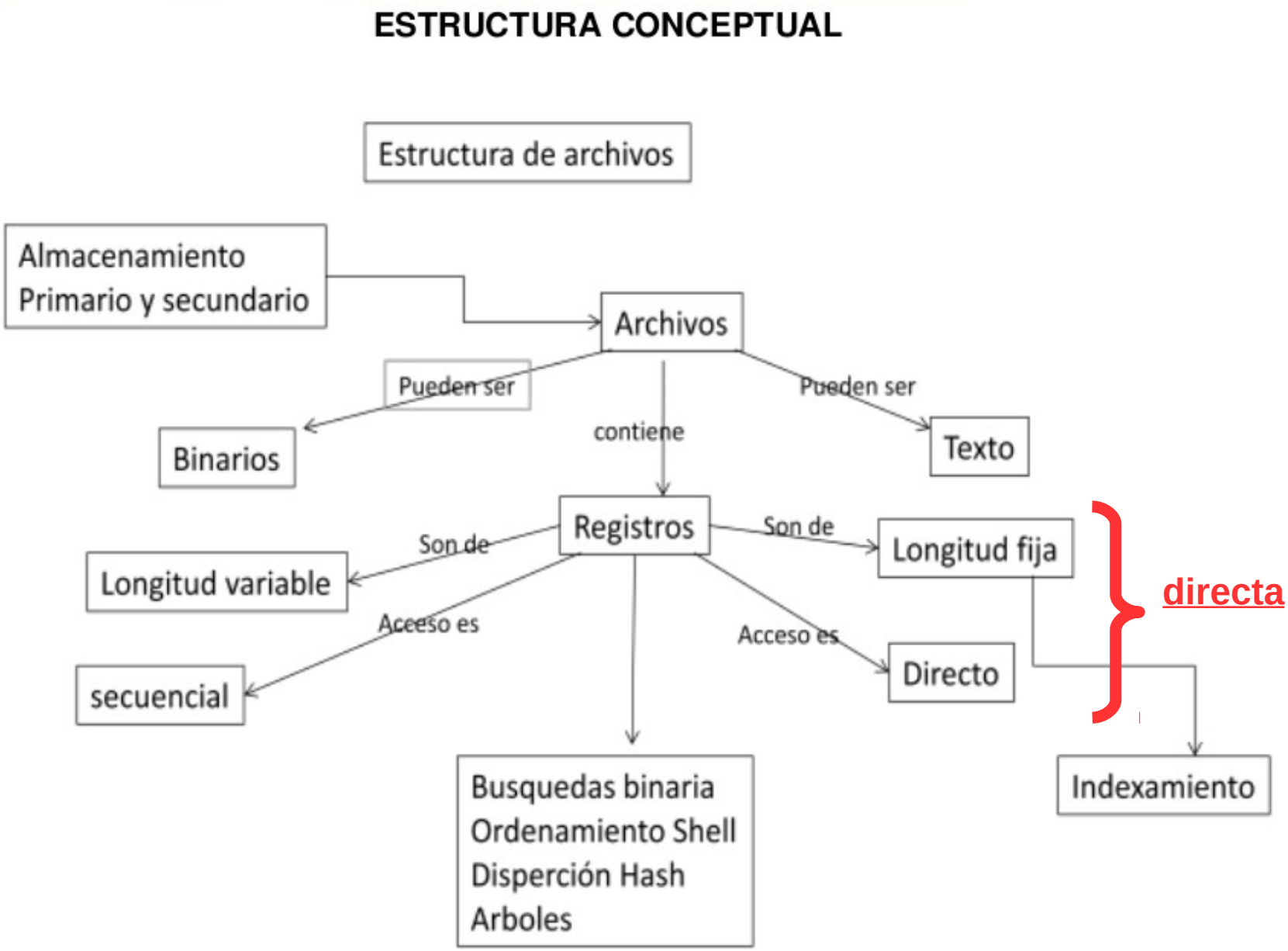
Organización Directa



Organización Directa

Organización de Archivos Directos

Acceso Directo

El acceso directo es el que permite acceder de manera rápida y simple a los registros de un archivo. Se debe aclarar que la secuencia u ordenamiento lógico de los registros no tiene, necesariamente, una relación con la secuencia física.

Pos. Física	Clave
0	20438705
1	40436878
2	nulo
3	nulo
4	48878956
5	54567896
N	nulo

Dirección Física=Dirección Base + Tamaño del Registro * (Valor Clave -1)

La forma de acceder a los registros es a través de la clave de dicho archivo. La clave o llave principal es el campo o la combinación de campos del archivo que permiten identificar o diferenciar plenamente cada registro de los demás.

Organización de Archivos

Organización de Archivos Directos

Organización Directa o Relativa

Pos. Física	Clave
0	20438705
1	40436878
2	nulo
3	nulo
4	48878956
5	54567896
N	nulo

Para este tipo de organización, es conveniente que El medio de almacenamiento permita acceso directo.

La memoria principal y los discos magnéticos son la mejor opción. Las cintas son la peor.

Complejidad: *Programar la relación existente entre el contenido de un registro y su posición física.*

Suele suceder que existan huecos libres dentro del medio magnético, y por lo tanto pueden existir huecos libres entre registros.

Registro Lógico Vs. Registro Físico

Organización de Archivos

Organización de Archivos Directos

Características que permitan acceso Directo:

1. *El conjunto de claves debe tener un orden ascendente, con pocos valores no utilizados, "bajar el desperdicio"*

2. *Ideal: La clave de los registros corresponden con los números de las direcciones.*

3. *Existe una dirección de almacenamiento en el archivo por cada valor posible de la clave, y éstas no tiene valores duplicados.*

Pos. Física	Clave
0	20438705
1	40436878
2	nulo
3	nulo
4	48878956
5	54567896
N	nulo

Organización de Archivos

Organización de Archivos Directos

Características que permitan acceso Directo:

1. *El conjunto de claves debe tener un orden ascendente, con pocos valores no utilizados, "bajar el desperdicio"*
2. *Ideal: La clave de los registros corresponden con los números de las direcciones.*
3. *Existe una dirección de almacenamiento en el archivo por cada valor posible de la clave, y éstas no tiene valores duplicados.*
4. *Los valores de las claves están en un rango acotado.*

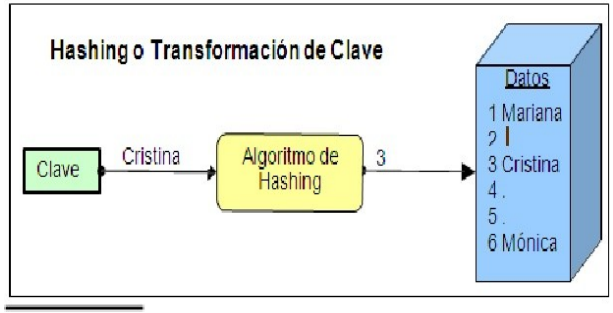
Pos. Física	Clave
0	20438705
1	40436878
2	nulo
3	nulo
4	48878956
5	54567896
N	nulo

Problema: *Si las claves de un registro no representan una dirección hay que utilizar:*

Hashing*; Granularidad y posible Overflow.*

Organización Directa

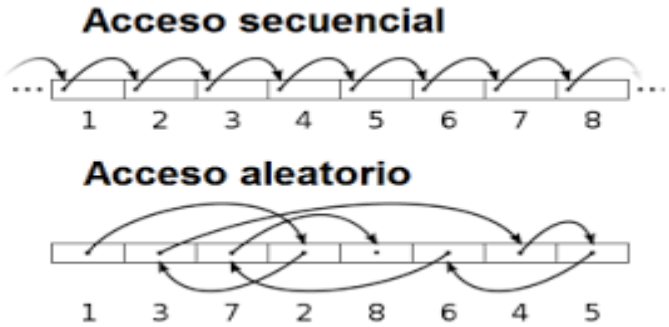
Características:



- El archivo tiene longitud fija y Los no extensible. Registros tienen longitud fija y tienen una clave primaria.
- Cada registro del archivo posee un atributo extra que indica si el mismo está "libre", "ocupado", o fue "borrado" del archivo.
- Para realizar altas, bajas y modificaciones de registros en el archivo, se debe calcular su posición en el mismo con una función de hashing. Si dos registros hashean a la misma posición, la colisión debe resolverse.
- Factor de carga del archivo:
  - (#registros_ocupados/#registros_disponibles)

Organización Directa

Antecedentes y Motivación:



- **Costos promedio de acceso en un Archivo de N ejemplo 10000 registros:**
  - **Secuencial desordenado N/2 (5000).**
  - **Secuencial Ordenado Log2(N) (13).**
  - **Arboles Balanceados de orden d (carga promedio de nodos [3/4*d]-1 para un d=16 menos de 3.**
- **¿Es posible mejorar el costo de acceso?**
- **¿Y si se prueba con una función de mapeo de claves de registros a direcciones?**

Concepto de Disperción

Algoritmos de Dispersión (hashing)

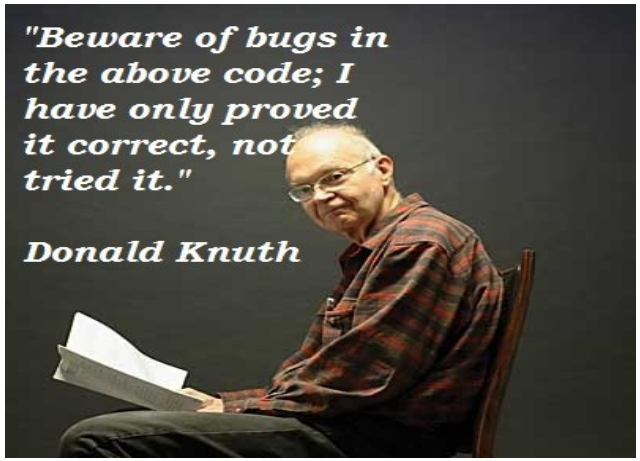
- **Organización indexada**
  - Mantener un índice (claves + referencias)
  - Búsqueda en índice para acceso al dato
- **Objetivo:**

  Reducir al máximo el coste de las operaciones de búsqueda, a partir de la clave: idealmente un sólo acceso, y eliminar el índice.
- **Solución:** Diseñar una función (cálculo “simple”), h, de transformación de la clave en una referencia física
- **Organización Dispersa:**
  - No se almacenan las claves en un indice.
  - El acceso al dato a través función de dispersión aplicada a la clave = h(clave $_i$) o dirección Dato $_i$.

Organización Directa

Que significa HASHING?

- El término hash proviene, aparentemente, de la analogía con el significado estándar (en inglés) de dicha palabra en el mundo real: picar y mezclar.
- Donald Knuth cree que H. P. Luhn, empleado de IBM, fue el primero en utilizar el concepto en un memorándum fechado en enero de 1953.
- Su utilización masiva no fue hasta después de 10 años.



Concepto de Disperción

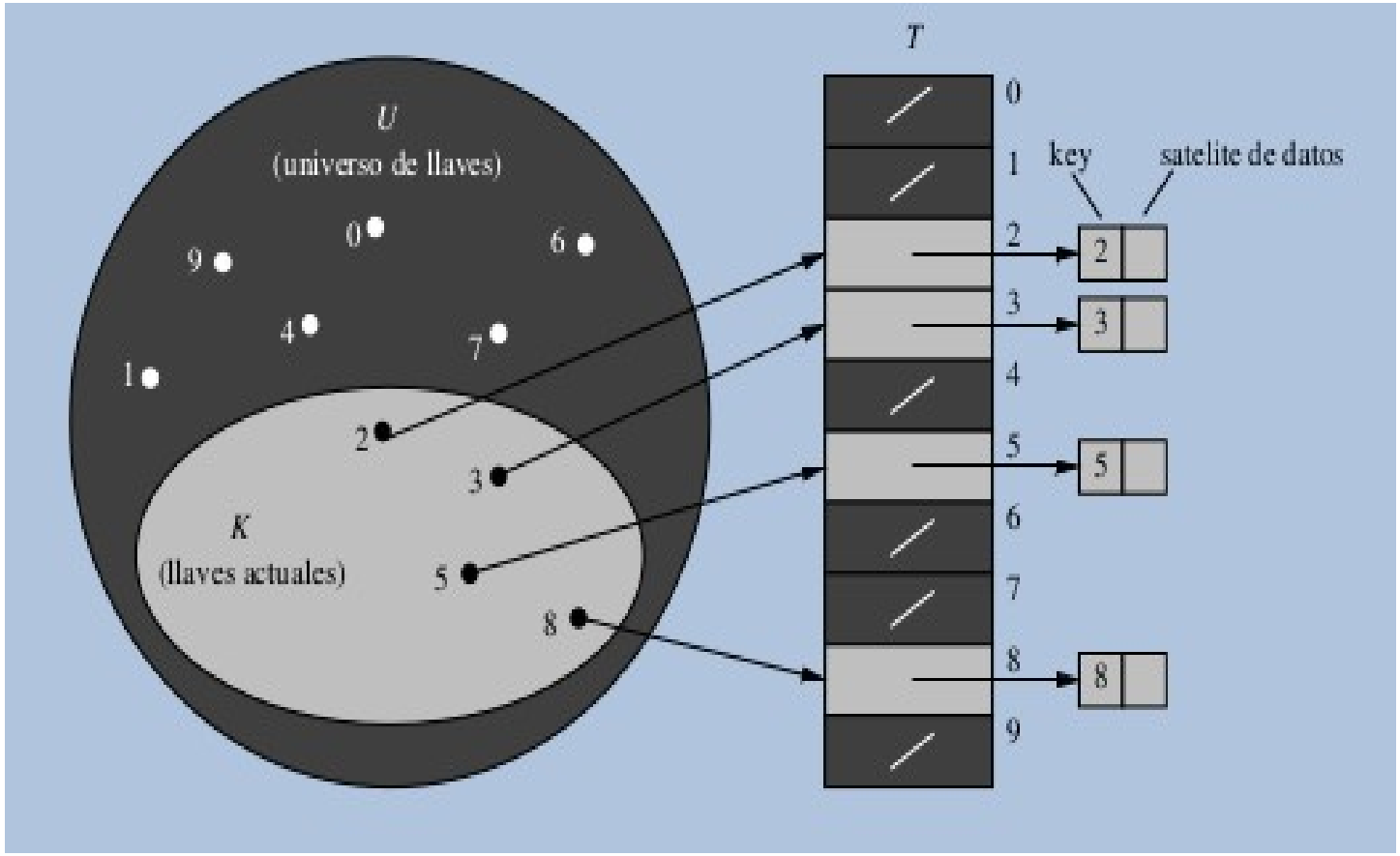
Que es la Dispersión (hashing)

- **Técnica para generar una dirección base única para una clave dada. La dispersión se usa cuando se requiere acceso rápido a un registro.**
- **Técnica que convierte la clave del registro en un número aleatorio, el que sirve después para determinar donde se almacena el mismo.**
- **Técnica de almacenamiento y recuperación que usa una función de dispersión (Hash) para mapear registros en direcciones de almacenamiento.**

Concepto de Disperción

Que es la Dispersión (hashing)

- **Para representar conjuntos din´amicos, se usa un arreglo o un tabla de direccionamiento directo T [0..m − 1], en la cual cada posici´on, o slot, corresponde a una llave en el universo de llaves U .**



Concepto de Disperción

Tablas de Direccionamiento Directo

- **Las operaciones son triviales de implementar:**
  - **Direct-Address-Search(T ,k)**
    return T [k]
  - **Direct-Address-Insert(T ,x)**
    T [key[x]] ← x
  - **Direct-Address-Delete(T ,x)**
    T [key[x]] ← nil
  - **Cada una de la operaciones es r apida: solo O(1) es requerido.**

Dispersión Hashing

CONCEPTOS BÁSICOS

- Las Direcciones de los registros físicos son Números.
- En la gran mayoría de los casos imposible de recordarlos. Por asociación.
- Utilizamos como claves DNI, Numero de legajo, otros Identificadores numéricos
  - Problema Si cada numero lógico es un registro fisico.
    - Puffff Cuanto desperdicio.
- Utilizamos Palabras como claves, ejemplo Apellido y Nombres, Nombre Producto, etc.
    - Hay que convertirlos a Numero o traducirlos.

Dispersión Hashing

CONCEPTOS BÁSICOS

- **Las tablas hashing constituyen un TAD especialmente indicado para la manipulación y almacenamiento de la información en memoria secundaria.**
- **Cualquier tabla es una asociación de celdas.**
- **Las tablas hashing son tablas que relacionan claves con posiciones de memoria donde se almacenan atributos.**
- **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 Hashing

CONCEPTOS BÁSICOS

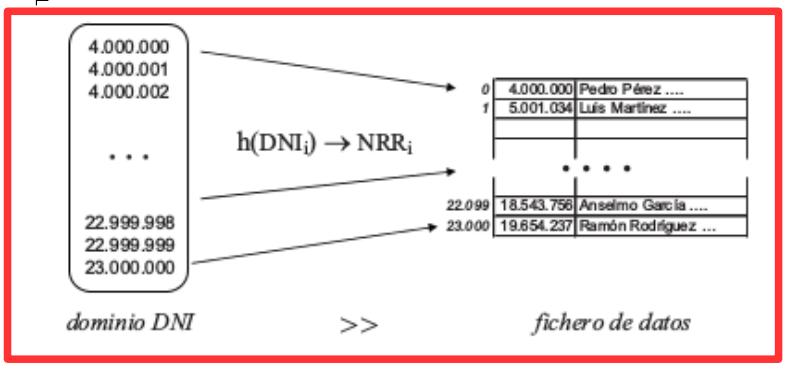
- **Problema**:
  - **Cómo asignar una clave de todas las posibles a una dirección de memoria.**
  -
- **Solución**:
  - **Una función de transformación. Pero, ¿cómo es esa función?**
  -
- **Respuesta**:
  - **Las tablas hashing se aplican cuando el conjunto de claves posibles es mucho mayor que el de claves reales a almacenar.**

Dispersión Hashing

CONCEPTOS BÁSICOS

- **Definición**: Dado un conjunto de claves posibles, X, y un conjunto de direcciones de memoria, D, una función de transformación, H(x), es una aplicación suprayectiva del conjunto de claves posibles en el conjunto de direcciones de memoria.

$$H: X \rightarrow D$$



- **El TAD tabla de dispersión:**
  - Tipo de datos homogéneo de tamaño fijo.
  - Ttabla, compuesto por un número fijo de componentes, denominadas celdas.
  - Se accede mediante una dirección de memoria resultante de una función de transformación.
  - Sobre este TAD se definen los operadores:

- **Insertar, Buscar y Eliminar.**

Dispersión Hashing

CONCEPTOS BÁSICOS

- **Desbordamiento:**
  - **Se dice que se ha producido un desbordamiento cuando una nueva clave se aplica a una dirección de memoria completamente ocupada.**
- **Colisión:**
  - **Se dice que se ha producido una colisión cuando dos claves distintas se aplican sobre la misma celda.**
- **Caso Particular:**
  - **En el caso habitual en el que una celda contiene un único registro el desbordamiento y la colisión se producen simultáneamente.**

Dispersión Hashing

CONCEPTOS BÁSICOS

- **Densidad de Claves:**
  - **Se denomina densidad de claves al cociente entre el número de claves en uso, m, y el número total de llaves posibles, n x .**
- **Factor de Carga o Densidad de Carga α:**
  - **Se denomina al cociente entre el número de claves en uso y el número total de registros almacenables en la tabla de dispersión.**

S: número de registros por bloque

B: número de bloques que hay en la tabla de dispersión.

$$\alpha = \frac{m}{s \cdot b}$$

000	00077641	00034567
001	00131178	00139179
549	54913110	54939189
999	99911011	99931419

Dispersión Hashing

Funciones Hash

- **Características:**
  - **a) Fáciles de calcular.**
  - **b) El valor que calcula depende de la clave.**
  - **c) Producir el menor numero de colisiones porque hace que la ocupación de los bloques sea Equiprobable.**
- **Problema de las Funciones:**
  - **Funciones uniformes que dividen el conjunto de claves posibles en grupos de claves aproximadamente iguales.**
- **Métodos de construcción:**
  - **Dada a características distintas de las posibles claves, Existen distintos Métodos de construcción Hash.**

Funciones Hashing

Métodos de construcción

- **Resto de División o Modulo.**
  - **La clave se divide entre el número de direcciones (debe tratarse de que sea número primo, pues tiende a distribuir residuos en forma más eficiente)**
- **Multiplicación.**
  - **Se multiplica la llave por una constante y luego se multiplica por m.**
- **Plegado FOLD & ADD:**
  - **Dividir la clave en partes iguales y sumarlas. La suma de las partes puede realizarse de dos formas:**
    - **Por Desplazamiento**
    - **Por Fronteras.**

Funciones Hashing

Métodos de construcción (Cont.)

- **Compresión:**
  - **Dividir la clave en componentes, traducir su número de cotejo a binario y aplicar la operación XOR y al resultado la operación resto de división entera.**

- **Extracción:**
  - **Método de la mitad del cuadrado: Si tenemos claves numéricas (si no las transformamos), calculamos su cuadrado y nos quedamos con algún número de la zona central del resultado. Nos quedamos con tantos dígitos como necesitemos para mapear el array.**

- **Otros ..**

Existen Otros métodos de Generar una Hash.

Funciones Hashing

MÉTODO Resto de División o Modulo

- **Calcular el Resto de la División de la Clave por un numero M.**

$$f(x) = X \bmod M.$$

a) Cuidar el elegir M
b) Espacio de direcciones: [0, M-1]
c) La recomendación es que M sea un Primo >M

- **Por ejemplo, si la tabla hash tiene tamaño m = 12 y la llave es k = 100, entonces h(k) = 100 mod 12 = 4.**

- **Cuando se usa el método de la división, usualmente se evitan ciertos valores de m. Por ejemplo, m no debe ser una potencia de 2, ya que si $m = 2^p$, entonces h(k) es igual a los p bits de más bajo orden de k**

Funciones Hashing

MÉTODO Multiplicación

- **El método de la multiplicación para crear funciones hash opera en dos pasos.**

  1) Se multiplica la llave k por una constante A en el rango 0 < A < 1 y se extrae la parte fraccionaria de kA

  2) Se multiplica este valor por m y se toma el piso del resultado

  La función hash es:

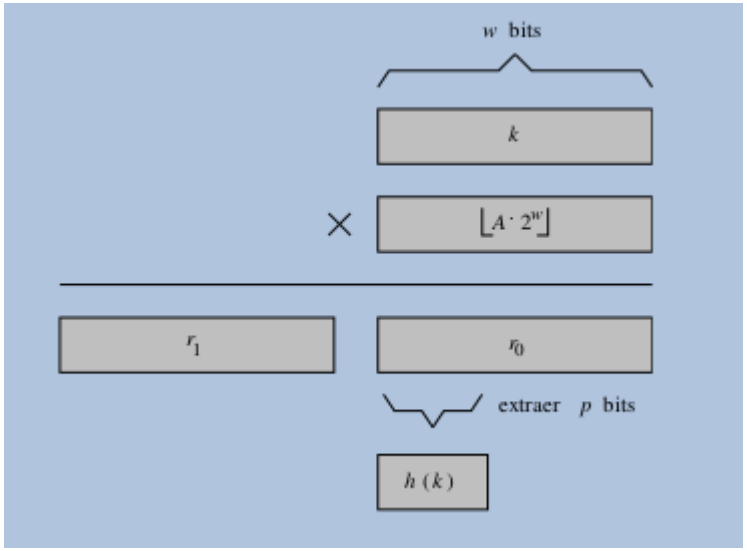
$$h(k) = m(k\ A \bmod 1)$$

- **donde "k A mod 1" significa la parte fraccionaria de kA, que es, kA − kA**

Funciones Hashing

MÉTODO Multiplicación

- La ventaja del método de la multiplicación es que el valor de m no es crítico.
- Comúnmente se escoge como una potencia de 2, m = 2^p para un entero p.



- Suponiendo que el tamaño de palabra en la máquina es w bits y que k cabe en una palabra, se multiplica primero k por el entero A [2^w] . El resultado es un valor $r_1 2^w + r_0$, donde r_1 es la palabra de m´as alto orden del producto y r_0 es la palabra de m´as bajo orden del producto.

Funciones Hashing

MÉTODO Multiplicación

- **Aunque este método funciona para cualquier valor de A, se ha encontrado que función mejor con algunos valores.**

- **Donald E. Knuth sugiere que:**

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887...$$

  - **Como un ejemplo, si se tiene k = 123456, m = 10000, y A es como en la ecuación 2, entonces:**

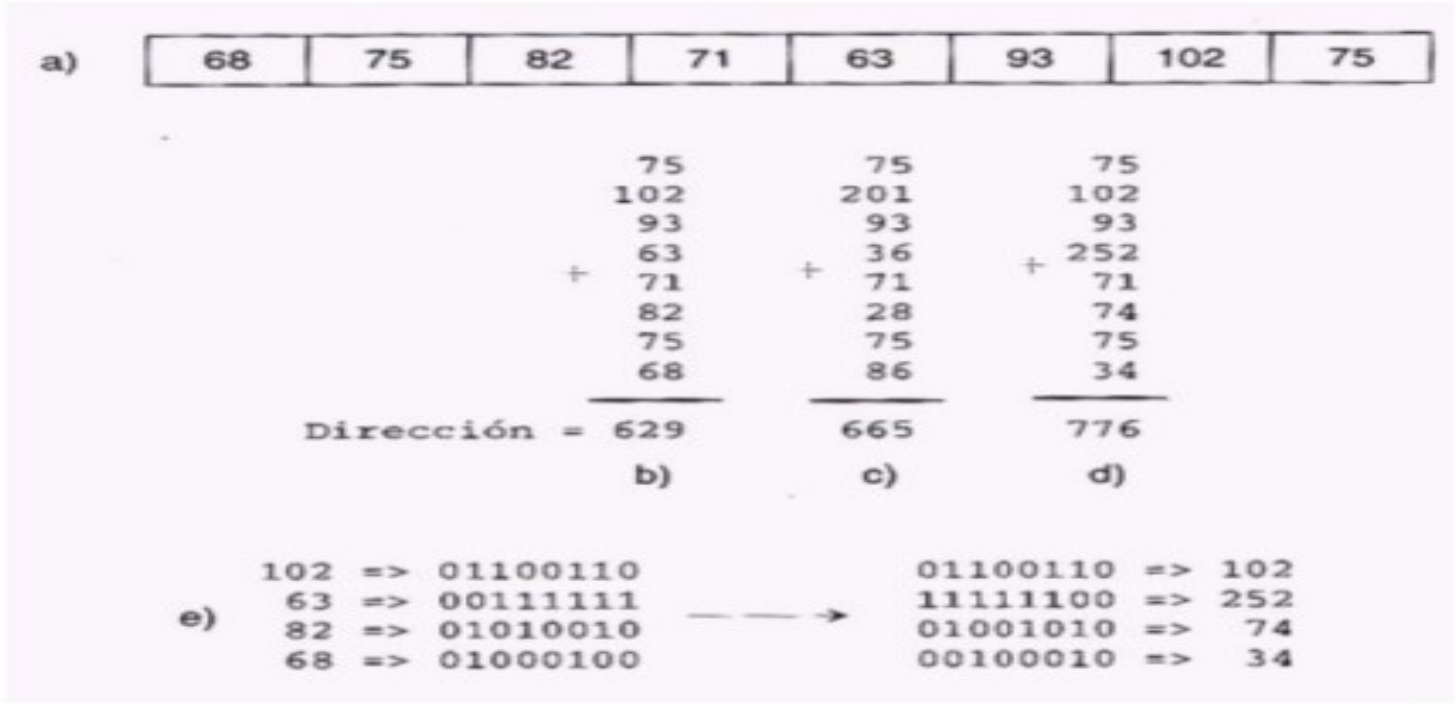
$$\begin{aligned} h(k) &= \lfloor 10000 \cdot (123456 \cdot 0.601803... \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot (76300.0041151... \bmod 1) \rfloor \\ &= \lfloor 10000 \cdot 0.0041151... \rfloor \\ &= \lfloor 41.151... \rfloor \\ &= 41. \end{aligned}$$

Funciones Hashing

MÉTODO Plegado Fold and Add

- Las claves es una cadena de caracteres, los dígitos de cada carácter por el valor decimal de la secuencia (ASCII, EBCDIC, etc.)
- Dividir la clave en partes iguales y sumarlas.
- El plegado por desplazamiento consiste en sumar las partes directamente.
- El plegado por las fronteras consiste en plegar el identificador por las fronteras de las partes y sumar los dígitos coincidentes.

Dispersión Hashing



a) Cadena de ocho caracteres representada por los números de orden dentro de la secuencia de cotejo correspondiente.

b) Plegado por desplazamiento.

c) Plegado en las fronteras en base decimal.

D)Plegado en las fronteras en base binaria.

e) Detalle del plegado en base binaria.

Dispersión Hashing

Ejemplo de Dividir la clave en partes iguales y sumarlas.

Ejemplo: el valor de la llave es 123456789 para una dirección relativa de 4 dígitos.

a) Partición

1	2345	6789

b) Plegado y suma

1
2345
9876

13221

d) El dígito de alto orden se trunca: La dirección relativa es 3221

Otros ejemplos:

Valor llave	dirección relativa	
123456789	3221	
987654321	8999	
555555555	6110	
000000472	2740	
117400000	2740	**colisión**
027000400	2740	**colisión**

Dispersión Hashing

MÉTODO Compresión

- **Dividir la clave en componentes, traducir su número de cotejo a binario y aplicar la operación XOR y al resultado la operación resto de división entera.**

- Ejemplo: C = Cadena de caracteres.

  C = 'David'

  'D' = 01100100
  'A' = 01100001
  'V' = 01110010
  'I' = 01101001
  'D' = 01100100

  XOR

  01111010 = 122 mod 26 = 18

Dispersión Hashing

MÉTODO EXTRACCIÓN
Cuadrado Medio

- **Método de la mitad del cuadrado: Si tenemos claves numéricas (si no las transformamos), calculamos su cuadrado y nos quedamos con algún número de la zona central del resultado. Nos quedamos con tantos dígitos como necesitemos para mapear el array.**

Ejemplo: Array con N = 0..99

C = 340 i^2 = 115600 H (C) = 56

C = 521 i^2 = 271441 H (C) = 14

Dispersión Hashing

MÉTODO EXTRACCIÓN
Cuadrado Medio

- **Ejemplo del Método Cuadrado Medio:**

Valor llave	llave al cuadrado	dirección relativa
123456789	15241578750190521	8750
000000472	00000000000022784	0000 **colisión**
117400000	13782700000000000	0000 **colisión**

Dispersión Hashing

Especificación TAD Hashing

Informal

TAD TablaHash (VALORES: Atributos almacenados en las posiciones resultado de una transformación Hash de claves; OPERACIONES: Inicializar, Hash, Buscar, Insertar, Eliminar)

Inicializar → TablaHash

Efecto: Crea una tabla hash vacía.

Hash (Clave) → Posicion

Efecto: Devuelve la transformación adecuada a partir de la clave.

Buscar (TablaHash, Clave) → Atributo, Boolean

Efecto: Si la clave está en la tabla, devuelve el atributo correspondiente y un valor lógico llamado error a false. En caso contrario error valdrá true.

Excepción: Error si la clave no está en la tabla.

Insertar (TablaHash, Clave, Atributo) → TablaHash, Boolean

Efecto: Si la posición no está libre, el valor lógico error vale true. En caso contrario en la posición Hash (clave) se almacena el par (clave, atributo).

Excepción: Error si hay colisión, desbordamiento o tabla llena.

Eliminar (TablaHash, Clave) → TablaHash, Boolean

Efecto: Si la clave existe en la tabla, se deja vacía esa posición y se actualiza error a false. En caso contrario, error vale true.

Excepción: Error si la clave no está en la tabla.

Dispersión Hashing

Función de hashing

Resumen:

Es una función f tal que f (k) = offset .

- Donde k es el **espacio de claves**
- Donde offset es el **espacio de direcciones**
- Claves sinónimas producen **colisión**: evento en el cual, para dos claves k1 y k2 (k1 != k2), se da que f (k1) = f(k2).
- Hashing perfectas → No producen colisiones. Se conoce perfectamente el espacio de claves.
- Hashing no perfectas/imperfectas → pueden producir coliciones.

Dispersión Hashing

Colisiones

- En el ejemplo:
  - Se obtiene el código ASCII de las 2 primeras letras de la llave Se multiplican dichos números
  - La dirección serán los 3 últimos dígitos de dicho resultado (esto para limitar nuestro archivo únicamente a 1000 direcciones)

Llave	*ASCII*	*Producto*	*Dirección*
BALL	66 65	66x65=4290	290
LOWELL	76 79	76x79=6004	004
TREE	84 82	84x82=6888	888

- Si quisiera insertar OLIVIA, ya tendria una colisión.
- Obviamente La hash es mala. Pero el "algoritmo de hash perfecto" NO existe.
- Hay que acostumbrarse a trabajar con colisiones.

Dispersión Hashing

Colisiones

- **colisiones se tienen algunas soluciones:**
  - **Propagar los registros:** **Buscar funciones que distribuyan muy aleatoriamente los registros podemos evitar agrupaciones" de llaves que produzcan las mismas direcciones.**
  - **Usar memoria extra:** **Poner mas espacio de direcciones posibles que registros a utilizar.**
  - **Colocar más de un registro en una dirección:** **Este concepto se basa en "buckets" o cubetas de datos en cada dirección, ahí se colocan algunos (casi todos) los registros que colisionan de manera que al hacer una búsqueda debemos recuperar la cubeta entera y ahí buscar por el registro deseado.**

Dispersión Hashing

Colisiones

- Direccionamiento Hashing Abierto o Hashing Cerrado
- sondeo lineal
  - En el que el intervalo entre cada intento es constante (frecuentemente 1).

$$Pi = f(k) + 1$$

- sondeo cuadrático
  - En el que el intervalo entre los intentos aumenta linealmente (por lo que los índices son descritos por una función cuadrática).

$$Pi = f(k) + i^2$$

- doble hasheo
  - En el que el intervalo entre intentos es constante para cada registro pero es calculado por otra función hash.

$$Pi = f(k) + i*1 + i^2$$

$$Pi = f(k) + i \cdot g(k)$$

Salto con m-1

Dispersión Hashing

Colisiones

Si dos valores de llave K1 y K2, son sinónimos aplicando H.
Suponga que K1 y K2 no tienen otros sinónimos.
Si K1 es primero almacenado su dirección es H(K1)
Entonces K1 está almacenado en su dirección de origen.

Existen dos métodos para Alojar K2:

Direccionamiento abierto:
En el cual otra dirección distinta de la dirección de origen
es encontrada para K2 en el archivo relativo. (Progresive Overflow)

Separación de desborde:
En el cual alguna dirección es encontrada para K2 fuera del área principal del archivo relativo. En un área especial de desborde, exclusiva para registros que no pueden ser asignados a su dirección de origen. (área de Overflow)

Dispersión Hashing

Distribución de un archivo de Hash

Es posible predecir cómo se van a distribuir los registros en un archivo, esto se hace a través de probabilidades:

Aplicando la distribución de poisson -->

$$p(x) = \frac{(r/N)^x \; e^{(-r/N)}}{x!}$$

Donde:

- P(x) que x registros con la misma dirección.
- N → número de direcciones posibles.
- R → número de registros a insertar.
- X → número de registros asignados a alguna dirección

Ejemplo: N=1000 y r= 1000

P(2)= 0.184 ; p(3)= 0.061; p(4)= 0.015

Ejemplo de Densidad: tenemos 100 espacios y 75 registros.

Densidad = 75/100 = 0.75

mientras más densa sea la relación mayor será el número de colisiones

Dispersión Hashing

Resolución de Colisiones por el método "Progressive Overflow"

Algoritmo mas popular y bastante eficiente:

1)Cuando insertamos un registro obtenemos su dirección con la función Hash

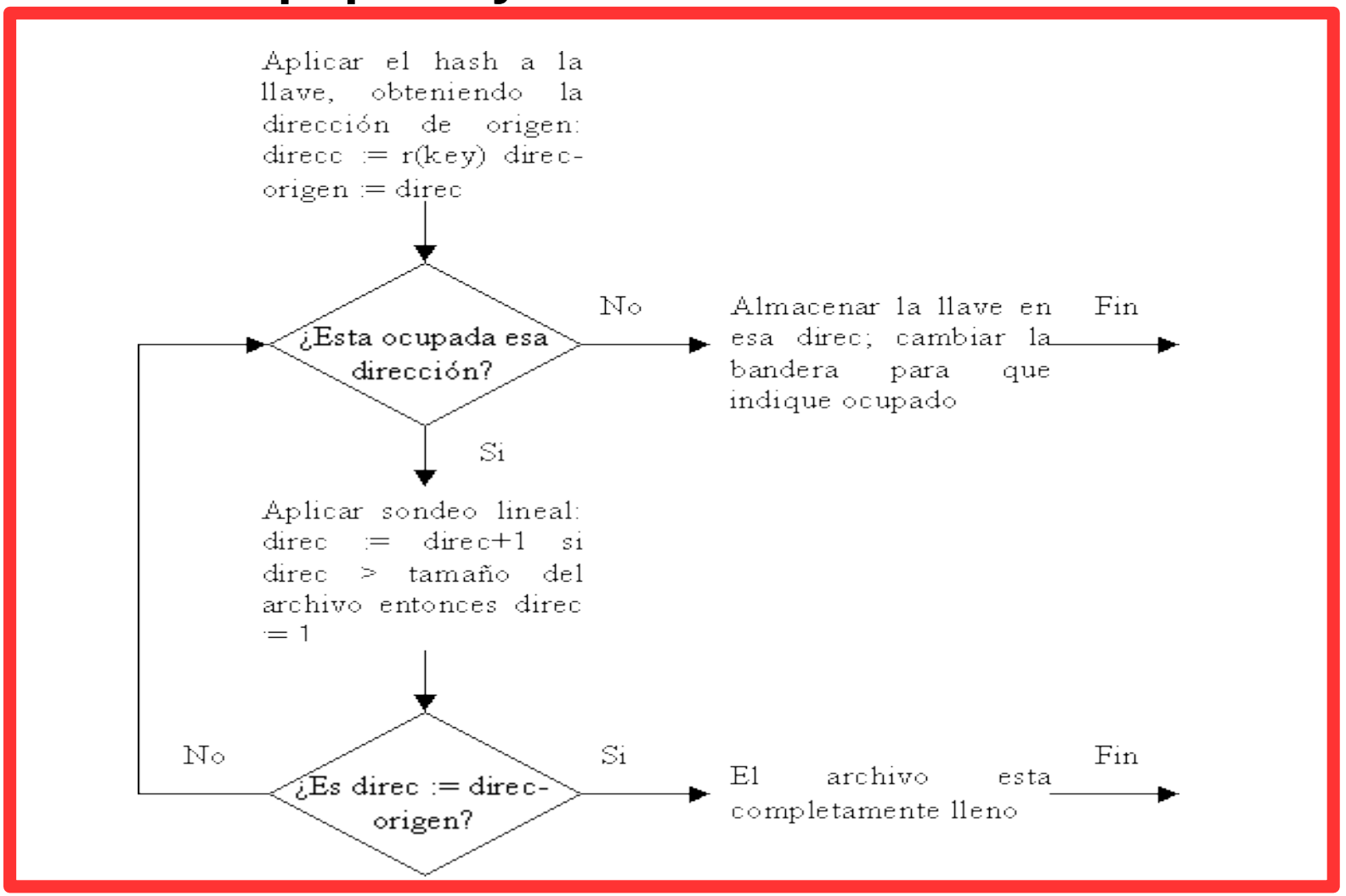
1)Si la dirección esta libre entonces lo insertamos ahí
2)Si NO se busca la primer dirección subsecuente que esté libre y se inserta.
1)Si llegamos al Final Empezar desde el comienzo.

- *Importante: El archivo debe en muchos casos tener un número fijo de direcciones posibles, Si no queda ninguno entonces quizás tuvimos un error en nuestros cálculos y planificación del archivo.*
- *Nota: Libre es Vacio o Marcado como Borrado .*

Dispersión Hashing

Resolución de Colisiones por el método "Progressive Overflow"

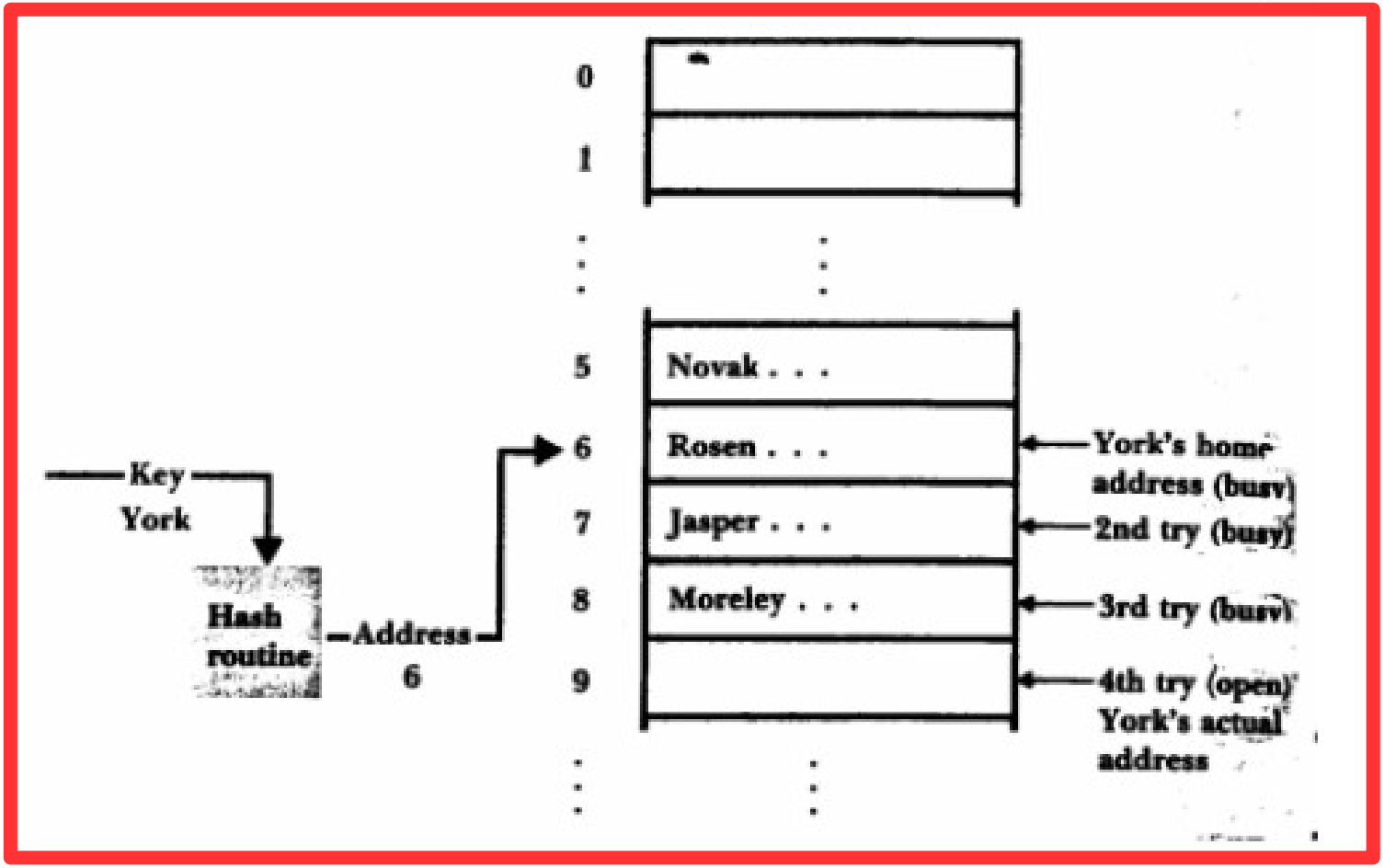
Algoritmo mas popular y bastante eficiente:



Dispersión Hashing

Resolución de Colisiones por el método "Progressive Overflow"

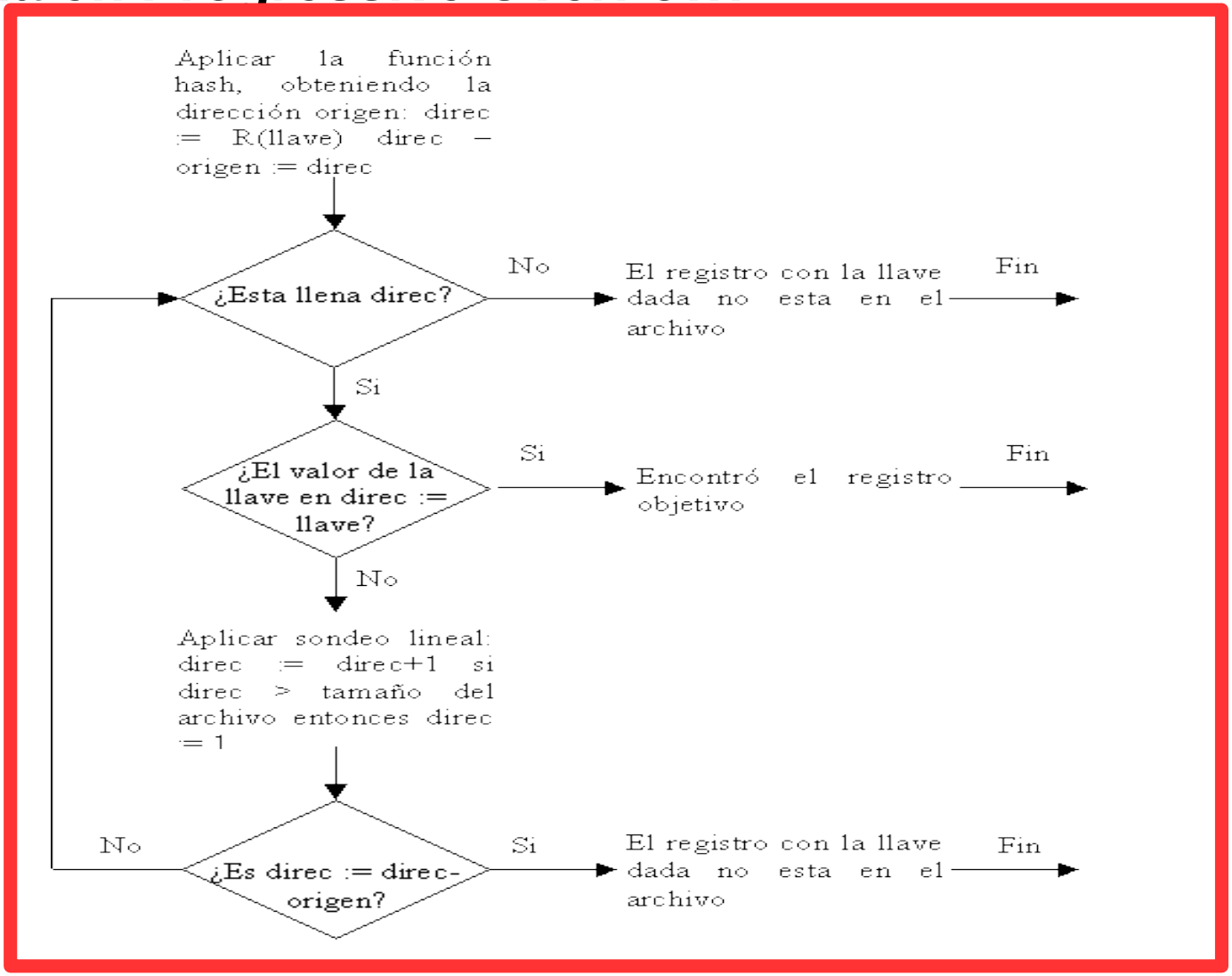
Algoritmo mas popular y bastante eficiente:



Dispersión Hashing

Resolución de Colisiones por el método "Progressive Overflow"

Búsqueda en Progressive Overflow:



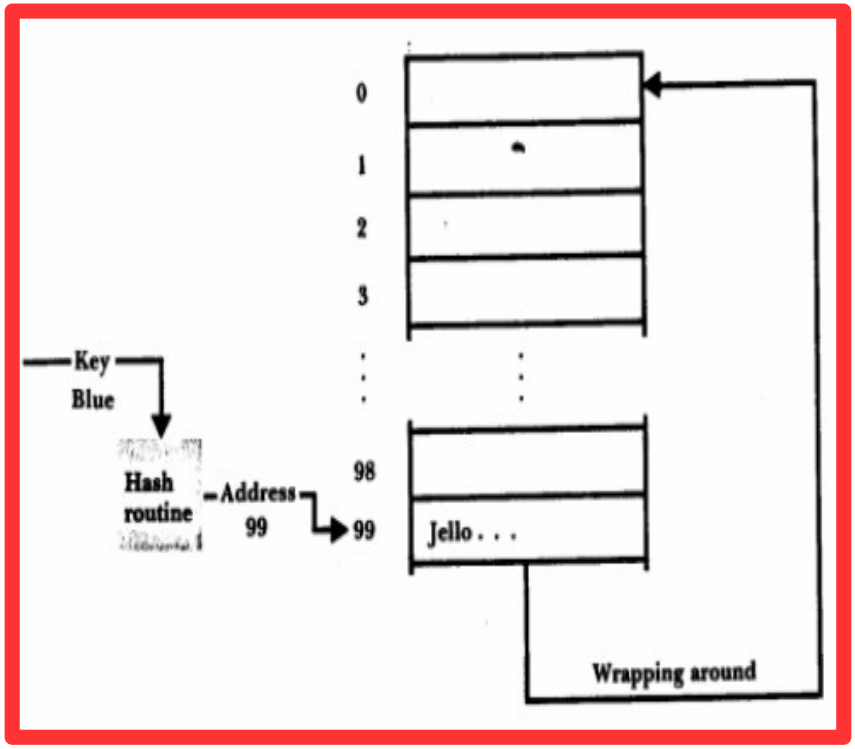
Dispersión Hashing

Resolución de Colisiones por el método "Progressive Overflow"

Búsqueda en Progressive Overflow:

1) Se obtiene el hash correspondiente.
2) Se busca a partir de esa dirección.
   Hasta que se encuentra o este vacío
Si llego al Final:
   Comienza desde el principio.
   Hasta la dirección calculada hash.

2) Si Es Vacío No existe el registro.



Dispersión Hashing

Resolución de Colisiones por el método Distribución en el almacenamiento por cubetas

Se asigna bloques de espacio (o “cubetas”), almacenar múltiples de registros, en lugar de asignar celdas individuales a registros.

Cuando una cubeta es desbordada, una nueva cubeta deberá ser asignada para el registro.

Los métodos para el problema de desborde de cubetas son iguales al de las colisiones por celda o ranura.

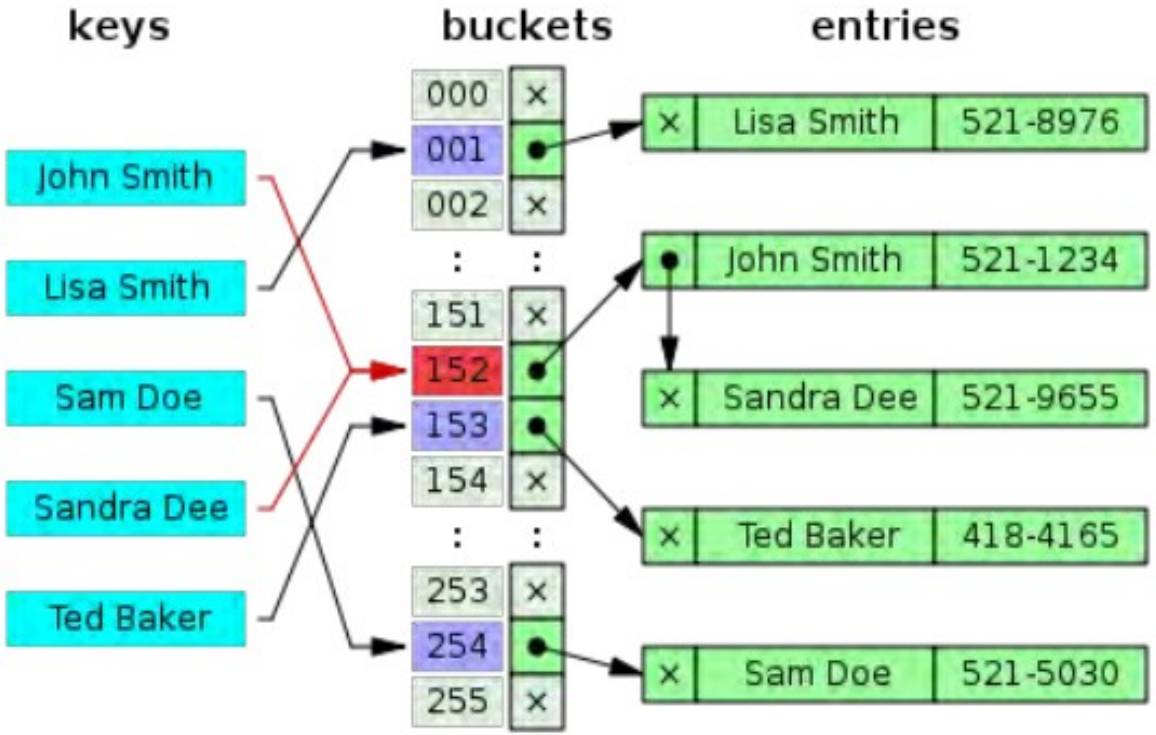
Si el direccionamiento abierto es utilizado, el espacio disponible se busca en la siguiente cubeta.

Cada cubeta principal generalmente tendrá un apuntador a una cadena de registros, en el área de sobre cupo.

Dispersión Hashing

Resolución de Colisiones por el método Distribución en el almacenamiento por cubetas

Cada Posición es una cubeta, la cual apunta a una lista de claves.
SI dos claves colisionan van a la misma cubeta.
Hay que minimizar la cantidad de claves x cubeta.



Dispersión Hashing

Otras soluciones para las Colisiones

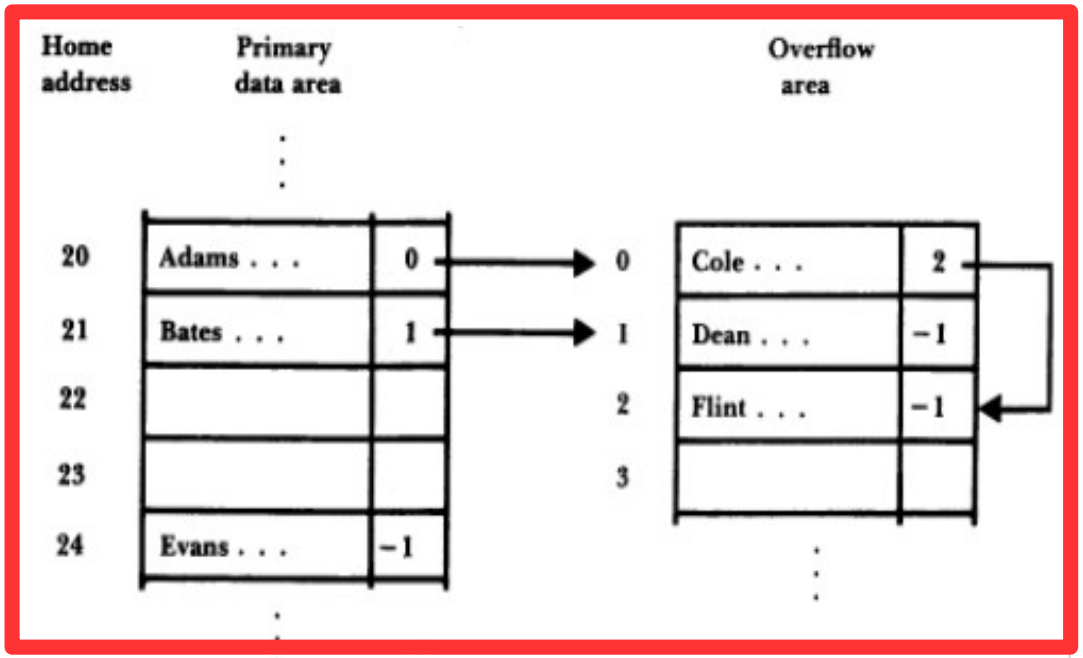
- **Doble Hashing**
  - Esta técnica se basa en la idea de generar otro hash, quizás con un formula diferente.
  - Por ejemplo:una clave que su hash da 5 5 y ya existe podriamos aplicar otro hash mas disperso que de 505. Para minimizar colisiones.
- **Encadenado de Progressive Overflow**
  - La idea, es muy simple y se basa en crear una lista encadenada entre los distintos registros "sinónimos".

Home address	Actual address	Data	Address of next synonym
20	20	Adams . . .	22
21	21	Bates . . .	23
20	22	Cole . . .	25
21	23	Dean . . .	−1
24	24	Evans . . .	−1
20	25	Flint . . .	−1

Dispersión Hashing

Otras soluciones para las Colisiones

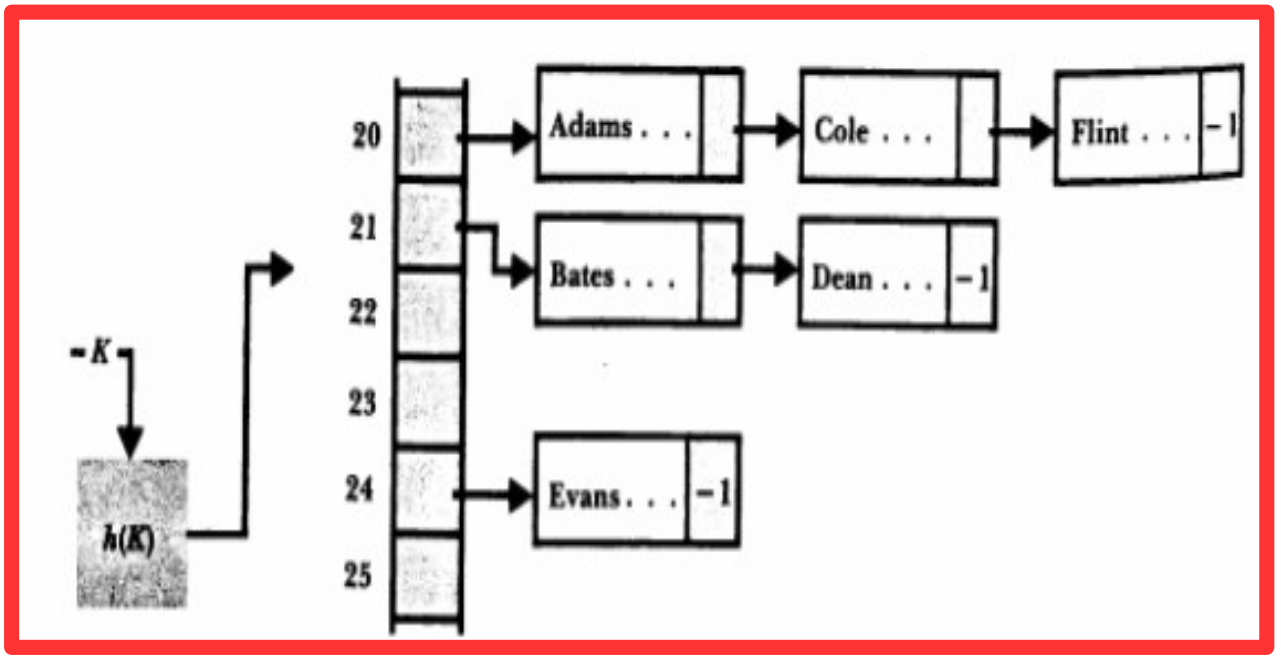
- **Encadenado con Separate Overflow Area**
  - **Aquí se emplea nuevamente el concepto de encadenamiento a través de una lista Encadenada**
  - **La diferencia es que se almacenan en otra area de overfow.**
  - **En el archivo de datos (Primary Data Area) sólo se almacena la cabeza de dicha lista.**



Dispersión Hashing

Otras soluciones para las Colisiones

- **Scatter Tables Tablas Dispersas**
  - **La idea no existe un área primaria de datos, sólo apuntadores hacia distintas cabeceras de listas donde residen aquellas llaves "sinónimas".**
  - **Hay distintas implementaciones desde muchos archivos, listas o archivo con listas de listas. de complicada implementacion.**



Dispersión Hashing

Hash Extendido

Permite que el archivo crezca sin cambiar la función de hashing.

Se necesitan 4 estructuras:

1. Archivo de tabla de dispersión, que guarda su tamaño de tabla (tt). La tabla posee punteros a bloques.
2. Archivo con bloques. Cada bloque soporta cantidad fija de registros de longitud fija . Cada bloque una tiene su tamaño de dispersión (td).

bloque.td = ((tabla.tt) / #ref s bloque en tabla)= distancia en tabla a otra ref

3. Archivo con lista de bloques libres.
4. Función de hashing

Métodos:

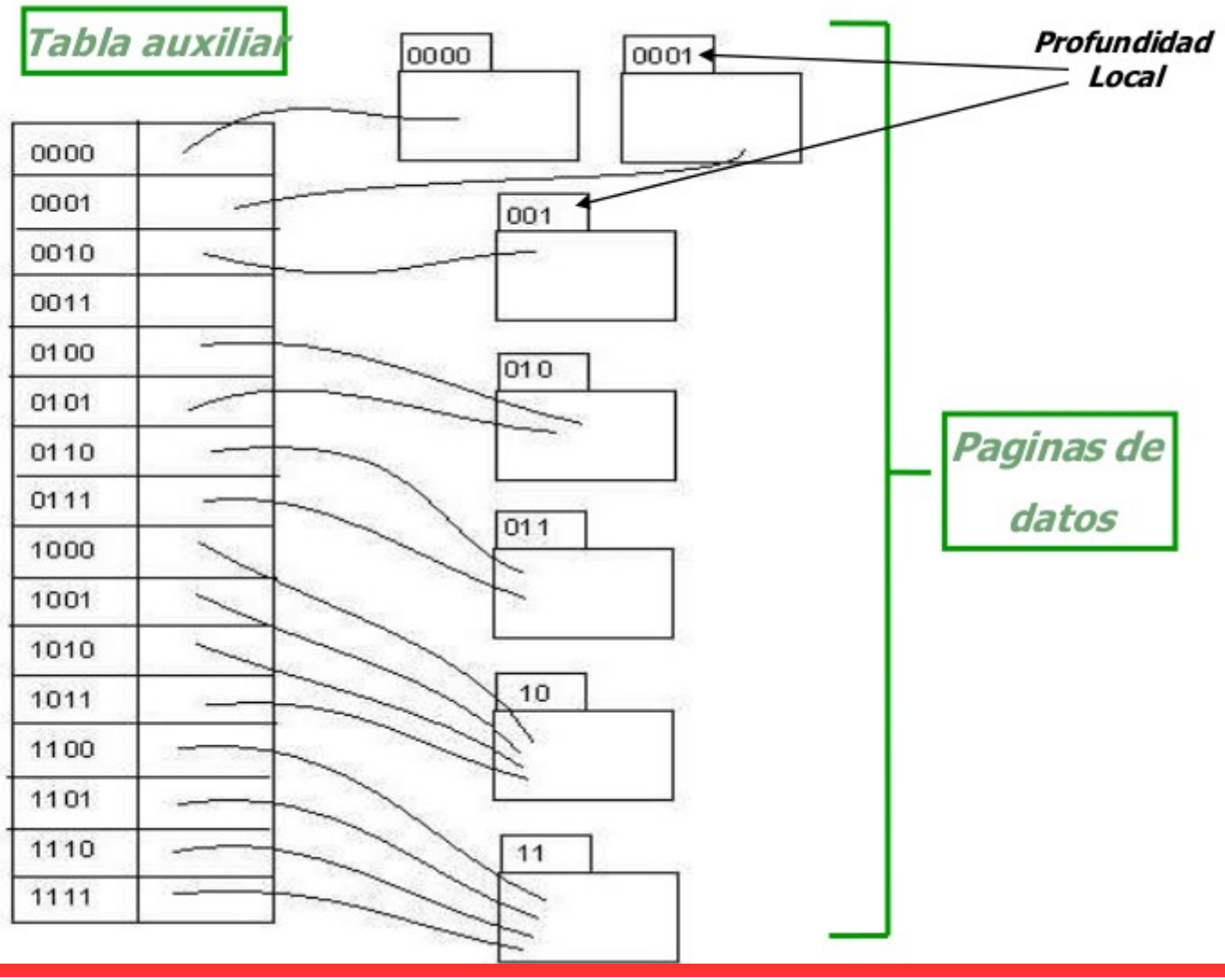
Mediante bit sufijos :

- Las claves se representan en base 10.
- f (clave) = clave mod tabla.tt

Dispersión Hashing

Hash Extendido

Ejemplo:



Dispersión Hashing

Hash Extendido

Datos de Partida

- *k= clave física para direccionar*
- *k'= h(k) valor entero entre 0 y M*
- *n= número de bits que tiene k' en binario*
- *d <=n, los d primeros dígitos de k' seleccionan la página donde está el registro y se llaman pseudollave.*
- *b<d<=n, inicialmente el archivo tiene 2^b cubos distintos , como máximo tendrá 2^d*

Algoritmo

- *Se considera una tabla índice en memoria con 2^d filas , en la primera columna de esta tabla, (valores de campo llave) se sitúan todas las posibles sucesiones de d dígitos binarios. d es la "profundidad" de la tabla*
- *En principio, todas las entradas cuyos b primeros dígitos son iguales apuntan al mismo cubo. Allí se almacenan los registros cuyo valor de k' tiene esos b primeros dígitos. Todos los cubos tienen en principio "profundidad local" igual a b.*
- *Cuando se llena un cubo se divide en 2, poniendo en uno de ellos los registros con el dígito b+1 de k' a 0 y en otro los que lo tienen igual a 1. La profundidad local de estos cubos aumenta una unidad.*

Dispersión Hashing

ANÁLISIS DE TABLAS HASH

- **Concluyendo:**
  - **El concepto de Hash es un caso ideal en la recuperación de información O(1)**
  - **El Hash tradicional es muy eficiente pero tiene 2 problemas principales: colisiones y el hecho de ser estático**
  - **El Hash extendido es una estructura relativamente moderna que promete ser mejor que el B-Tree, desafortunadamente existen algunas deficiencias que lo limitan, entre ellas:**
    - **No se tienen algunas métricas importantes acerca de la capacidad de cada cubeta**
    - **Al igual que el BTree la estructura crece dinámicamente y permite búsquedas de alta velocidad; pero para el acceso secuencial (B+Tree) sigue ganando la competencia.**

Dispersión Hashing

Funciones de hashing unidireccionale

Una función de hashing unidireccional es de la forma H(M) = k .

Tienen ciertos requisitos:

- Toma un valor de longitud variable M y devuelve un valor de longitud fija k (generalmente la longitudes fijas de 128 bits o superior).
- Dado M debe ser sencillo calcular k.
- Dado k debe ser muy difícil encontrar un M tal que H(M) = k
- Dado M debe ser muy difícil encontrar un M' tal que H(M) = H(M')

Usos

- Construcción de estructuras de datos.
- Integridad de documentos (cualquier tipo cualquier tamaño)
- Firmas digitales.
- Sellos de Tiempo confiables.
- Códigos de verificación
- Combinación, con salto para claves de usuarios

Ejemplos

- MD5 : devuelve un valor de 128 bits.
- SHA : devuelve un valor de 160 bits.

Dispersión Hashing

Ejemplo de MD5

1) Si En consola con "|" le paso la salida del comando "echo" al "md5sum"

```
echo hola mndo|md5sum
```

y retorna el hashing de "hola mndo"

```
4bac26ae4cb200b5f602427658b8adae  -
```

2) En consola ejecuto el md5sum al archivo "RedesDeComputadoras.pdf"

```
md5sum RedesDeComputadoras.pdf
```

y retorna el hashing del contenido del archivo:

```
fa76f3c302763a9216bc70479b014c4b  RedesDeComputadoras.pdf
```

Muy útil para firmar archivos y detectar si alguno fué modificado.