2. ARCHIVOS

Entendemos por **estructura** a la forma cómo están constituidos físicamente los archivos, y **organización** de archivos a la forma de administración de los archivos, en función de las relaciones de los registros.

Criterios para Elegir un tipo de Organización

Se pueden considerar tres criterios básicos:

- Rápido Acceso
- Economía de Almacenamiento
- Facilidad de Uso

La elección de la organización determina el rendimiento relativo del sistema para cada una de las tres características enunciadas.

Algunas mediadas de rendimiento son:

a) Almacenamiento requerido por un registro.
b) Tiempo de búsqueda de un registro.
c) Tiempo requerido para leer todo el archivo.
d) Tiempo requerido para insertar un registro.
e) Tiempo para modificar un registro.

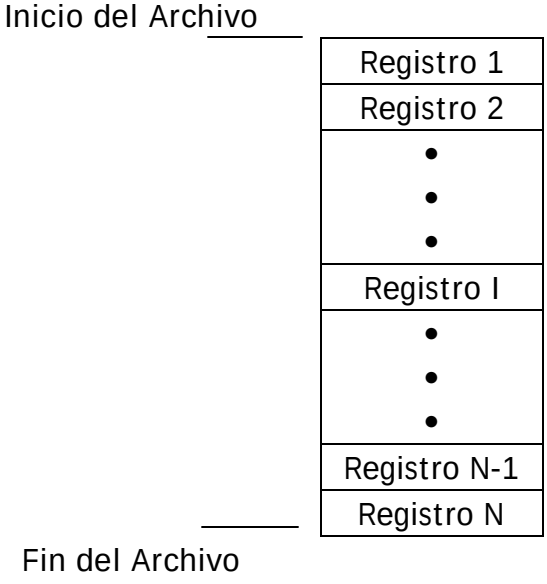
Para archivos, existen las siguientes organizaciones:

- Secuencial
- Secuencial Indexado
- Directa (relativa)

2.1 Organización de Archivos Secuenciales

Los registros están grabados consecutivamente cuando el archivo se crea, y de igual forma deben ser accesados consecutivamente cuando el archivo es procesado o tomado como entrada de datos.

La estructura básica puede verse en la figura siguiente:



Los registros de un archivo secuencial quedan ordenados de acuerdo con el valor de algún campo o grupo de campos, denominados **clave** o **llave**.

Esta organización resulta adecuada cuando se tiene posicionamiento secuencial:

- Accesar el próximo registro es trivial.
- Con respecto a la actualización, el reemplazo de campos de igual largo puede hacerse reescribiendo dicho campo.
- Para agregar registros a un archivo secuencial hay dos opciones:
  - ⇒ Crear un nuevo archivo.
  - ⇒ Agregar al final del archivo.
- Para eliminar los registros estos se pueden marcar (necesidad de un campo extra) o se debe crear un nuevo archivo.
- Los archivos secuenciales ocupan un tamaño mínimo, o sea, sólo el espacio requerido para el almacenamiento de los registros.
- Mientras que el patrón de acceso al archivo sea el mismo que el dado por el ordenamiento de los registros, el tiempo de acceso será mínimo.

2.2 Sorting, Merging, Reporting

2.2.1 Sorting

La idea del ordenamiento es reubicar los registros de un archivo, de acuerdo a alguna estrategia predefinida, de manera que queden en una secuencia tal que pueda ser utilizado por otro proceso. Como resultado de este ordenamiento, se genera un segundo archivo, que contiene exactamente los mismos registros, pero en la secuencia deseada.

Según algunos autores, el ordenamiento se produce cuando el campo clave tiene valores en todos los registros, de lo contrario se habla de **clasificación**.

Por otra parte, si hablamos de archivos secuenciales, cuyos registros han sido almacenados por dos campos claves, no podrá estar ordenado también por otro campo, sino que se deberá generar otro archivo que cumpla con esta nueva condición de ordenamiento.

Una representación es la siguiente:

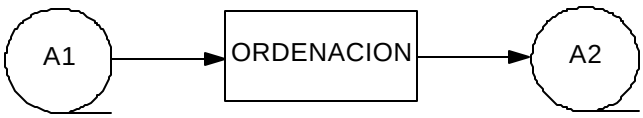
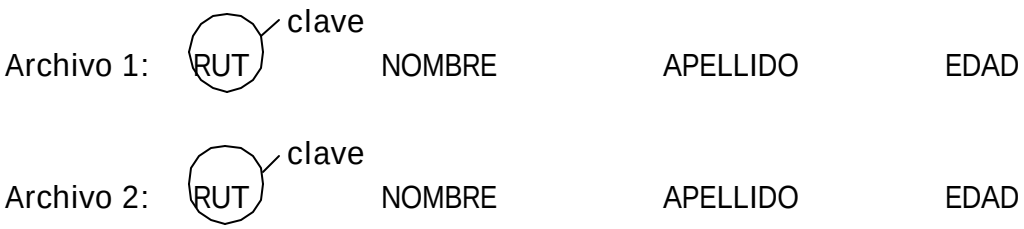


Figura 3

2.2.2 Merging

La intercalación consiste en agregar registros a un archivo, a partir de dos o más archivos, los cuales deben estar ordenados por un campo clave. El archivo resultante también estará ordenado por el mismo campo clave.

Se hace hincapié en que ambos (todos) archivos deben poseer los mismos campos, y ordenados en la misma secuencia. Por ejemplo:



Archivo 3: APELLIDO NOMBRE RUT clave EDAD

Se puede hacer un merge entre 1 y 2 pero no entre 1 y 3 o entre 2 y 3.
Una representación es la siguiente:

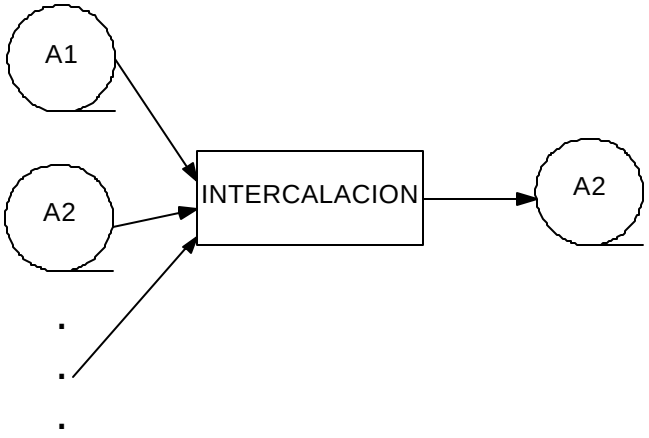


Figura 4

2.2.3 Reporting

Una vez que se han generado los archivos, éstos se deben ocupar para algún trabajo específico. Esta es la idea de los **reportes**. Los reportes son datos de salida ubicados según alguna estrategia, que permite un cabal entendimiento de los datos.

Un reporte puede verse como un conjunto de campos, no necesariamente de un mismo archivo (por lo menos esta es la idea más actual de un reporte).

Los registros de los archivos de reporte contienen las siguientes partes:

a) Registros de Encabezados: Incluye título del reporte, encabezado de página y encabezado de grupo para identificar la información.
b) Registros de Detalle: Incluye toda la información que se desea mostrar o reportar, y es la parte variable. Generalmente se encuentran arregladas en columnas.
c) Registros de Pié de Página: Incluye pies de página de grupos, páginas y reportes (información sumaria).

Los datos sumarios se utilizan para calcular cortes de control, los cuales corresponden a cambios de algún campo clave.

Una representación es la siguiente:

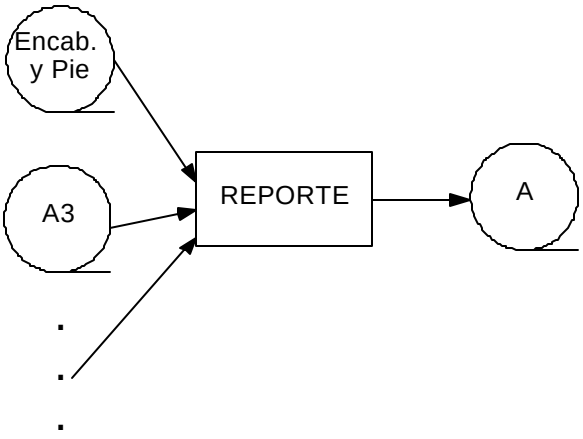


Figura 5

2.3 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.

La forma de acceder a los registros es a través de la clave de dicho archivo.

	Clave		Posición Física
Inicio del Archivo			
	Zalo		1
	Ana		2
	Patricio		3
	•		
	•		
	•		
	Norma		I
	Bruno		I+1
	•		
	•		
	•		

	Carla		N-1
	Esteban		N

Fin del Archivo

Para sacar provecho de este tipo de organización, es conveniente que el medio de almacenamiento permita un acceso directo. Esto nos lleva a concluir que la memoria principal y los discos magnéticos son la mejor opción, en cambio las cintas son la peor.

El inconveniente que tiene este tipo de acceso es que se debe programar la relación existente entre el contenido de un registro y su posición física.

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

Para poder utilizar direccionamiento directo se debe tener un conjunto de datos con las siguientes características:

1. El conjunto de claves debe tener un orden ascendente, con pocos valores no utilizados, que podrían significar espacio de almacenamiento desperdiciado.
2. La clave de los registros corresponden con los números de las direcciones. Existe una dirección de almacenamiento en el archivo por cada valor posible de la clave, y éstas no tiene valores duplicados.

Los valores de las claves están en un rango acotado, ya que cuando se asigna el espacio de almacenamiento, se comienza con el valor más bajo de la clave y se termina con el valor más alto.

A continuación se dará un ejemplo, en el que se podrán ver las características en 2 archivos, y como ellas están mejor dadas en un archivo que en otro.

Ejemplo:

001	001
002	004
004	007
005	010

006	036
008	047
010	048
011	050
012	056
014	

La secuencia de claves de la izquierda permite un buen direccionamiento directo ya que hay pocas claves que faltan, caso contrario ocurre con la de la derecha. Ahora bien, con ambos conjuntos se puede tener direccionamiento directo.

Otro problema que prohibe el uso de direccionamiento directo es cuando las claves para un registro no representan una dirección. Ahora,, si se necesita acceso directo y no es posible el direccionamiento directo, se debe utilizar hashing.

Ejemplo:

Sea un conjunto de claves, en el que el mínimo es 1 y el máximo es el 18. Sean los registros con las siguientes claves:

2, 4, 7 ,8, 10, 14, 15

Al ser puestos en registros tendrían la siguiente disposición:

2	
4	
7	
8	
10	

14	
15	

también se puede presentar la ocasión de utilizar una fórmula para el cálculo de la dirección física.

Dirección Base
Tamaño del Registro

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

2.4 Archivos Indexados

En los archivos indexados pueden verse como un conjunto de registros, los que pueden accesarse mediante una clave.

Este tipo de archivos constan de 2 partes:

- Area Principal
- Area de Indices

2.4.1 Area Principal

En esta área se almacenan los registros, con los datos, al momento de crear el archivo.

El archivo es creado secuencialmente, es decir, se escriben los registros en el archivo primario en una secuencia indicado por el ordenamiento previo de las claves. Esta área incluye a todos los campos de cada registro.

2.4.2 Area de Indices

Esta área es creada automáticamente por el sistema.

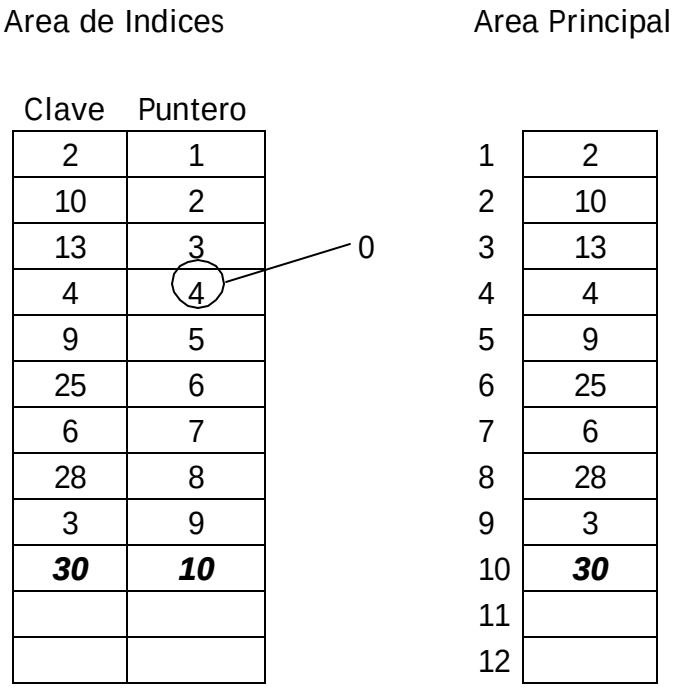
Esta área contiene tantos registros como registros existan en el área principal. Cada registro del área de índices consta de 2 campos:

- Clave de los Registros
- Puntero al Registro en el área principal

Ejemplo:

Sea el siguiente archivo (sólo se muestran las claves):

2, 10, 13, 4, 9, 25, 6, 28, 3



Clave	Puntero
2	1
10	2
13	3
4	4
9	5
25	6
6	7
28	8
3	9
30	***10***

1	2
2	10
3	13
4	4
5	9
6	25
7	6
8	28
9	3
10	***30***
11	
12	

Insertar 30: Siempre se inserta al final, es decir, el archivo crece siempre hacia abajo.

Eliminar 4: Se **marca** el registro con cero, y así se elimina lógicamente

Insertar 8: Se pone al final del archivo o se busca un hueco.

2.5 Acceso Secuencial Indexado

Para aclarar la idea del acceso secuencial indexado veremos el caso del diccionario.

Un diccionario es un ejemplo de archivo secuencial, cuyos registros son las definiciones dadas allí. Para buscar una palabra no se recorre todo el diccionario,

sino que primero se abre éste en la letra correspondiente, para luego buscar en el extremo superior, hasta encontrar la palabra más próxima a la que se busca, para así recorrer dicha hoja palabra por palabra, hasta encontrar la que buscamos.

De esta forma el diccionario es un ejemplo típico de archivo secuencial indexado, con dos niveles de índices:

- El nivel superior para las letras iniciales.
- El nivel menor para las palabras ubicadas en el extremo superior de cada página.

Un archivo en organización secuencial indexada consta de tres partes:

- Area Primaria o de Datos.
- Area de Indices.
- Area de Desbordamiento u Overflow.

2.5.1 Area Primaria de Datos

Esta contiene los registros que componen el archivo, el cual es creado secuencialmente.

El proceso de escritura comienza en la segunda pista de un cierto cilindro, hasta completar las pistas de este cilindro. Una vez completo se continua en el siguiente cilindro, en la segunda pista, hasta completar el archivo.

Si el archivo es accesado secuencialmente, según el orden de la clave, los registros serán accesados en el orden que se han escrito.

2.5.2 Area de Indices

Corresponde al lugar donde se almacenan algunos índices, no todos.

La primera pista de cada cilindro contiene un índice a las claves de los registros de ese cilindro.

Un esquema sencillo es suponer un único nivel de índices.

El índice de pista contendrá dos elementos:

- Normal
- Overflow

La entrada normal está compuesta por la dirección de la pista primaria y por el mayor (o menor) valor de la clave de los registros almacenados en dicha pista. Otra alternativa es que la clave vaya primero y luego la dirección de la pista.

Si no hay datos en el área de overflow, el contenido de la entrada es el mismo que la de la entrada normal.

Ejemplo:

Sea el siguiente archivo, del cual sólo se muestran las claves ordenadas:

Supongamos que en cada pista se puede almacenar cuatro registros

P0	P1:20|Λ:20	P2:33|Λ:33	P3:62|Λ:62	P4:67|Λ:67
P1	1	9	17	20
P2	23	30	32	33
P3	48	50	61	62
P4	65	66	67	

2.5.3 Area de Overflow

El área de Overflow está destinada a contener las inserciones de los registros que no pueden ser realizadas en el área primaria.

En el área de overflow los registros se presentan como una lista encadenada, en que el puntero al próximo registro está compuesto por la pista y el lugar que ocupa el registro dentro de la pista (Se acostumbra a usar el mismo cilindro para el área primaria y la de overflow).

Esta lista se mantiene ordenada por clave. Cuando una pista pasa hacia área de overflow, la entrada en el área de overflow contiene el puntero al comienzo de la lista (número de pista, registro) y el máximo valor en la lista.

Ejemplo:
Cilindro 1

P0	P1:9 | Λ:9		P2:36 | Λ:36		P3:70 | Λ:70	Area de Indices
P1	1	3	8	9	
P2	12	19	24	36	Area Principal
P3	61	70			
P4					
P5					Area de Overflow

Insertar 62, 6

P0	P1:8 | P4,1:9	P2:36 | Λ:36	P3:70 | Λ:70	
P1	1	3	6	8
P2	12	19	24	36
P3	61	62	70	
P4	9 | Λ			
P5				

Insertar 7

P0	P1:7 | P4,2:9	P2:36 | Λ:36	P3:70 | Λ:70	
P1	1	3	6	7
P2	12	19	24	36
P3	61	62	70	
P4	9 | Λ	8 | P4,1		
P5				

Insertar 2

P0	P1:6 | P4,3:9	P2:36 | Λ:36	P3:70 | Λ:70	
P1	1	2	3	6
P2	12	19	24	36

P3	61	62	70	
P4	9 | Λ	8 | P4,1	7 | P4,2	
P5				

Borrado de Registros

El borrado puede manejarse de 2 maneras:

- Borrado Físico: Si el registro estaba en el área de overflow, se debe ajustar la lista ligada para compensar el borrado. Se deben contemplar posibles modificaciones al área de índices. Si el registro estaba en el área principal, las entradas son corridas hacia la izquierda; un registro de la cadena de overflow es llevado al área primaria y se modifica el área de índices.
- Borrado Lógico: Se marca el registro, pero no se elimina físicamente. Las recuperaciones de información saltarán estos registros. Después mediante una **recolección de basura** se puede recuperar dicho espacio.

Ejemplo:

Borrar 7, Insertar 25

P0	P1:6 | P4,2:9	P2:25 | P4,3:36	P3:70 | Λ:70	
P1	1	2	3	6
P2	12	19	24	25
P3	61	62	70	
P4	9 | Λ	8 | P4,1	36 | Λ	
P5				

Insertar 7, Borrar 8

P0	P1:6 | P5,1:9	P2:25 | P4,3:36	P3:70 | Λ:70	
P1	1	2	3	6
P2	12	19	24	25
P3	61	62	70	
P4	9 | Λ		36 | Λ	

P5	7 | P4,1		

Borrar 6

P0	P1:3 | P5,1:9	P2:25 | P4,3:36	P3:70 | Λ:70	
P1	1	2	3	~~6~~
P2	12	19	24	25
P3	61	62	70	
P4	9 | Λ		36 | Λ	
P5	7 | P4,1			

El 6 se marca, por lo que queda lógicamente eliminado.

El manejo del área de overflow se puede realizar mediante tres estrategias distintas:

- Area de Overflow en el Cilindro
- Area de Overflow Independiente
- Area de Overflow Mixta

a) En el Cilindro:

Es una cantidad de pistas por cilindro que se reserva como área de overflow para el área primaria de ese cilindro, es decir, cada cilindro tiene su propia área de overflow, y cuando ésta se llena, ya no se pueden hacer más inserciones, aunque las áreas de overflow de otros cilindros tengan espacios disponibles.

Una ventaja de esta forma de manejar el overflow es que no se requiere de movimientos del brazo del disco (no hay tiempo de seek incorporado) para accesar los registros de overflow.

Una desventaja es que queda mucho espacio sin usar si las inserciones no se distribuyen de manera uniforme.

b) Independiente:

Reside en un cilindro aparte de cualquier cilindro del área primaria.

Los registros de overflow son encadenados dentro de esta única área de overflow, no importando de cual cilindro o área primaria vengan.

La ventaja de esta forma es que se hace una mejor utilización del espacio, ya que el área de overflow es compartida. La desventaja es que hay un mayor tiempo de búsqueda , ya que se necesita mover las cabezas lecto-grabadoras para accesar los registros de overflow correspondiente a un área primaria determinada.

c) Mixta:

Este tipo de manejo de área de overflow es una combinación de los dos anteriores, en el sentido que cada cilindro tiene su propia área de overflow, y además se establece un área de overflow independiente, común a todos los cilindros, la cual es utilizada una vez que el área de overflow del cilindro se encuentra llena.

2.6 Arboles B^+

La principal característica de los árboles B^+ es que todas las claves se encuentran en las hojas, por lo que la distancia desde la raíz a cualquier clave es siempre la misma. Por otro lado, los árboles B^+ ocupan un poco más de espacio que los árboles B, debido a la duplicidad de alguna de sus claves.

Un árbol B^+ se define de la siguiente manera:

1. Cada página, excepto la raíz, contiene entre **d** y **2d** elementos.
2. Cada página, excepto la raíz, tiene entre **d+1** y **2d+1** descendientes. Se utiliza **m** para expresar el número de elementos por página.
3. La raíz tiene al menos dos descendientes.
4. Las hojas están todas al mismo nivel.
5. Todas las claves se encuentran en las hojas.
6. Las claves de las páginas raíz e interiores se usan como índices.

2.6.1 Búsqueda en Arboles B^+

El proceso de búsqueda se realiza de manera similar a los árboles B, diferenciándose con éstos en que la búsqueda no termina al encontrar la clave, sino que se debe llegar a las hojas.

2.6.2 Inserción en Arboles B^+

La inserción es similar a los árboles B. La diferencia se presenta cuando se inserta una clave en una página llena (m=2d). En este caso, la página se divide en dos:

las d primeras claves se ubican en la página de izquierda. Las d+1 restantes en la página de la derecha. Una copia de la clave del medio sube a la página antecesora.

Se debe notar que el desbordamiento en una página que no es hoja, no produce duplicidad de claves.

Ejemplo:

Insertar las siguientes claves en un árbol B^+ de grado 2.

3, 33, 50, 11, 80, 1, 44, 34, 9, 40, 70, 75, 20, 30, 27, 60, 15

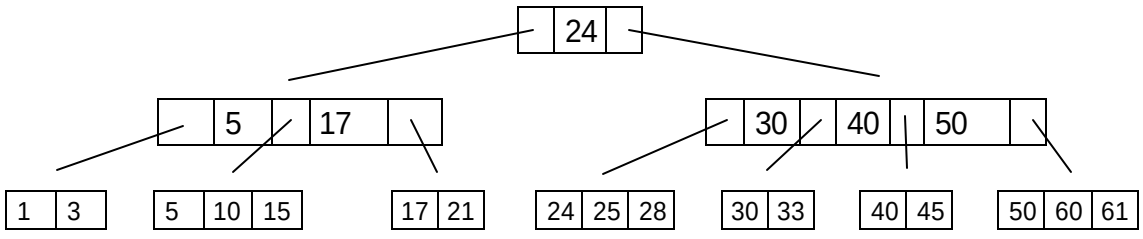
2.6.3 Eliminación en Arboles B^+

Se deben distinguir básicamente dos casos:

1. Si al eliminar una clave, m queda mayor o igual a d, entonces se termina la eliminación. Las claves en la raíz o páginas internas no se alteran, aunque sean copia de una clave eliminada.
2. Si al eliminar una clave, m queda menor que d, entonces se deben redistribuir las claves, ya sea en el índice como en las páginas hojas.

Ejemplo:

Del siguiente árbol B+, elimine las siguientes claves:



2.10 Acceso Aleatorio con Técnicas de Hashing

El acceso mediante hashing consiste en realizar un cálculo sobre el valor de la clave del archivo, lo que da como resultado un dirección relativa.

La idea del hashing es aplicar la función sobre un número de claves, que se traduzcan en un número pequeño de direcciones.

Por ejemplo, si se sabe que un archivo estará compuesto por a lo menos 5000 registro, y la clave es el RUT, entonces se debe elegir una función que a lo menos proporcione 5000 valores posibles de direcciones.

Un problema típico es que la elección de la función no es siempre tan buena como se quisiera, ya que no existe una relación uno a uno entre el valor de la clave y la dirección física (dirección relativa). Cuando esto sucede, entonces nos encontramos con una colisión, es decir:

$$F(k1) = F(k2)$$

$$\text{con } k1 \neq k2$$

De todas formas esta técnica tiene ciertas ventajas:

1. Se pueden usar valores cualesquiera para la clave, ya que se traducen internamente a direcciones.
2. Se logra independencia física y lógica, ya que los valores de las claves son independientes del espacio de direcciones. Por ejemplo, si el archivo es reorganizado (u ordenado por otra clave), la función de hash deberá cambiar, pero los valores de la nueva clave no.

También posee desventajas:

1. El tiempo de procesamiento requerido para la función hash.
2. El tiempo de procesamiento y los accesos de entrada/salida requeridos para las colisiones.

Se debe recordar que para hacer un mejor uso del hashing, se puede utilizar una tabla en memoria, la que tenga calculada de antemano las direcciones, es decir que se genere en el momento de ingresar un registro. Ahora, puede suceder que el tiempo de búsqueda aumente, si elegimos este esquema.

La eficiencia de una función hashing depende de:

1. La distribución de los valores de claves que realmente se usan.
2. El número de valores de clave que realmente están en uso, con respecto al tamaño del espacio de direcciones.
3. El número de registros que pueden almacenarse en una dirección sin causar colisión.
4. La técnica usada para resolver las colisiones.

2.10.1Funciones Hashing

La función hash es la que determina en qué dirección se debe ubicar una clave dada.

La elección e esta función para que le tiempo de ejecución sea el menor posible.

Por ejemplo, si la clave de entrada es un entero, entonces elegir como función hashing,

f (K) = K MOD max_long

Es una estrategia adecuada. Ahora si el tamaño de la tabla es 10, y todas las claves terminan en cero, entonces la elección de esta función es muy mala.

Según varios autores, es una buena idea que le tamaño del arreglo o tabla sea un número primo.

Ahora, las características deseables de una función hashing son al menos las dos siguientes.

- f (K) debe tender a minimizar el número de colisiones.
- f (K) debe considerar el conocimiento respecto de las propiedades de las claves (en relación con su frecuencia de uso).

2.10.1.1 Función Módulo (Por División)

Consiste en tomar el residuo de la clave por el número de componentes del arreglo.

K: Clave del dato

N: Cantidad de elementos del arreglo.

f (K) = K MOD N + 1
1 < f(K) < N

2, 1, 34, 24, 15, 23, 0, 47 La elección de N es critica, y en la practica se ha observado que es suficiente elegir
N de tal forma que no tenga divisores primos menores que 20.

Ejemplo:

Si tenemos un arreglo declarado de la siguiente forma:

```
CONST N= 23;
VAR arreglo: ARRAY        OF INTEGER;
```

Y aplicamos la función hashing anterior a las siguientes claves

y obtenemos:

f (2) = 3
f (1) = 2
f (34) = 12
f (24) = 2
f (15) = 16
f (23) = 1
f (0) = 1
f (47) = 2

2.10.1.2 Mid-Square hashing

La clave K se eleva al cuadrado y se obtiene un índice truncado dígitos en ambos
extremos del valor de K, hasta obtener un valor que esté dentro del rango 1..N.

Para mejorar el rendimiento, el tamaño de la tabla se recomienda una potencia de
Dos.

¿Para qué valores de clave sirve esta función? Para valores tales que K>1.

Ejemplo:

N = 10
K = 122

Existen algunos problemas con ciertas claves, como por ejemplo K = 10; para solucionar este problema se puede cambiar el rango de direccionamiento a 0..(N-1).

2.10.1.3 Folding

La clave K se particiona en varias partes, todas ellas del mismo tamaño, salvo quizás
la última. Dichas partes se suman para obtener la dirección de entrada a la tabla.

Ejemplo:

K = 123 456 196 43 E = 123+456+196+43= 818

2.10.1.4 Folding Modificado

r r
K = 123 456 196 43 E = 321+456+691+43= 1511

Para todas las funciones hashing es indispensable realizar un análisis del problema,
para poder identificar cuales son los valores posibles de clave, espacio de direccionamiento y función hashing que más se acomoda al problema a solucionar.

2.10.2Manejo de Colisiones

Si al insertar un elemento, la función hash devuelve el mismo valor que ya ha sido

Asignado a otro elemento, entonces se ha producido una colisión.

Se consideran dos formas o métodos para solucionar el problema de colisión:

- Open Hashing (Separate Chaining)
- Close Hashing (Open Addressing)

La primera permite que los elementos se almacenen en un espacio potencialmente

ilimitado. La segunda usa un espacio fijo para el almacenamiento. Para efectos de este curso sólo veremos Close Hashing.

2.10.2.1 Close hashing

En close hashing, si ocurre una colisión, se recorren las celdas, según alguna estrategia, hasta que se encuentre alguna vacía.

La representación es la siguiente:

Formalmente las celdas h0 (K), h1 (K), h2 (K), ... son probadas sucesivamente donde:

$$Hi(K) = (hash(K) + f\ (i)\ MOD\ N$$

Con f (0) = 0. La función **f** es la estrategia de resolución de colisiones.

De acuerdo a esta última función, se definen tres estrategias para enfrentar colisiones:

- Prueba Lineal
- Prueba Cuadrática
- Doble Dirección Hash.

2.10.2.1.1 Prueba Lineal

La función **f** es típicamente f (i) = i. Esto provoca que la lista sea recorrida secuencialmente hacia delante (y con vuelta atrás), en busca de una celda vacía.

Ejemplo:

Hash (K) = MOD 10

Inserción de las claves {0, 1, 23, 13, 43}

0
1
2
3
4
5
6
7
8

9

Si la tabla está relativamente vacía, entonces se empezarán a formar bloques de claves. Este efecto es conocido como **Primary Clustering**, o en español **Agrupamiento Primario.**

2.10.2.1.2 Prueba Cuadrática

La función f es típicamente f (i) = i. Esta prueba elimina el problema del agrupamiento primario.

Ejemplo:

Hash(K) = K MOD 10

Inserción de las claves {0, 1, 23, 13, 43}

0
1
2
3
4
5
6
7
8
9

2.10.2.1.3 Doble Dirección Hash

Cuando se detecta la colisión se debe generar otra dirección, aplicando la función
hashing a la dirección previamente obtenida.

Conceptualmente la secuencia de pasos puede ser la siguiente:

Direc = hash 0 (K)
Direc' = hash 1 (Direc)
Direc'' = hash 2 (Direc')

-
-
-

La función hashing puede o no ser la misma en cada colisión detectada. Lo importante es que para las primeras colisiones se use siempre la misma función, para las segundas las mismas, y así sucesivamente. En otras palabras:

hash 0 (K1) = hash 0 (K2) = = hash 0 (Kn)
hash 1 (Direc) = hash 1 (Direc) = = hash 1 (Direc)

-

(lo anterior representa que se usa la misma función hash, no que se obtiene la misma dirección).

Empleados de Ventas

1	4	5	6	8

Con esta organización, la formación de esquemas compuesto de acceso puede ser hecha directamente sobre los directorios. Así, si se necesita conocer a los empleados de ventas que se encuentran casados, sólo basta con realizar la intersección de las dos listas correspondientes.

Desde un punto de vista más purista, el uso del termino inversión implica que los valores de los datos indexados han sido sacados del registro de datos, y residen solamente en el correspondiente índice de inversión. Así, el archivo **completamente invertido** tiene un índice de inversión para cada campo de datos. O sea, el registro podría excluir el campo.

En el ejemplo anterior, el campo ESTADO CIVIL, HIJOS y DEPARTAMENTO pueden no estar presentes. Para realizar las actualizaciones, se debe contemplar la existencia de estos otros archivos o directorios.

Por otro lado, un archivo completamente invertido podría no tener campos de datos.

Si un archivo no es completamente invertido, pero tiene al menos un índice de inversión, entonces se dice que es **parcialmente invertido**.

Un índice de inversión puede construirse sobre un archivo secuencial indexado, con lo que se podrá tener un archivo que puede ser accesado por mas de una clave. Las otras podrían llamarse **claves secundarias**.

Para más ejemplo, ve el capitulo de búsqueda externa (capitulo 8), listas invertidas, en el libro "Estructura de Datos" de Cairó/Guardati.

2.11 Organización de Lista Invertida

Este método permite proporcionar el encadenamiento entre un índice y los registros del archivo.

Un índice de inversión contiene todos los valores que la clave tiene presente en los registros del archivo. Cada uno de los valores de la clave en el índice de inversión apunta a todos los registros que tienen el valor correspondiente. El archivo de datos se dice que está **invertido** sobre esa clave.

Según lo anterior, podemos pensar en lo siguiente: la organización, al poseer dos áreas de almacenamiento (una de datos y otra de índice o directorio), podemos tener en el directorio más de un archivo, como se verá en el siguiente ejemplo.

Dirección	Nombre	Otros Campos
0	Alarcón	• • •
1	Barra	• • •
2	Campos	• • •
3	Iglesias	• • •
4	León	• • •
5	Mora	• • •
6	Opazo	• • •
7	Pérez	• • •
8	Ríos	• • •

Si uno de los campos es **ESTADO CIVIL**, entonces se puede tener un archivo en el cual está la siguiente información:

Empleados casados:

0	2	3	5	8

Cada casillero es un apuntador a cada registro de datos correspondiente

Si uno de los campos es **HIJOS**, entonces se puede tener un archivo como el siguiente:

Empleados con hijos

0	5	8

O si otro de los campos es el **DEPARTAMENTO** en el que trabaja cada empleado:

Empleados de Ventas

1	4	5	6	8

Con esta organización, la formación de esquemas compuestos de acceso puede ser hecha directamente sobre los directorios. Así, si se necesita conocer a los empleados de ventas que se encuentran casados, sólo basta con realizar la intersección de las dos listas correspondientes.

Desde un punto de vista más purista, el uso del término inversión implica que los valores de los datos indexados han sido sacados del registro de datos, y residen solamente en el correspondiente índice de inversión. Así, el archivo **completamente invertido** tiene un índice de inversión para cada campo de datos. O sea, el registro podría excluir el campo.

En el ejemplo anterior, el campo ESTADO CIVIL, HIJOS y DEPARTAMENTO pueden no estar presentes. Para realizar las actualizaciones, se debe contemplar la existencia de estos otros archivos o directorios.

Por otro lado, un archivo completamente invertido podría no tener campos de datos.

Si un archivo no es completamente invertido, pero tiene al menos un índice de inversión, entonces se dice que es **parcialmente invertido**.

Un índice de inversión puede construirse sobre un archivo secuencial indexado, con lo que se podrá tener un archivo que puede ser accesado por más de una clave. Las otras podrían llamarse **claves secundarias**.

Para más ejemplos, ve el capítulo de búsqueda externa (capítulo 8), listas invertidas, en el libro “Estructura de Datos” de Cairó/Guardati.