta)$  que se encuentra en la entrada de la tabla, se actualiza el valor de la misma, para poder realizar las búsquedas.

Se calcula  $FH2(Pi)$  y se suma a  $FH1(Pi)$  para obtener la nueva cubeta, que da como resultado el nodo 52.

En el nodo 52 se produce nuevamente overflow. Entre las tres claves Pi, Gamma y Epsilon, la que tiene FH3 mayor es Epsilon. En el nodo 52 quedarán entonces, Pi y Gamma, actualizando la entrada de la tabla con el valor de  $FH3(Epsilon)$ .

Epsilon se redireccionará a  $FH1(Epsilon) + FH2(Epsilon) = 52 + 3 = 55$ , es decir, al nodo 55.

# Hash asistido por Tabla

**Tabla en memoria**

....	1111
50	1000
51	1111
52	0010
53	1111
54	1111
55	1111
...	1111
59	1111
...	1111

**Nodos en disco**

50	Alfa	Omega
51	Beta	Rho
52	Gamma	Pi
53	Delta	Tau
55	Epsilon	
59	Psi	

# Hash asistido por Tabla

## Proceso de Eliminación

Se tiene el siguiente proceso para dar de baja a un registro:

1. Se localiza el registro de acuerdo con el proceso de búsqueda definido anteriormente.
2. Si se encontró el registro con la clave buscada, se reescribe el nodo que la contiene sin la clave.



# Hash asistido por Tabla

## Proceso de Inserción

Hay que tener en cuenta un **caso especial**: cuando se quiere borrar una clave de un nodo que se encuentra completo, cuando se intente insertar una nueva clave en el nodo habrá lugar.

Si el nodo ya había producido overflow, la entrada en la tabla contiene el valor de la FH3 de la clave que produjo saturación.

En ese caso, el nuevo elemento a insertar en ese nodo, debe cumplir la siguiente propiedad:

$$\text{FH3}(\text{nuevo elemento}) < \text{Tabla}(\text{valor de la entrada del nodo})$$

Es decir, el valor de FH3(nuevo elemento) debe tener un valor menor al valor que se encuentra en la entrada de la tabla del nodo, para que puedan realizarse las búsquedas de manera correcta.

# Hash asistido por Tabla

## Conclusiones

El método asegura un sólo acceso a disco para encontrar una clave.

Pero el costo asociado presenta dos cuestiones:

1. La necesidad de usar espacio extra para la administración de la tabla en memoria principal.
2. La complejidad adicional del algoritmo de inserción en el caso de que haya saturación. La cantidad de accesos para terminar el proceso está vinculada a la cantidad de overflow sucesivos que se produzcan.

**Podemos decir como conclusión final que esta variante de hash con almacenamiento estático, pondera la búsqueda sobre las otras operaciones.**

# Hash Extensible

- Es una alternativa de implementación para hash con espacio de **direccionamiento dinámico**.
- El método comienza con espacio para un solo nodo para almacenar registros, y luego, va aumentando la cantidad de direcciones disponibles a medida que los nodos se completan.
- Los métodos que trabajan con espacio dinámico, no utilizan la DE, ya que el espacio en disco aumenta o disminuye en función a la cantidad de registros que dispone el archivo en cada momento.

# Hash Extensible

## Problema

El principal problema que se tiene con los métodos de hashing dinámicos en general, y en particular, con el hash extensible, es que las direcciones de los nodos no están prefijadas a priori, por lo tanto, la función de hash no puede retornar una dirección de memoria fija.

**Es necesario cambiar la política de trabajo de la función de dispersión.**



# Hash Extensible

**Función de hash:** retorna un string de bits.

La cantidad de bits que retorna determinará la cantidad máxima de direcciones de memoria a las que se pueda acceder con el método.

$f(x)$  permite  $2^n$  direcciones posibles

donde  $n$  es el número de bits del string que retorna la función para la clave  $x$ .



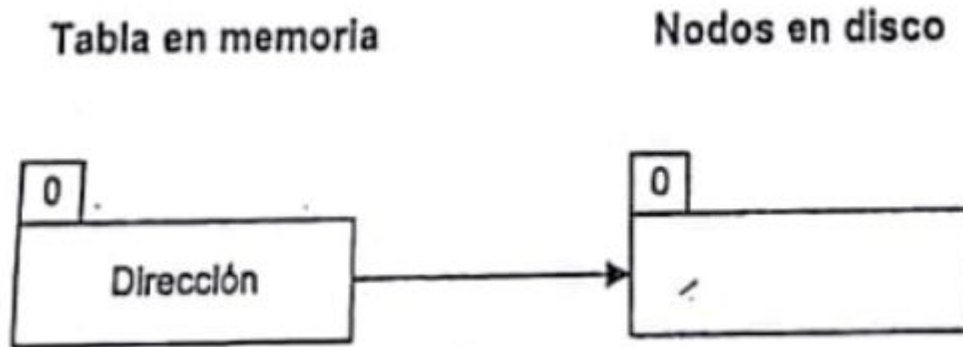
# Hash Extensible

Este método también necesita una estructura auxiliar para poder implementarse:

- Una tabla que se administra en memoria principal.
- **A diferencia del método de hashing asistido por tabla, esta estructura contiene la dirección física de cada nodo.**
- **La secuencia de bits que retorna la función de hash, permite obtener de la tabla en memoria, la dirección física del nodo para almacenar la clave.**
- La tabla será usada posteriormente para recuperar cada registro en un solo acceso a disco.

# Hash Extensible

**El método de hash extensible comienza con un sólo nodo en disco y una tabla que solamente contiene una dirección, la del único nodo disponible.**



El número cero sobre la tabla inicial, indica que no es necesario usar ningún bit de la secuencia obtenida por la función de dispersión, para obtener la dirección física para almacenar el registro.

# Hash Extensible

Supongamos un nodo con capacidad para dos registros y la tabla muestra una secuencia de llaves a insertar en el archivo.

Clave	FH(Clave)
Alfa	00.....1001
Beta	00.....0100
Gamma	00.....0010
Delta	00.....1111
Epsilon	00.....0000
Rho	00.....1011
Pi	00.....0110
Tau	00.....1101
Psi	00.....0001
Omega	00.....0111

# Hash Extensible

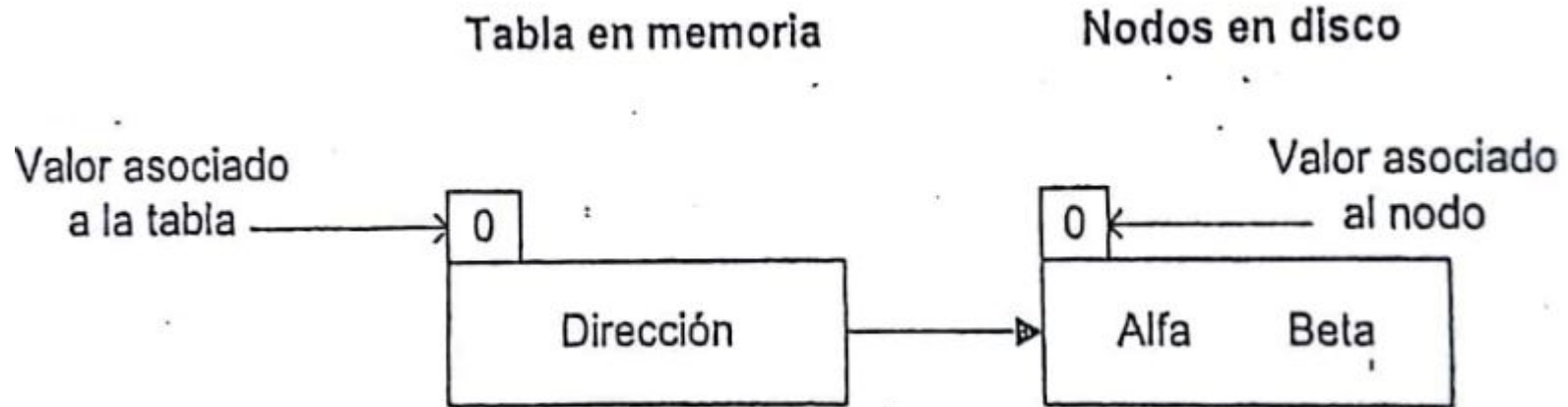
## ¿Cómo funciona el método para realizar las inserciones?

La primer clave a dispersar es Alfa, se calcula la función de hash y se toman tantos bits como dice el valor asociado a la tabla en memoria. En este caso, el valor es 0, esto indica que hay una única dirección de nodo disponible, y la clave debe insertarse en ese nodo.

Como no hay saturación (el nodo está vacío), Alfa se almacena en ese lugar.

Para insertar la clave Beta, se procede de la misma forma, dado que no se produce overflow.

# Hash Extensible

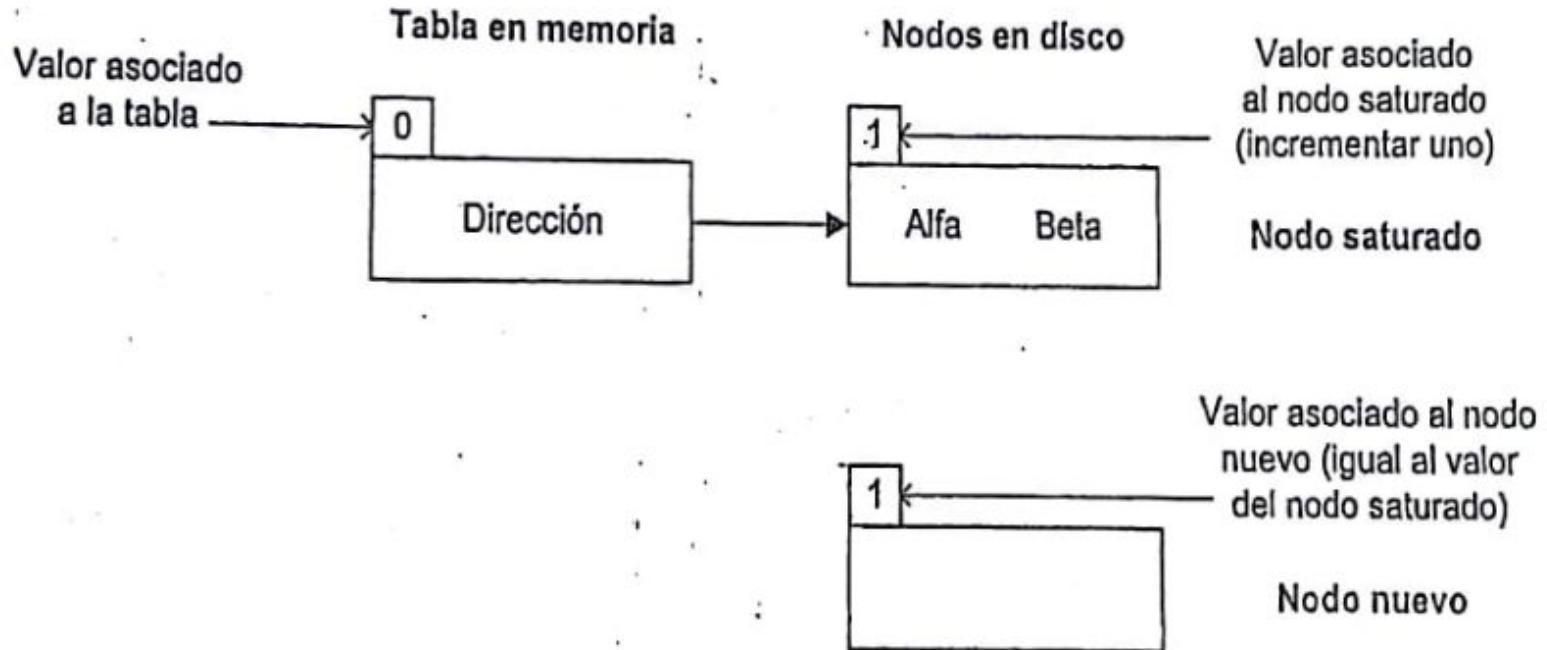


# Hash Extensible

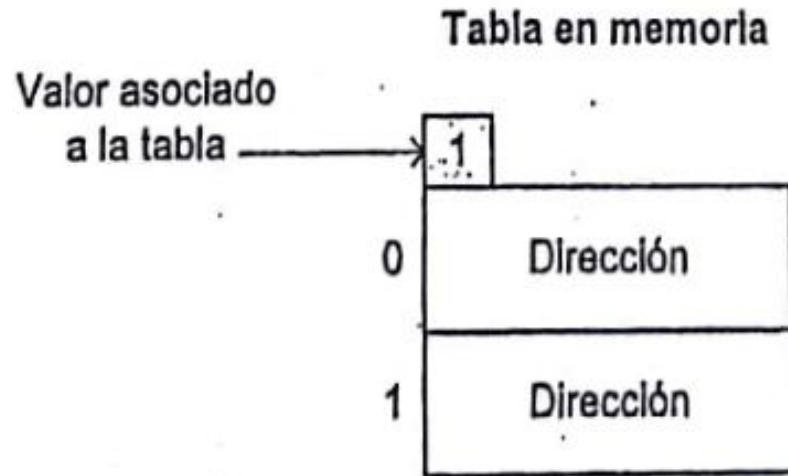
La tercera clave a dispersar es Gamma, se procede de la misma forma que la anterior, pero ahora se produce overflow, porque el nodo está lleno. Por lo que se se realizan los siguientes pasos:

- Primero se aumenta el valor asociado al nodo saturado a uno.
- Se genera un nuevo nodo con el mismo valor asociado que el nodo saturado.
- Se compara el valor asociado al nodo con el valor asociado a la tabla, como el primero es mayor que el segundo, significa que la tabla no dispone de entradas suficientes para direccionar al nodo nuevo.
- Se duplica la cantidad de celdas de la tabla y se incrementa a uno el valor asociado a la tabla.

# Hash Extensible



# Hash Extensible

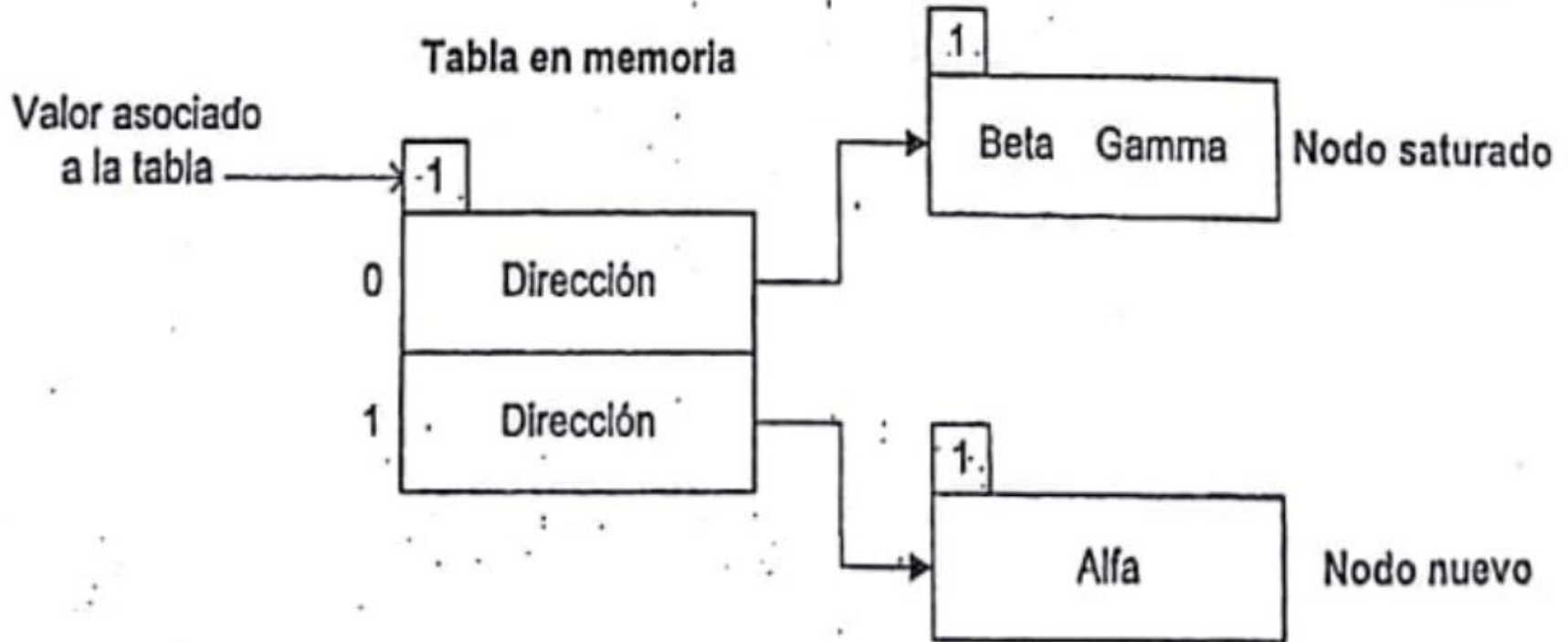




# Hash Extensible

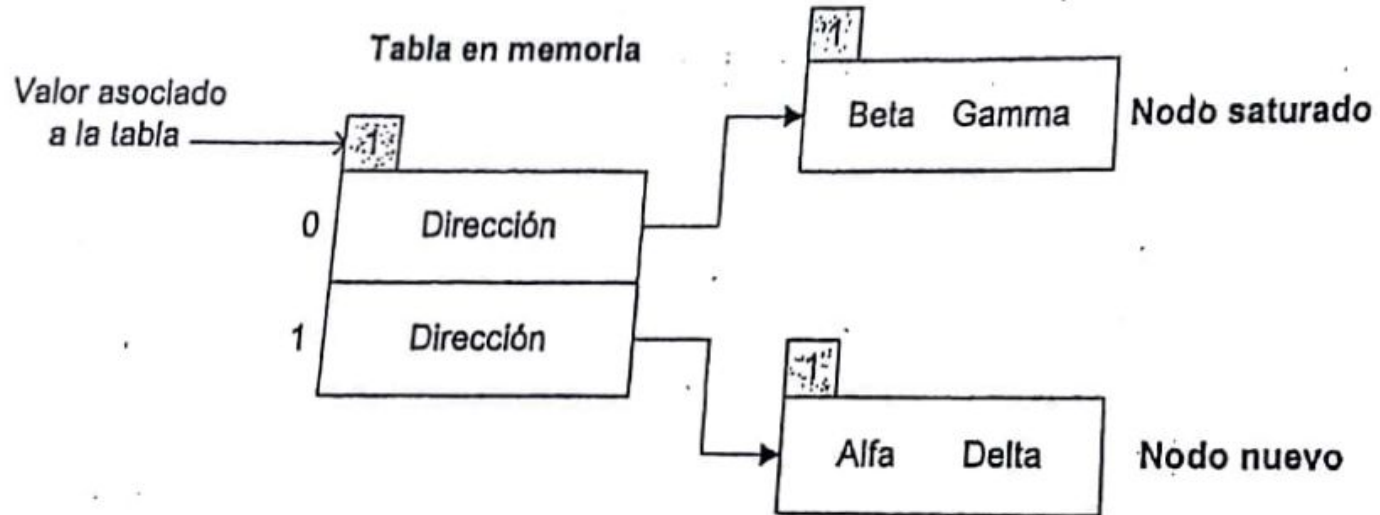
- El valor asociado de la tabla indica la cantidad de bits que es necesario tomar de la función de hash.
- La primera celda de la tabla direcciona al nodo saturado y la nueva celda apunta al nodo nuevo.
- Los elementos de la cubeta saturada (Alfa y Beta) más el elemento que generó la saturación (Gamma), son re-dispersados entre ambos nodos, de acuerdo al bit menos significativo que determina la función de hash.
- Beta y Gamma tienen cero, por eso van al primer nodo y Alfa tiene un uno, por lo que va al nodo nuevo (se ubican por número de celda de la tabla).

# Hash Extensible



# Hash Extensible

Luego se debe dispersar la clave Delta, que como tiene un uno en el bit menos significativo de la función de hash, al direccionar el nodo correspondiente al bit uno, no genera saturación, por lo que Delta se almacena junto a Alfa (siempre hay orden secuencial dentro del nodo).



# Hash Extensible

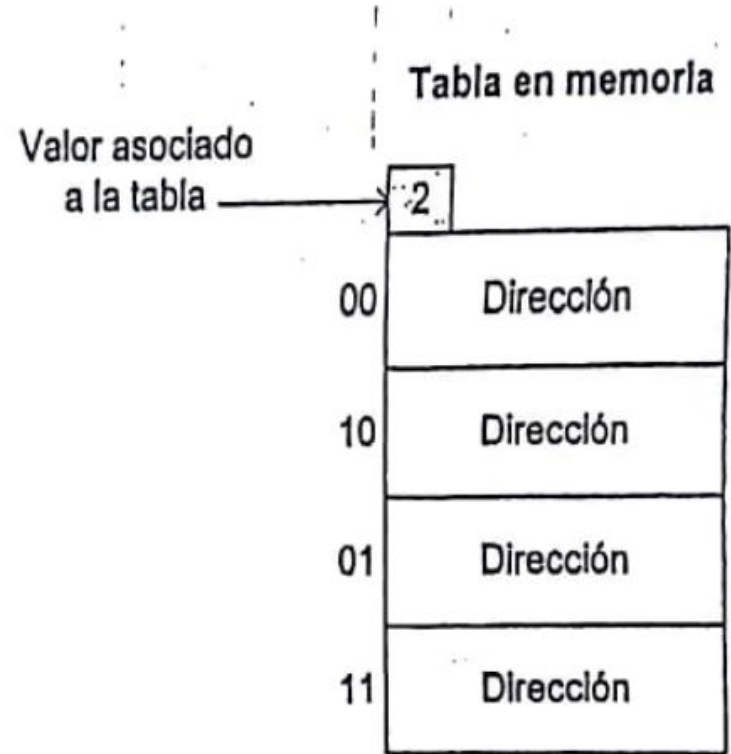
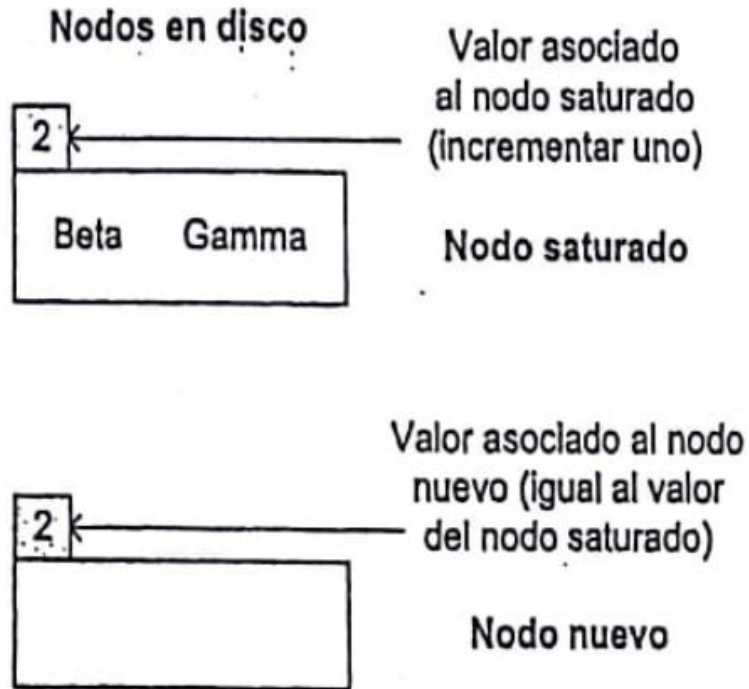
Epsilon, debe ser almacenada en el nodo asociado a la celda 0 de la tabla, como está completo, se genera overflow.

Al no disponer de celdas suficientes en la tabla en memoria, se duplica el espacio disponible, y a partir de este momento, se necesitarán 2 bits de la función de hash para poder direccionar un registro.

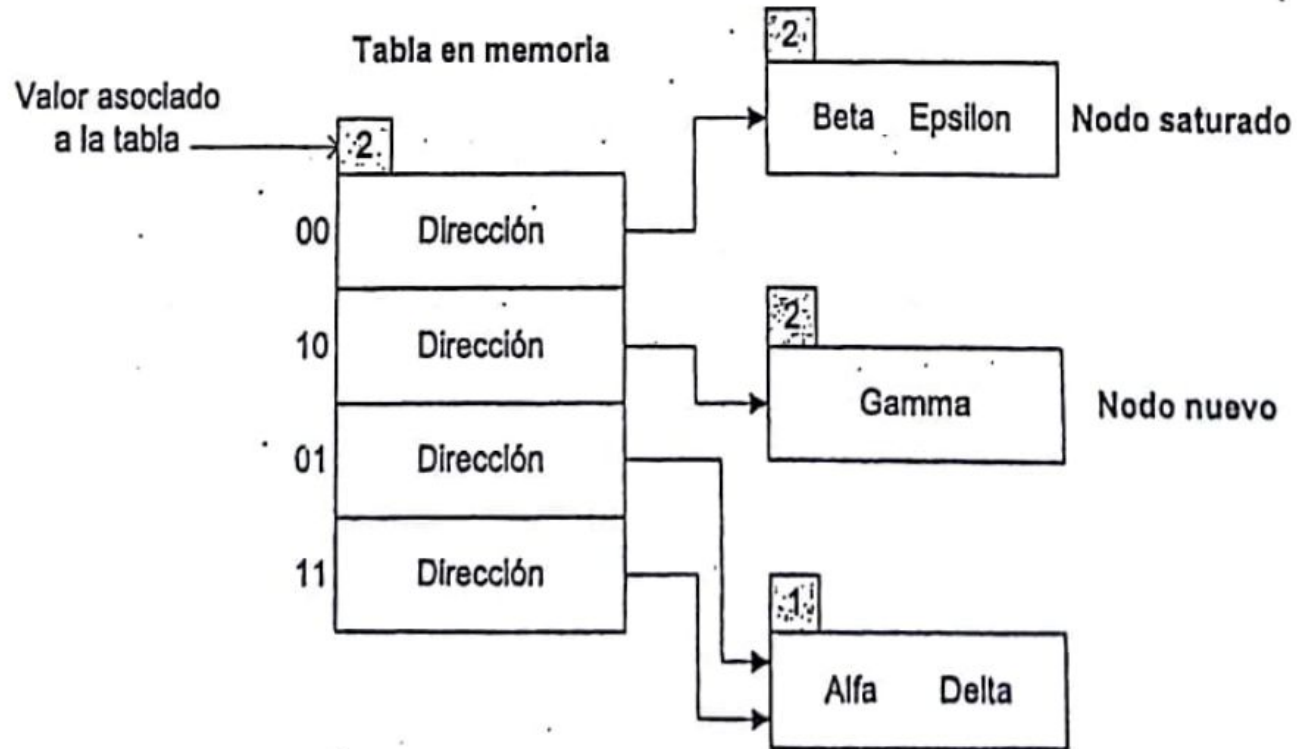
La celda de referencia 00 contiene la dirección del nodo saturado, en tanto que la celda de referencia 10 contiene la dirección del nuevo nodo.

Los registros Beta, Gamma y Epsilon son reubicados en función de sus 2 bits menos significativos. Se debe notar que el nodo asociado a las celdas de referencia 1, ahora es referenciado por dos celdas de la tabla, las correspondientes a 01 y 11.

# Hash Extensible



# Hash Extensible



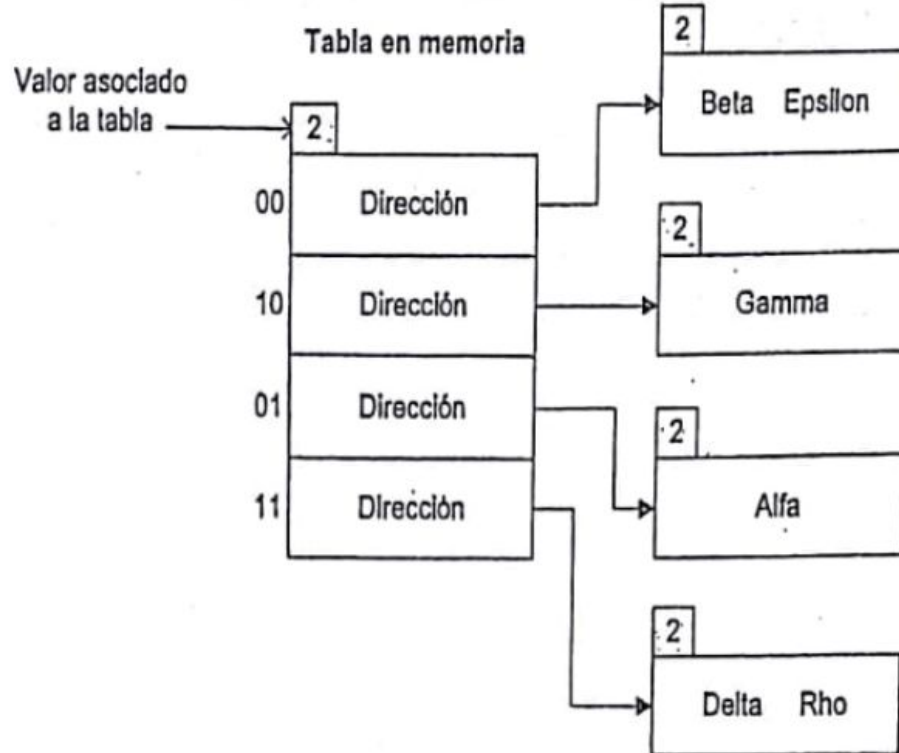
# Hash Extensible

La siguiente clave a insertar es Rho, su dirección de almacenamiento corresponde al nodo asociado a la celda 11. Dicho nodo está completo por Alfa y Delta, por lo que se genera saturación y se debe crear otro nodo.

Se incrementan a 2, los valores asociados del nodo en saturación y el nodo nuevo.

Ahora cada valor asociado a un nodo, se corresponde con el valor asociado al nodo de la tabla. Lo que significa que la tabla tiene direcciones suficientes para direccionar al nuevo nodo, por lo tanto, la cantidad de celdas no debe ser duplicada. Se re-acomodan los punteros de la tabla a los nodos.

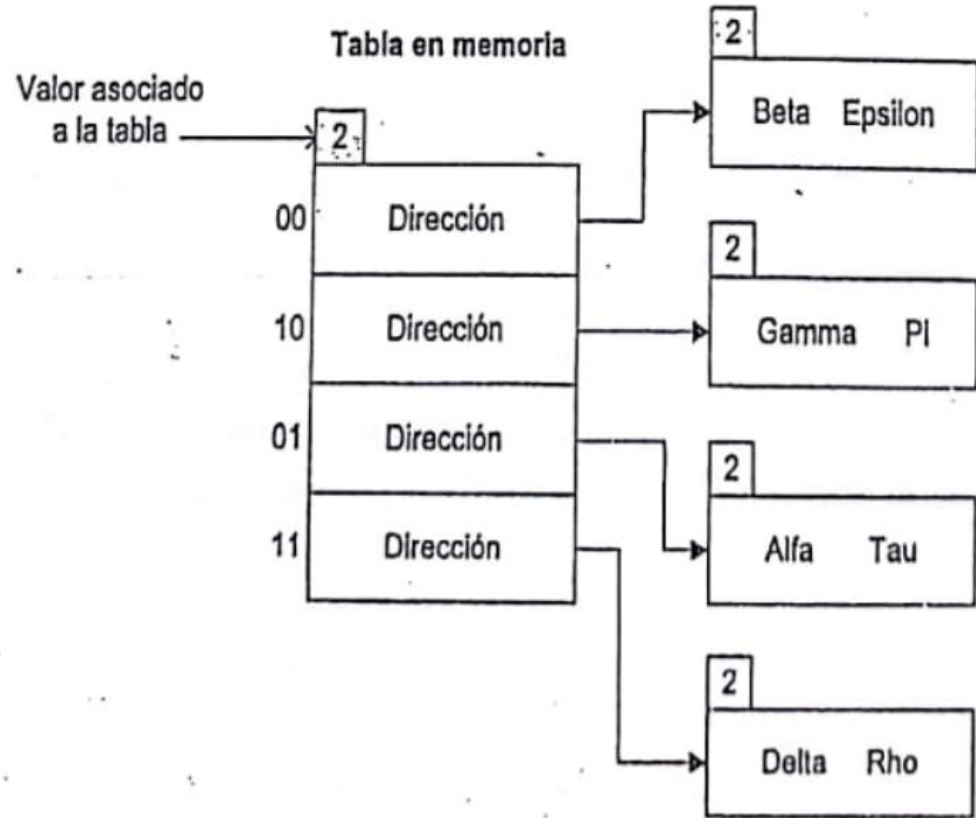
# Hash Extensible





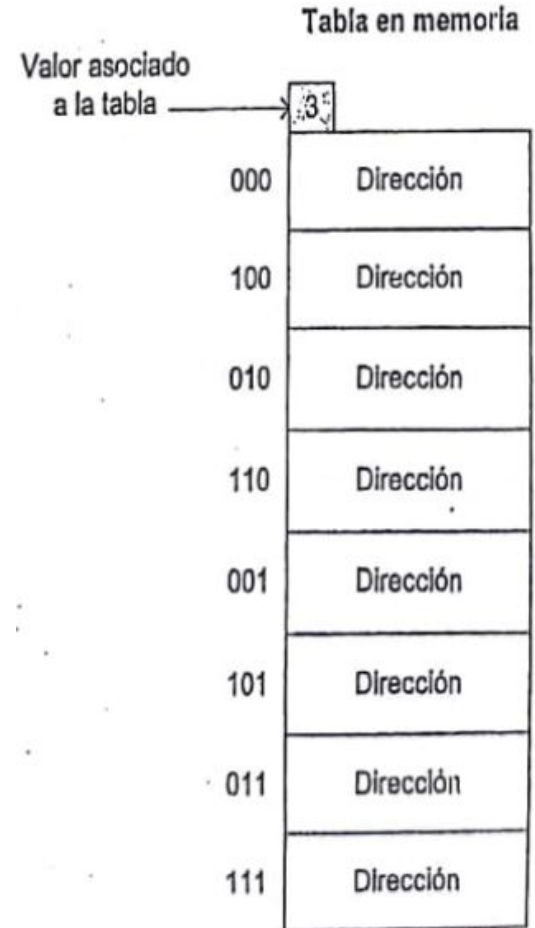
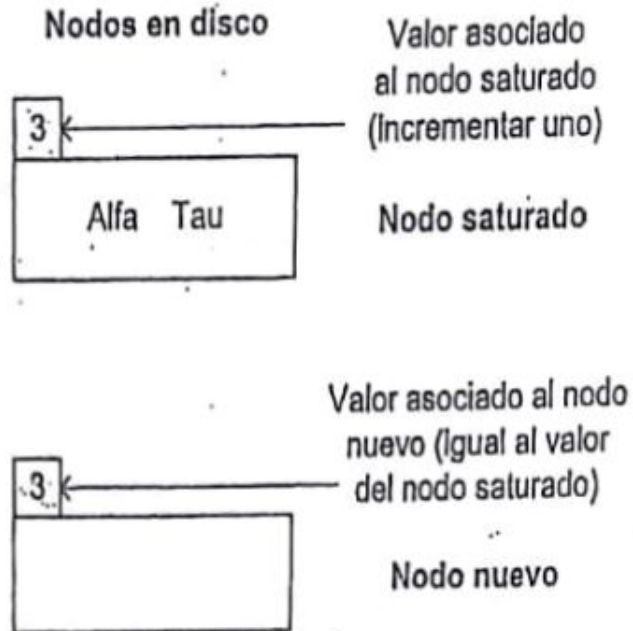
# Hash Extensible

Luego viene la clave Pi, cuya función de hash retorna en sus dos bits menos significativos 10, y el nodo correspondiente tiene lugar, por lo que no se produce overflow, y se inserta sin problemas. Lo mismo ocurre con la clave Tau, en el nodo correspondiente a la celda 01.

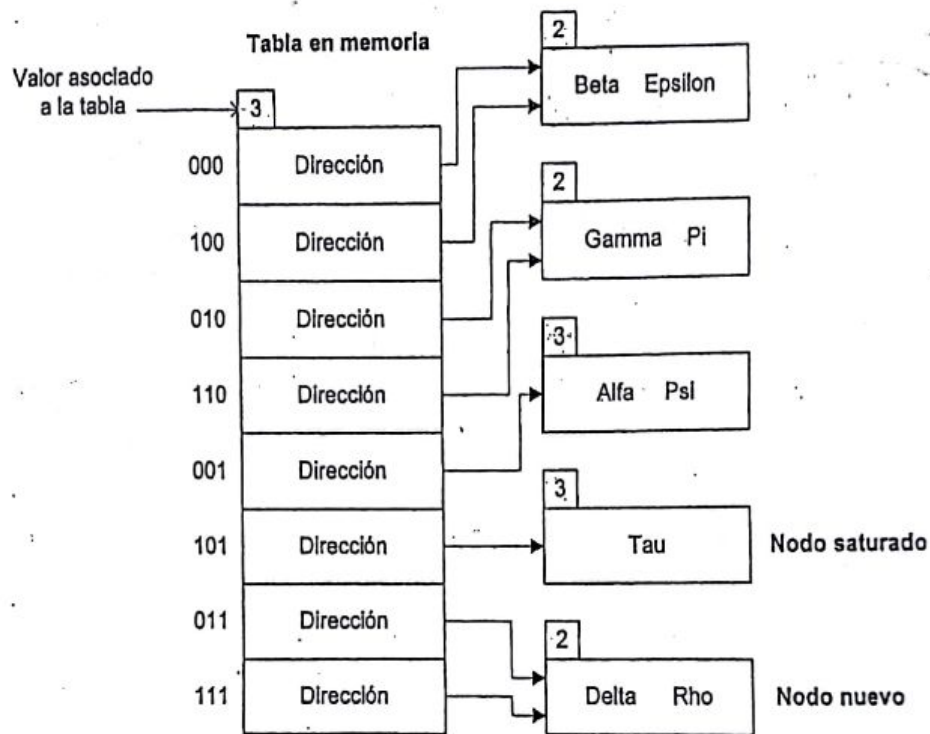


# Hash Extensible

La llave Psi se direcciona al nodo correspondiente a la celda 01, en el cual se produce desborde.

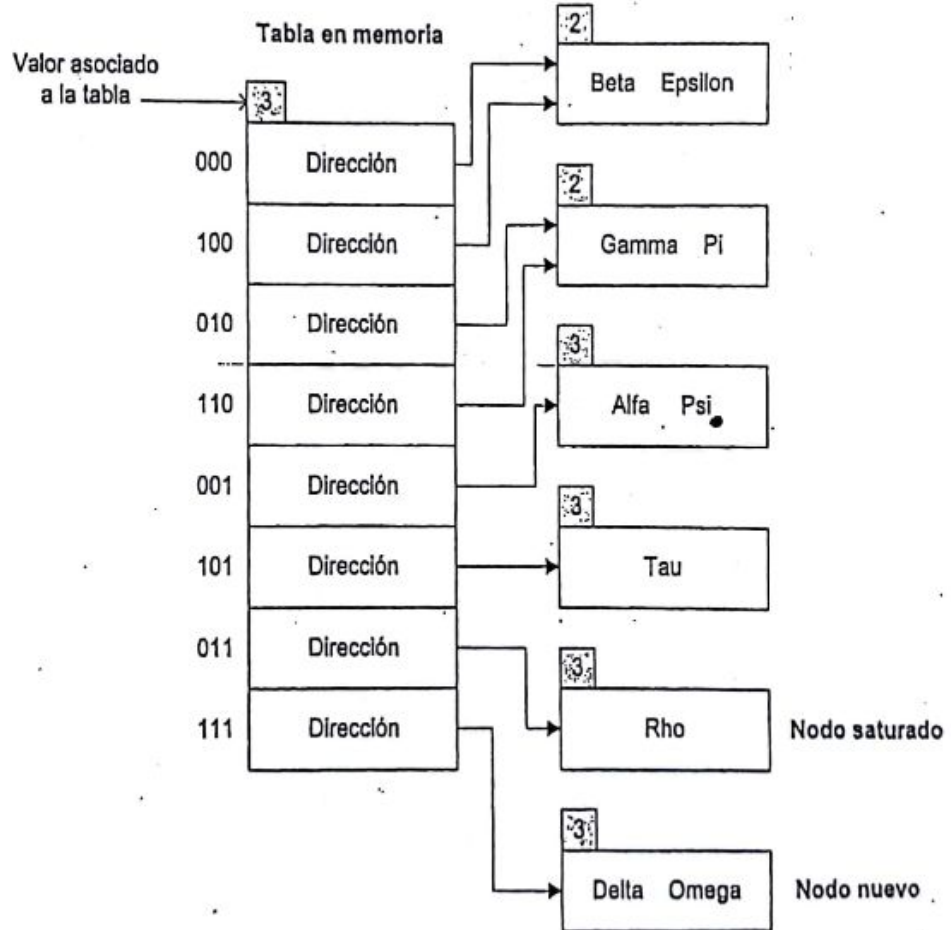


# Hash Extensible



# Hash Extensible

La última llave es Omega, se direcciona al nodo correspondiente a la celda 111, nuevamente hay saturación.



# Hash Extensible

Resumiendo, el **método de hash extensible trabaja según las siguientes pautas:**

- Se **utilizan sólo los bits necesarios** de la función de hash de acuerdo con cada instancia del archivo.
- Los bits tomados definen la dirección del nodo que se utilizará.
- Si se intenta insertar en un nodo lleno, deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nodo nuevo, para ello se usa un bit más.
- La tabla tendrá tantas entradas (direcciones de nodos) como  $2^n$ , siendo  $n$  el número de bits actuales del sistema.

# Hash Extensible

## Búsqueda

El proceso de búsqueda **asegura encontrar cada registro en un sólo acceso.**

Se calcula la secuencia de bits para la llave, se toman tantos bits de misma como indique el valor asociado a la tabla, y la dirección del nodo contenida en la celda respectiva debería contener al registro buscado.

En caso de no encontrar el registro, significa que no está en el archivo, esta búsqueda secuencial dentro del nodo, se realiza en memoria.

