

Archivos

BASE DE DATOS

la base de datos es una colección de elementos interrelacionados con un propósito específico vinculado a problemas del mundo real
necesitan ser preservadas más allá de la ejecución de un algoritmo por lo tanto deben residir en archivos

PROPIEDADES IMPLÍCITAS

- las bases de datos con una representación de aspectos del mundo real
- es una colección coherente de datos que tienen cierta relación lógica
- tiene un propósito específico
- las bases de datos están sostenidas físicamente en almacenamiento persistente porque más allá de la ejecución de un algoritmo, deben residir en algún lado. No podría estar en RAM

BUFFER

el buffer es una memoria intermedia entre el archivo y el programa donde los datos residen provisoriamente hasta ser almacenados definitivamente en la memoria secundaria o donde los datos residen una vez recuperados de dicha memoria secundaria. Ocupa lugar en la RAM

el uso del buffer se justifica con cuestiones de performance ya que las operaciones en RAM tardan nanosegundos y en otras memorias milisegundos, si hago varias operaciones, el tiempo será notable

la diferencia principal entre RAM y otra memoria secundaria está en

- capacidad: la RAM es más limitada que un disco rígido por ejemplo
- tiempo de acceso: RAM mide acceso en nanosegundos 10^{-9} , y otro lo hace en milisegundos 10^{-3}

ARCHIVOS

los archivos son una colección de registros guardados en almacenamiento secundario

abarcán entidades con aspecto común para un propósito general. Son capaces de retener información luego de apagar la computadora

ACCESOS

hay 3 formas de acceder a los datos de un archivo

secuencial físico	secuencial indizado	directo
acceso a registro uno tras uno y en el orden en el que están guardados	acceso según orden establecido y en el que están almacenados ejemplo: las agendas, tengo un índice donde están todos los contactos ordenados pero después están uno tras otro	se accede directamente sin pasar por predecesores

TIPOS DE ORGANIZACIÓN

secuencia de bytes	registros y campos
no sé el comienzo y fin de cada uno generalmente archivos de texto, sin estructura	los campos son entidades más pequeñas, que en conjunto forman los registros, de forma que el trabajo es más fácil

TIPOS SEGÚN ACCESO

serie	secuencial	directo
cada registro es accesible solo luego de procesar su antecesor, simples de acceder	los registros son accesibles en orden de alguna clave	accedo directamente al registro deseado

TIPOS DE ARCHIVOS

lógico	físico
archivo usado desde el algoritmo. Cuando el algoritmo necesita operar un archivo, genera una conexión con el sistema operativo el cual se encarga de esta administración, conocido como <i>independencia física</i>	se refiere a que los archivos residen en una memoria secundaria y el sistema operativo se encarga de cuestiones relacionadas al lugar de almacenamiento. Es el archivo que reside en memoria secundaria y es administrado por el S.O.

VIAJE DE UN BYTE

para que el programa opere en el archivo, el S.O tiene un protocolo de como administrar esas operaciones involucra:

- **administrador de archivos:** programas del sistema operativo que tratan aspectos de archivos y dispositivos de entrada/ salida. Sus capas superiores contienen aspectos lógicos de datos (tabla) y establece si el archivo es compatible con la operación deseada. Las capas inferiores

tienen aspectos físicos del FAT (file allocation table) y determina dónde guarda datos

- **buffer entrada/salida:** agilizan la entrada/ salida de datos. El procesador entrada/salida es un dispositivo para transmisión desde o hacia el almacenamiento externo
- **controlador de disco:** controla operación de disco. Se coloca en la pista, sector y transfiere a disco

capas de protocolo de transmisión de un byte

- 1) el programa le pide al sistema operativo escribir el contenido de una variable en un archivo
- 2) S.O transfiere el trabajo al administrador de archivos
- 3) el administrador busca el archivo en su tabla y verifica las características
- 4) el admin. obtiene la ubicación física de dónde se guardará el byte
- 5) el admin. se asegura que el sector esté en el buffer y graba el dato
- 6) el admin. da instrucciones al procesador de E/S
- 7) el procesador de E/S encuentra el momento para transmitir el dato al disco y la CPU se libera
- 8) el proc. envía el dato al controlador de disco
- 9) el controlador prepara la escritura y transfiere el dato bit por bit en la superficie del disco

LONGITUD FIJA VS LONGITUD VARIABLE

la información de un archivo es homogénea o sea que todos los elementos almacenados en él son del mismo tipo, generando lo que se denomina

archivos con registros de *longitud fija*

la desventaja es que a veces hay desperdicio de espacio

en la *longitud variable* el uso de espacio en disco está optimizado pero conlleva operaciones mucho más minuciosas

OPERACIONES BÁSICAS EN ARCHIVOS

TIPO DE OPERACIÓN	DEFINICIÓN	OPERACIÓN
definición	como todo dato, los archivos necesitan definirse	<code>var archivo_logico: file of tipo_de_dato;</code>
correspondencia	indica a qué archivo físico corresponde un archivo lógico	<code>assign (nombre_logico, nombre_fisico);</code>
creación	indica que el archivo será creado y puedo escribirlo	<code>rewrite(nombre_logico);</code>
apertura	indica que ya existe y podré leerlo/ escribirlo	<code>reset(nombre_logico);</code>
cierra	transfiere definitivamente la información volcada sobre el archivo a disco, es decir transfiere el buffer a disco	<code>close(nombre_logico);</code>
EOF	controla las lecturas sobre el archivo y no exceder la longitud del mismo	<code>EOF(archivo);</code>
lectura	leer información del archivo	<code>read(nombre_logico, var_dato);</code>
escritura	escribir información en el disco	<code>write(nombre_logico, var_dato);</code>
tamaño	indica la cantidad de registros que tiene el archivo	<code>filesize(nombre_logico);</code>
posición dentro del archivo	indica posición actual del puntero	<code>filepos(nombre_logico);</code>
cambio de posición del puntero	retorna a una posición determinada, como si fuese un vector	<code>seek(nombre_logico, posicion);</code>

ACTUALIZACIÓN DE UN ARCHIVO MAESTRO CON UN ARCHIVO DETALLE

precondiciones

- hay un archivo maestro
- hay un único archivo detalle que modifica al maestro
- no todos los registros del maestro con necesariamente modificados
- ambos archivos están ordenados por igual criterio

ACTUALIZACIÓN DE UN ARCHIVO MAESTRO CON N ARCHIVOS DETALLE

ídem precondiciones y se implementa el procedimiento *mínimo* que determina el menor de los registros de cada archivo

CORTE DE CONTROL

proceso mediante el cual la información de un arhcivo es presentada en forma organizada de acuerdo con la estructura del archivo

precondiciones

- el archivo está ordenado por determinado criterio ejemplo, por provincia, ciudad, sucursal y vendedor
- se debe informar determinada información por sucursal, por ciudad, por provincia y en total

ARCHIVOS ESTRUCTURADOS

pueden ser registros y campos, cada uno puede ser de longitud fija o variable

- **campos de longitud variable:** nunca se cuándo empieza o termina. Para trabajar con longitud variable puedo usar
 - indicadores de longitud: leo valor que indica el tamaño del campo
 - delimitador de final: tiene un char o algo que indica cuándo termina
- **campos de longitud fija/ predecible:** puedo llegar a desperdiciar espacio. Hoy en día se usan más porque es barato y eficiente, cosa que no pasaba antes porque las memorias eran muy limitadas

CLAVES

a partir de la necesidad de extraer un registro específico en vez del archivo completo, surge la necesidad de identificar los registros con una llave o clave que se base en el contenido del mismo

una clave permite la identificación del registro y deben permitir generar orden en el archivo por ese criterio

- única/ primaria: identifican un elemento en particular dentro del archivo
- secundaria: reconocen un conjunto de elementos con igual valor

forma canónica: es una forma estándar para una llave, puede derivarse a partir de reglas bien definidas

ESTUDIO DE PERFORMANCE

puedo evaluar cada ciclo de máquinas que hace y cuántos accesos hace. Evalúa *caso promedio= mejor caso + peor caso*

- **acceso secuencial:** el mejor caso sería leer el primer registro y encontrar lo buscado, el peor es encontrarlo en el último. En promedio es $n/2$ comparaciones y el lineal porque a medida que aumentan los elementos, aumentarían las comparaciones
la lectura de bloques de registros se hace en buffer, si hago N comparaciones no significa que hago N accesos a discos, eso depende del tamaño del buffer
- **acceso directo:** requiere una sola lectura para traer el dato O(1) pero debo necesariamente conocerse el lugar donde comienza el registro requerido. Es preferible cuando necesito pocos registros porque no siempre es la elección apropiada para traer información

NRR

el NRR indica la posición relativa con respecto al principio del archivo

es aplicable solo para registros de longitud fija

CANTIDAD DE CAMBIOS

según la cantidad de cambios, los archivos pueden ser

estáticos	volátiles
hay pocos cambios y no necesita estructuras para agilizar los cambios	sometido a operaciones frecuentes como agregar, borrar o actualizar. Su organización debe facilitar cambios, requiere estructuras adicionales para mejor tiempo de acceso. Tienen a tener más tamaño

OPERACIONES ESCENCIALES

nombre	definición
alta	ingresa nuevos datos al archivo
modificación	altera el contenido de algún dato del archivo
consulta	presenta el contenido total o parcial del archivo
baja	quita información del archivo

BAJAS

puede ser **física o lógica** y se realiza cuando un elemento es quitado del archivo

- **BAJA FÍSICA:** consiste en borrar efectivamente la información del archivo, recuperando el espacio físico. Hay 2 técnicas
 - generar un nuevo archivo con los elementos válidos. Hablando de performance, necesito leer tantos datos como tenga el archivo y escribirlos por lo tanto, si tengo N registros, haré N lecturas y N-1 escrituras
 - usar el mismo archivo generando los reacomodamientos necesarios. En términos de performance, la cantidad de lecturas es N y en el peor de los casos habrá N-1 escrituras
- **BAJA LÓGICA:** se realiza una baja lógica sobre un archivo cuando el elemento que se desea quitar es marcado como borrado pero sigue ocupando el espacio en el archivo. La ventaja es que mejora la performance porque al eliminar un elemento, solo lo marco para indicar que no está disponible. La desventaja es el espacio en disco porque no se recupera y tiende a crecer continuamente el proceso de baja lógica marca la información de un archivo como borrada. Esta información sigue ocupando espacio en el disco rígido, entonces hay dos opciones
 - **recuperación de espacio:** periódicamente usa el proceso de baja física para realizar un proceso de compactación de disco
 - **reasignación de espacio:** recupera el espacio usando los lugares indicados como borrados para altas de nuevos elementos al archivo

el proceso de baja lógica no varía, en el borrado lógico hay nuevas consideraciones relacionadas al espacio disponible ya que en los registros de

longitud fija los elementos a eliminar e insertar son del mismo tamaño, en los de longitud variable esta precondición no está sea cual sea la estrategia, debe haber alguna forma para reconocerlos una vez eliminados y cómo reutilizar ese espacio. Realizada la eliminación para recuperar ese espacio puedo: copiar todo en un nuevo archivo a excepción de los registros eliminados (baja física) o asignar marcas especiales (baja lógica)

Para agregar un dato nuevo, busco el primer registro eliminado y lo inserto. Lo malo es que es lento para operaciones frecuentes y necesito una forma de saber de inmediato si hay lugar vacío en el archivo y una forma de saltar directamente a uno de esos lugares

- lista encadenada con registros disponibles (longitud fija): al insertar un registro nuevo en un archivo de registro con longitud fija, cualquier registro disponible es bueno. La lista no necesita tener un orden particular, ya que todos los registros son de longitud fija y todos los espacios libres son iguales
- longitud variable: uso una marca de borrado al igual que en los fijos. El problema de estos registros está en que no se puede colocar en cualquier lugar, para poder ponerlo debe caber, necesariamente entonces tengo que buscar el registro borrado de tamaño adecuado, aparece el concepto de **fragmentación**

la fragmentación puede ser

- interna: ocurre cuando se desperdicia espacio en un registro, se le asigna el lugar pero no lo ocupa totalmente. Ocurre en los registros de longitud fija. La solución será que el espacio libre pasará a ser un nuevo registro libre
- externa: espacio no asignado, no utilizado. Se genera por dejar espacios tan pequeños que no pueden ser usados. Hay algunas soluciones para este caso: unir espacios libres pequeños adyacentes para generar un espacio disponible mayor o minimizar la fragmentación

con respecto a los registros de longitud variable, el proceso de baja lógica no varía, en el borrado lógico hay nuevas consideraciones relacionadas al espacio disponible ya que en los registros de longitud fija los elementos a eliminar e insertar son del mismo tamaño, en los de longitud variable esta precondición no está, las soluciones son:

primer ajuste	mejor ajuste	peor ajuste
se selecciona la primer entrada disponible que pueda almacenar al registro hay poca búsqueda y no me preocupo por la exactitud del ajuste es el más rápido y genera fragmentación interna	elige la entrada que más se aproxime al tamaño del registro y se le asigna completa pero exige búsqueda genera fragmentación interna	selecciona la entrada más grande para el registro y se le asigna solo el espacio necesario, el resto queda para otro registro. genera fragmentación externa

- **modificaciones:** surgen consideraciones

cuando tengo registros de longitud variable, se altera el tamaño, si es menor no importa y si es mayor, no cabrá otros problemas serán claves duplicadas o cambiar claves del registro

PROCESO DE BÚSQUEDA

en la búsqueda entra en consideración la cantidad de accesos a disco (costo alto en memoria secundaria) y la cantidad de comparaciones (costo bajo, en memoria principal). Se tiene que hacer un análisis de situaciones según el archivo que se manipula

- 1) ARCHIVO SERIE: el *mejor caso* es encontrar el dato en el primer registro (1 lectura) y el *peor caso* es encontrarlo en el último registro (N lecturas), el *caso promedio* es $N/2$ lecturas por lo tanto depende de la cantidad de registros del archivo, siendo orden N también tengo la *búsqueda binaria*, que a partir de un archivo ordenado por claves que contiene registros de longitud fija, inicio la búsqueda a partir del archivo a la mitad y comparo la clave. Como es longitud fija, puedo acceder al medio del archivo. Si N es el # de registros la performance es $\log N$ y un promedio de $\log_2 N + 1/2$

ORDENACIÓN

tengo 3 métodos de ordenación según el caso de nuestro archivo

- 1) en RAM: si el archivo a ordenar cabe en la RAM lo traslado de memoria secundaria a principal y lo ordeno ahí pero solo puedo usarlo en archivos pequeños
- 2) claves en RAM: si el archivo no cabe en RAM, traslado las claves por las que deseo ordenar cada registro y así podré almacenar mayor cantidad de registros y así ordenar un archivo más grande
- 3) archivos grandes: diviso el archivo en particiones iguales, transfiero cada una a memoria principal y las ordeno y reescribo ordenadas en memoria secundaria, estaría realizando un merge generando un nuevo archivo ordenado



INDIZACIÓN

el propósito de ordenar es minimizar el acceso a memoria secundaria durante la búsqueda y dada esta situación podría usar una fuente de información adicional para mejorar aún más la performance

un **índice** es una estructura de datos adicional que permite agilizar el acceso a la información y es una estructura de datos de otro archivo con registros de *longitud fija* independiente al archivo original que se caracteriza por imponer un orden a un archivo sin que se reacomode

a la hora de buscar un dato, primero se busca el índice, obtenida la dirección real del dato en el archivo de datos, se accede ejemplo: sea una canción "here comes the rain again" se busca la clave primaria en el índice (puedo encontrarlo con búsqueda binaria porque es fijo). Hallada la referencia, accede directamente al archivo de datos según lo indicado con la referencia

operaciones básicas

- crear archivo, cargar el índice, si tengo más de un índice, creo los archivos de índice necesarios
- cargar índices en memoria
- reescritura del archivo de índice

la ventaja es que está almacenada en memoria principal, para la búsqueda binaria el costo de mantenimiento es bajo

desventaja: puede no caber en RAM.

el índice siempre puede persistir en disco y no importa si es grande porque está en el disco pero será caro porque no es eficiente. En su contraparte, logramos la persistencia. Actualmente no hay índice en RAM, solo operaciones

- clave primaria: es una clave única que fue elegida como primaria. Es la única que siempre tiene dirección fija, más eficiente
- clave secundaria: surge problema de repetición de información entonces uso clave + vector con puntero con concurrencias. Su único reacomodamiento es en el archivo índice al agregar concurrencias en lista invertida. Según caso para cambiar, modificar, borrar o agregar grabaciones
- clave candidata: claves que no admiten repeticiones de valores para sus atributos, similar a una clave primaria pero por alguna cuestión no fueron seleccionadas como clave primaria.

un **índice secundario** es una estructura adicional que permite relacionar una clave secundaria con una o más claves primarias. Para acceder a un dato deseado, primero se accede al índice secundario por clave secundaria, allí se obtiene la clave primaria y se accede con dicha clave al índice primario para obtener finalmente la dirección efectiva del registro que contiene la información buscada

Árboles

los índices resultaron ser una forma de ordenar archivos sin modificar su orden, pero la búsqueda binaria aún resulta costosa e inefficiente. Entonces, surge la idea de los árboles

un árbol es un tipo abstracto de datos ampliamente usado que imita la estructura jerárquica de un árbol, con un valor en la raíz y subárboles con un nodo padre, representado como un conjunto de nodos enlazados. Los árboles se implementan en disco para lograr persistencia. Los punteros son almacenados en RAM pero el foto de estudio es cómo uso esos punteros en memoria principal

tienen una buena performance de búsqueda y es una estructura no lineal en la que cada nodo tiene a lo sumo dos hijos. La estructura tiene sentido cuando está ordenada

CONSTRUCCIÓN DE ÁRBOLES

- 1) agregar el nuevo elemento de datos al final del archivo
- 2) busca el parente de dicho elemento. Para ello se recorre el archivo desde raíz hasta nodo terminal
- 3) actualiza el parente, haciendo referencia a la dirección del nuevo hijo

NODOS HERMANOS

los *nodos hermanos* son aquellos que tienen el mismo nodo padre

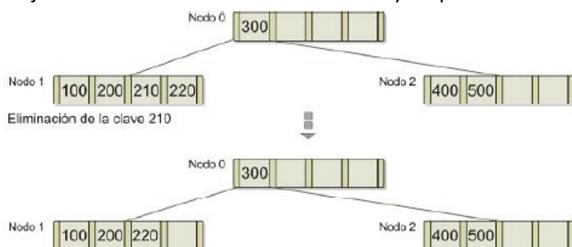
los *nodos adyacentes hermanos* con aquellos que siendo hermanos son además dependientes de punteros consecutivos del parente

ÁRBOL BALANCEADO

un árbol balanceado resulta cuando su altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más grande

tienden a desbalancearse y en general su construcción es de forma ascendente para evitar este problema. Siempre que construyo voy a tener en todo el árbol M claves y $M+1$ punteros nulos en los nodos terminales

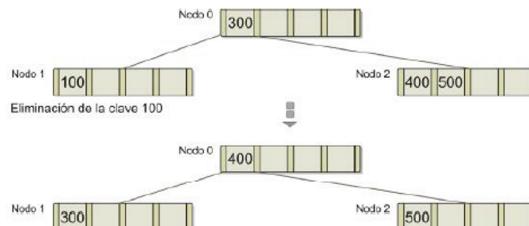
- búsqueda: los mejores casos llevarán una lectura y el peor caso será H lecturas donde H es la altura del árbol
- inserción: los registros se insertan en un nodo terminal. Hay varios casos posibles
 - 1) el registro tiene lugar en el nodo terminal y solo reacomodamos los valores en ese nodo
 - 2) el registro no tiene lugar (overflow) en el nodo terminal entonces el nodo se divide y los elementos se reparten. Para esto, entre los nodos hay una promoción (sube un elemento) al nivel superior y esto puede propagarse y generar una nueva raíz
el *mejor caso* (sin overflow) será H lecturas y una escritura y el *peor caso* (con overflow) será H lecturas y $2(H+1)$ escrituras cuando un overflow ocurre, significa que en el nodo no hay cantidad disponible para almacenar un nuevo elemento de datos
- eliminación: siempre elimino en nodos terminales. Si se elimina un elemento que no está en el nodo terminal entonces debo llevarlo primero al nodo terminal y reemplazar esa raíz con el mínimo valor de su hijo derecho. Hay varias posibilidades ante la eliminación
 - 1) mejor caso: borra un elemento del nodo y no produce underflow, solo hacemos reacomodaciones



- 2) peor caso: se produce underflow (elimino y # de nodos < $[M/2]$ -1 entonces hay dos soluciones

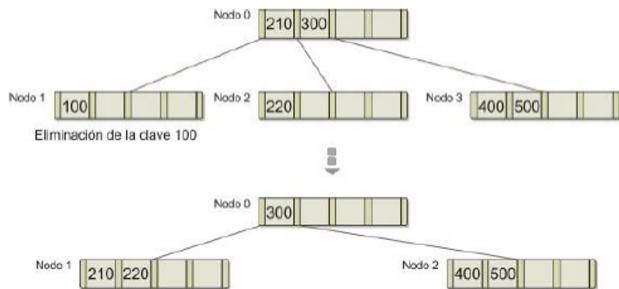
es importante primero saber que los *nodos hermanos* son aquellos nodos que tienen el mismo nodo parente y los *nodos hermanos adyacentes* son aquellos nodos que siendo hermanos además son dependientes de punteros consecutivos del parente

- i. redistribuir: alguno de los hermanos adyacentes del nodo en underflow puede cederle un elemento. Un hermano adyacente podrá ceder un elemento si al hacerlo no entra en underflow. El elemento más grande del hermano adyacente izquierdo o el más chico del hermano adyacente derecho (según la política) pasa al parente y el elemento divisor del parente pasa al nodo en underflow



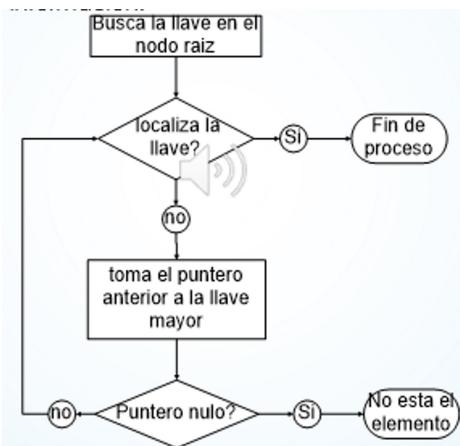
al eliminarse el 100, el nodo que lo contiene entra en underflow. Como el hermano adyacente derecho puede ceder un elemento sin quedar por debajo del mínimo de elementos permitidos pasa la clave 400 al parente y el elemento 300 del nodo 0 pasa al nodo que estaba en underflow.

- ii. fusión: ocurre cuando ningún hermano adyacente del nodo en underflow puede ceder un elemento sin quedar por debajo del mínimo permitido. En este caso, se produce la fusión. actúa de la siguiente forma: se unen dos nodos hermanos adyacentes más el elemento divisor del parente y formar un solo nodo con la unión de todos los elementos involucrados



al eliminar el elemento 100, el nodo que lo contiene queda por debajo del mínimo permitido. Como el hermano adyacente derecho, el nodo 2 no puede ceder un elemento, ya que si lo hiciera también entraría en underflow, se fusionan ambos nodos, uniendo en el nodo resultante los elementos de ambos, en este caso solo el 220, más el elemento separador que se encontraba en el padre, la clave 210 el mejor caso tiene H lecturas y una escritura y el peor caso, donde concatenar lleva a decrementar el nivel el árbol en 1, es $2h-1$ lecturas y $4+1$ escrituras

- búsqueda:



en el mejor caso habrá una sola lectura, en el peor paso habrá h lecturas, $h =$ altura del árbol. Dado un árbol balanceado de orden M , si el numero de elementos del árbol es N , hay $N+1$ punteros nulos

cada nodo (menos la raíz) tiene mínimo $[M/2]$ hijos
la raíz tiene mínimo 2 hijos
M es el orden y representa la máxima cantidad de descendientes de un nodo y $M-1$ es la cantidad de elementos posibles en un nodo
los nodos terminales SIEMPRE ESTÁN A IGUAL NIVEL
los nodos terminales tienen mínimo $[M/2]-1$ registros y máximo $M-1$

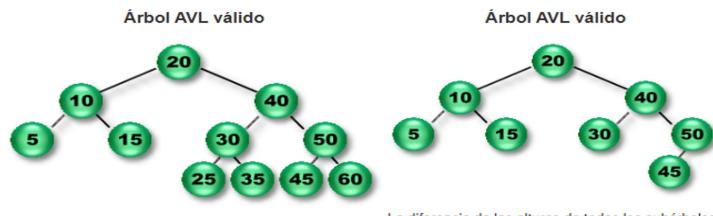
POLÍTICAS

las políticas determinarán con cuál de los hermanos adyacentes se realizará la redistribución en caso de producirse underflow

- 1) política derecha: se intenta redistribuir con el hermano adyacente derecho, llevándose a cabo dicha acción en caso de ser posible. Si no es posible, se fusiona con el hermano adyacente derecho
- 2) política izquierda: se intenta redistribuir con el hermano adyacente izquierdo, llevándose a cabo dicha acción en caso de ser posible. Si no es posible, se fusiona con el hermano adyacente izquierdo
- 3) política izquierda o derecha: determina que en primera instancia la redistribución debe realizarse con el hermano adyacente izquierdo, si no es posible, con el hermano adyacente derecho y en caso de no poder redistribuir con ninguno, se fusiona con el izquierdo
- 4) política derecha o Izquierda: determina que la redistribución debe realizarse primero con el hermano adyacente derecho, en caso de no ser posible, con el hermano adyacente izquierdo y en caso de no poder redistribuir con ninguno, se fusiona con el hermano adyacente derecho
- 5) caso especial: en cualquier política si se tratase de un nodo hoja de un extremo del árbol debe intentarse redistribuir con el hermano adyacente que el mismo posea

ÁRBOLES AVL

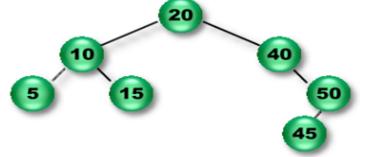
los árboles AVL es un árboles balanceado en altura en el que las inserciones y eliminaciones se efectúan con un mínimo de accesos busca que el trayecto más largo restándole el más corto, tenga una diferencia solo de 1. Tiene una eficiencia similar pero no está balanceado perfectamente por esa diferencia de 1 y para lograrlo tengo algoritmos complejos con muchos accesos. Entonces un árbol AVL es un árbol balanceado en altura donde el delta determinado es 1, o sea el máximo desbalanceo posible es 1



La diferencia de las alturas de todos los subárboles de los nodos es menor o igual a 1.

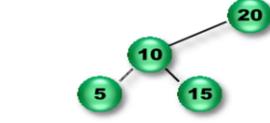
La diferencia de las alturas de todos los subárboles de los nodos es menor o igual a 1.

No es un Árbol AVL válido



Obsérvese que la diferencia de las alturas de los subárboles del nodo 20 es 1, pero su subárbol derecho no es AVL (porque la diferencia de alturas entre los subárboles de la clave 40 es 2).

No es un Árbol AVL válido



La diferencia de alturas entre los subárboles de la clave 20 es 2 porque la altura del subárbol izquierdo de 20 es 1 y la altura del subárbol derecho es -1.

en conclusión, para los árboles AVL y binarios

- la estructura de datos debe ser respetada
- búsqueda para árbol binario $\log_2(N + 1)$
- búsqueda para árbol AVL 1,44 $\log_2(N + 1)$
- tienen igual performance para el peor caso

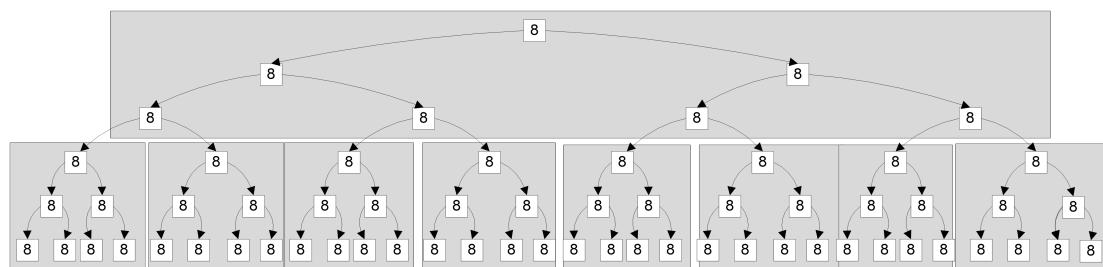
ÁRBOLES BINARIOS PAGINADOS

las operaciones de lectura y escritura de datos en un archivo usando buffers presentan una mejora de performance. Cuando se transfiere información desde o hacia el disco, dicha transferencia no se limita a un registro, sino que son enviados en conjunto de registros, es decir, los que quepan en un buffer de memoria. Las operaciones de lectura y escritura de datos de un archivo usando buffers presentan una mejora de performance.

un árbol binario paginado se divide en páginas donde cada página contiene un conjunto de nodos, los cuales están ubicados en direcciones físicas cercanas así, cuando se transfieren datos, no se accede al disco para transferir pocos bytes, sino que se transfiere una página completa. Una organización de este tipo reduce accesos a disco

al dividir un árbol en páginas, es posible hacer búsquedas más rápidas de datos. Para analizar la performance resultante deberíase tenerse en cuenta la cantidad de nodos que caben en una página

por ejemplo: suponiendo que en un buffer caben 255 elementos, el tamaño de cada página sería de 255 nodos, resultando la performance final de búsqueda del orden de $\log_{256} N$, es decir $\log_{k+1} N$, siendo N la cantidad de claves del archivo y k la cantidad de nodos por página. Se disminuye la profundidad del árbol



ÁRBOLES MULTICAMINO

los árboles multicamino es una estructura de datos en la cual cada nodo puede contener k elementos y k+1 hijos

se define el concepto de orden de un árbol multicamino como la máxima cantidad de descendientes posibles de un nodo

por ejemplo: el orden de un árbol binario es 2, el orden de un árbol ternario será de 3 y en general un árbol multicamino de orden K contendrá un máximo de K+1 descendientes

ÁRBOL B

son árboles multicamino con una construcción especial que permite mantenerlos balanceados a bajo costo
se caracteriza por

- cada nodo del árbol puede contener como máximo M descendientes y M-1 elementos
- la raíz puede no disponer de descendientes pero cuando lo haga, deberá contar como mínimo con dos
- un nodo con X descendientes directos contiene X-1 elementos
- los nodos terminales tienen mínimo [M-1] -1 elementos y máximo MM-1
- los nodos no terminales tienen mínimo [M/2] elementos
- todos los nodos terminales están al mismo nivel

la eficiencia de búsqueda consiste en contar los accesos al archivo de datos. El resultado es un valor acotado en el rango entero [1, H] siendo H la altura del árbol. Si el elemento está en la raíz, habrá un solo acceso si no, habrá H accesos

en la *eficiencia de inserción* está dado por H lecturas, 1 escritura en el mejor caso y H lecturas y $(2H) + 1$ escrituras en el peor caso
en la *eficiencia de la eliminación*, en el mejor caso hay H lecturas y 1 escritura y en el peor caso hay $2H-1$ lecturas y $H-1$ escrituras

ÁRBOL B*

los árboles B* son una variante de los árboles B, su característica principal es que cada nodo está lleno por lo menos en 2/3 partes

- cada nodo tiene como máximo M descendientes y M-1 elementos
- la raíz tienen al menos dos descendientes
- un nodo con M descendientes tiene M-1 elementos
- todos los nodos terminales están a igual nivel

la variante principal es que se llena al 66%, cosa que en los B sucede al 50%

operaciones

- búsqueda: igual que B común
- inserción: usa el concepto de política. La primera acción que se toma es buscar el nodo en donde corresponde insertar el elemento, si la inserción produce overflow, entra en consideración la política
 - 1) derecha: redistribuir cada nodo adyacente hermano de la derecha
 - 2) izquierda o derecha: si el nodo de la derecha está lleno y no se puede redistribuir, se busca el de la izquierda
 - 3) izquierda y derecha: se intenta redistribuir con el izquierdo, si no puedo, lo hago con el derecho y si tampoco puedo, se fusionan los tres nodos

mejoran la eficiencia de los árboles B aumentando la cantidad mínima de elementos en cada nodo pero el costo que pagan es que las operaciones de inserción son más lentas.

ÁRBOL B+

un árbol B+ consiste en un conjunto de grupos de registros ordenados por clave en forma secuencial junto con un conjunto de índices que proporciona acceso rápido a los registros

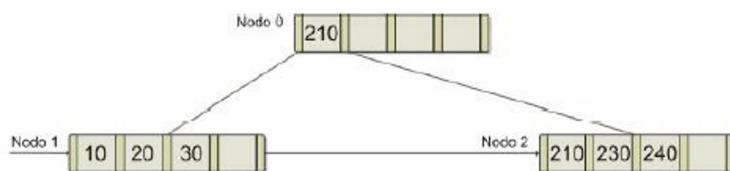
resulta ser un árbol multicamino donde

- cada nodo menos la raíz y hojas, tienen entre $M/2$ y M hijos
- la raíz tiene al menos 2 descendientes
- todas las hojas están en igual nivel
- un nodo que no sea hoja si tiene M descendientes tiene M-1 claves
- los nodos terminales representan un conjunto de datos y son enlazados entre ellos

la última definición establece la diferencia principal entre un árbol B y un árbol B+. Para poder hacer un acceso secuencial ordenado a todos los registros, es necesario que cada elemento aparezca almacenado en un nodo terminal. Así los árboles B+ diferencian los elementos que constituyen datos de aquellos que son separadores

operaciones

- insertar: al comenzar la creación, el único nodo disponible actúa como punto de partida para búsquedas como para acceso secuencial. En el momento que quiera insertar y el nodo se satura, se produce una división. Se genera un nuevo nodo donde se redistribuyen los elementos

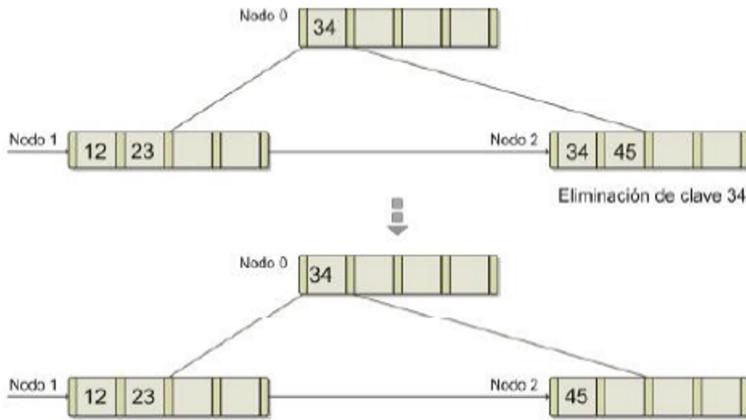


ahora se dispone de tres nodos, los nodos terminales contendrán todos los elementos, y el nodo raíz tiene al elemento que actúa como separador. Se debe notar que la clave utilizada como separador es la misma que está contenida en el nodo terminal, es decir, se utiliza una copia del elemento y no el elemento en sí.

es importante el enlace que existe entre los nodos del último nivel. Esto representa la posibilidad de acceso secuencial a los datos del árbol comenzando la lista en el nodo hoja que se encuentra más a la izquierda, en este caso el nodo 1.

el proceso de creación del árbol B+ sigue los lineamientos presentados anteriormente para árboles B. Los elementos siempre se insertan en nodos terminales. Si se produce una saturación, el nodo se divide y se promociona una copia (aquí yace la diferencia) del menor de los elementos mayores, hacia el nodo padre. Si el padre no tuviera espacio para contenerlo se dividirá nuevamente. Se debe notar que en caso de dividir un nodo no terminal, se debe promocionar al padre el elemento en sí y no una copia del mismo, es decir, sólo ante la división de un nodo terminal se debe promocionar una copia

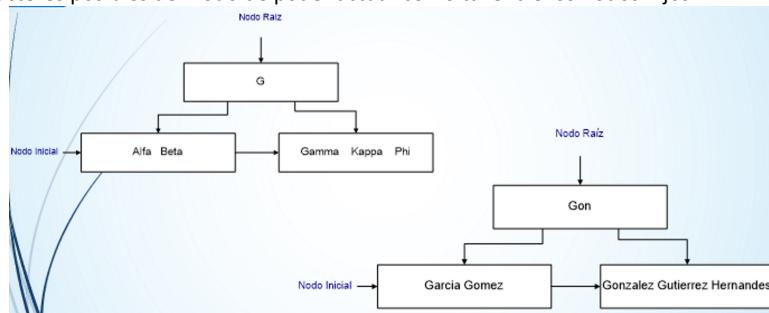
- eliminación: siempre se borra de un nodo terminal, y si hubiese una copia de ese elemento en un nodo no terminal, esta copia se mantiene porque sigue actuando como separador



se muestra un árbol B+ con orden 5 en donde se elimina la clave 34 contenida en el nodo 2. Como puede observarse al borrar la clave 34, el nodo 2 no entra en underflow, razón por la cual el elemento puede borrarse normalmente. Por otro lado es importante notar que al producirse la eliminación de la clave 34 no se borra la copia contenida en el nodo 0, la cual continúa siendo separador del árbol
en el caso de eliminar una clave y el nodo entra en underflow, habrá que hacer una redistribución y fusión de las mismas tanto en las hojas como en los nodos índice

ÁRBOL B+ DE PREFIJOS SIMPLES

un árbol B+ de prefijos simples es un árbol B+ en el cual el conjunto de índices está constituido por separadores más cortos. El separador siempre tiene la menor cantidad de caracteres posibles de modo de poder actuar como tal entre los nodos hijos



CONCLUSIÓN ÁRBOL B Y ÁRBOL B+

	árbol B	árbol B+
ubicación de datos	en cualquier nodo	solo nodos terminales
tiempo de búsqueda	=	=
procesamiento secuencial	lento	rápido(con punteros)
inserción de eliminación		puede requerir + tiempo

CONCLUSIÓN ÁRBOLES DE LA FAMILIA B

- manejo de nodos
- todos los nodos terminales están a la misma distancia que la raíz
- crecen de abajo hacia arriba
- son "bajos y anchos" a diferencia de los binarios que son "altos y delgados"
- en los árboles B+ toda la información de las claves está contenida en nodos terminales. Los no terminales actúan como separadores y poseen una copia de los elementos contenidos en los nodos terminales

ARCHIVOS SECUENCIALES INDIZADOS

los archivos secuenciales indizados permiten una mejor recorrida por algún tipo de orden. Indizado= ordenado por una llave, secuencial= acceder por orden físico, devolviendo el registro en orden de llave

intenta compatibilizar el orden físico de los elementos con un acceso indizado, de acuerdo a lo definido para árboles B

Hashing

un archivo directo es un archivo en el cual cualquier registro puede ser accedido sin acceder antes a otros registros. Los archivos directos están almacenados de diferentes formas pero se debe lograr una rápida recuperación de la información contenida en el archivo, que es el objetivo principal de los archivos directos
cuando era secuencial había $N/2$ accesos promedios, en ordenado teníamos $\log_2 N$ y en árboles había 3 o 4 accesos pero aún sigue siendo costoso el acceso
el hashing es un método que mejora la eficiencia de los árboles balanceados

HASHING

el hashing es una técnica para generar una dirección única para una llave dada. La dispersión se usa para cuando se requiere acceso rápido a una llave
otras definiciones son

- técnica que convierte la clave asociada a un registro de datos en un número aleatorio el cual posteriormente es usado para determinar dónde se almacena dicho registro
- técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en dirección de almacenamiento

algunos otros conceptos

- los archivos secuenciales indizados (que vimos hasta ahora) tenía un archivo de datos, archivo con índice primario y archivos con índices únicos o secundarios. Funcionaba: buscaba el índice primario en los índices secundarios y luego pasaba al archivo de datos. Era un camino eficiente. El mejor de los casos eran 2 accesos, uno para el índice primario y otro para el archivo de datos
- los archivos directos requieren UN ACCESO a disco. Por consecuencia, no puede haber estructuras adicionales porque lo que quiero hacer es encontrar el dato en un solo acceso entonces lo que se hace es organizar por un único criterio el archivo de datos, ese criterio será una clave primaria. Entonces, la clave primaria ordena el archivo

CARACTERÍSTICAS

- no requiere almacenamiento adicional o sea de un índice
- facilita inserción y eliminación rápida de registros
- encuentra registros con muy pocos accesos al disco en promedio

COSTO

- no puedo usar registros de longitud variable
- no puede haber orden físico de datos, o sea de ninguna manera podría sacar un listado ordenado de datos
- no permito llaves duplicadas ya que uso clave primaria



DETERMINACIÓN DE DIRECCIÓN

- 1) la clave se convierte en un número casi aleatorio
- 2) # se convierte en una dirección de memoria a través de una función de hash
- 3) el registro se guarda en esa dirección

si la dirección está ocupada puede haber una colisión/ overflow la cual tendrá un tratamiento especial

EFICIENCIA

se estudia a partir de

- función de hash: es una función matemática. Por ejemplo: la función es $F = x^2$ sería una mala elección porque solo devuelve números mayores que 0 y para dos claves diferentes devolvería la misma dirección como -1 y 1
- tamaño de nodos
- densidad de empaquetamiento
- método de tratamiento de overflow

TIPOS DE DISPERSIÓN

- 1) hashing con espacio de direccionamiento estático: política donde el espacio disponible para dispersar los registros está fijado previamente. Así la función de hash aplicada a una clave da como resultado una dirección física posible dentro del espacio del archivo
- 2) hashing con espacio de direccionamiento dinámico: política donde el espacio disponible para dispersar registros aumenta o disminuye en función de las necesidades de espacio en cada momento del archivo

FUNCIÓN DE HASH

una función de hash es una caja negra que a partir de una clave se obtiene la dirección donde debe estar el registro
es un módulo que tiene como entrada una clave y tiene como salida una dirección en disco
se diferencia con los índices porque no hay relación entre llave y dirección (no como en árboles que si insertaba algo nuevo, los punteros del padre se cambiaban) dos llaves distintas pueden transformarse en iguales dirección (colección). El hash es una caja negra porque no sabemos qué hacemos ni donde ubica cada dato, dos datos ingresados consecutivamente pueden ser ubicados cerca como lejos también

según FOD

colisión	overflow
situación en la que un registro es asignado a una dirección que está usada por otro registro ejemplo: el registro A para en dirección 10 y el B también es asignado a la dirección 10	situación en la que un registro es asignado a una dirección que está usada por otro registro y NO queda espacio para este nuevo. ejemplo: el registro A para en dirección 10 y el B también es asignado a la dirección 10 estarían coincidiendo pero no hay lugar en el nodo para almacenar

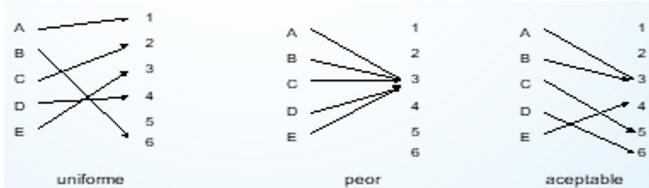
debo tener un algoritmo de hash sin colisiones o que nunca produzcan overflow (perfecto e imposible)

SOLUCIONES PARA COLISIONES

- elegir un algoritmo de dispersión perfecto
- buscar una función de dispersión que distribuya su resultado de la forma más aleatoria posible
- usar más espacio de disco. Si se intentan distribuir 10 registros en 10 lugares, significa que hay un lugar disponible para cada registro, con lo cual las posibilidades de colisión son altas. Ahora bien, si se dispone de 100 lugares para cada uno de los 10 registros, se tendría un archivo con 10 direcciones posibles para cada registro. Esto, sin duda, debería disminuir el número de colisiones que se producen
- ubicar o almacenar más de un registro por cada dirección física en el archivo.

ALGORITMO DE DISPERSIÓN

para estos algoritmos espero repartir registros uniforme (ejemplo: si tengo 10 cajas y 10 bolitas, luego de repartir tendría que tener una bolita en cada caja o sea lo más equitativo posible) y aleatoriamente (las claves son independientes y ninguna influye sobre otra). Un caso que hay dependencia sería en los árboles donde cada nodo depende del padre



TAMAÑO DE NODOS/ CUBETA

dentro de los parámetros que afectan la eficiencia del método de dispersión está presente el tamaño o capacidad de cada nodo. cada nodo puede contener más de un registro. A mayor tamaño, tengo menos overflow, mayor fragmentación y una búsqueda más lenta dentro del nodo

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

$DE = \text{número de registros del archivo} / \text{capacidad total del archivo}$ o podría decirse que es número de registros usados/ capacidad del archivo
si se debieran espaciar 30 registros entre 10 direcciones con capacidad de cinco registros por cada dirección, la DE sería de 0.6 o 60% ($30 / 5 * 10$)
cuanto mayor sea la DE, mayor será la posibilidad de colisiones, dado que en ese caso se dispone de menos espacio para espaciar registros. Por el contrario, si la DE se mantiene baja, se dispone de mayor espacio para espaciar registros y, por ende, disminuye la probabilidad de colisiones.
por otra parte, cuando la DE se mantiene baja, se desperdicia espacio en el disco, dado que se utiliza menor espacio que el reservado, generando fragmentación

la DE no es constante, al principio es baja

PERFORMANCE

es necesario analizar el comportamiento de un archivo directo cuando encontrar un registro requiere un solo acceso y cuando requiere más accesos, por lo tanto debo estimar el overflow lo que requiere analizar probabilísticamente si la inserción de un registro genera o no colisión y analizar si la colisión genera o no overflow

será necesario elementos básicos de probabilidades y la distribución de Poisson

para estimar overflow

- N # de cubetas
- C capacidad del nodo
- R registros del archivo
- $DE = \frac{R}{C \times N}$
- ejemplo si tengo mil cubetas y 100 capacidad de nodo tendré 1000×100 direcciones
- probabilidad de que una cubeta reciba un registro, está dado por la distribución de Poisson

$$\circ P(I) = \frac{R!}{I! * (R - I)!} * \left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

si quisiera calcular la probabilidad de overflow hacia P(C+1)

si tuviese dos claves, cuál sería la probabilidad de que dos claves vayan a la misma cubeta= $P(BB) = P(B) * P(B)$ si son dos eventos independientes

y siempre lo será porque en el hashing las claves son independientes
 si la secuencia fuera 3 claves, $P(BBB) = P(B) * P(B) * P(B) = (1/N)^3$
 o $P(BAA) = P(B)*P(A)*P(A) = (1/N) * (1-1/N)^2$

- la secuencia de R llaves, que I caigan en un nodo es la probabilidad $(1/N)^I * (1-1/N)^{R-I}$
- formas de combinar esta probabilidad hay $\frac{R!}{I!(R-I)!}$
- función de Poisson, probabilidad que un nodo tenga I elementos $P(I) = \frac{\left(\frac{R}{N}\right)^I * e^{-\frac{R}{N}}}{I!}$

en general si hay N direcciones, el # esperado de direcciones con I registros asignados, es $N * P(I)$
 con buen algoritmo, cantidad y nodos, el hashing es bueno y se reduce el overflow

MÉTODOS DE TRATAMIENTO DE OVERFLOW

un overflow ocurre cuando un registro es direccionado a un nodo que no dispone de capacidad para almacenarlo. Se presentan los siguientes métodos para resolver el overflow:

- saturación progresiva: el método consiste en almacenar el registro en la dirección siguiente más próxima al nodo donde se produce saturación
- saturación progresiva encadenada: un elemento que se intenta ubicar en una dirección completa es direccionado a la inmediata siguiente con espacio disponible. La diferencia radica en que, una vez localizada la nueva dirección, esta se encadena o enlaza con la dirección base inicial, generando una cadena de búsqueda de elementos
- doble dispersión: la saturación tiende a agrupar en zonas contiguas, búsquedas largas cuando al densidad tiende a uno. El método consiste en disponer de dos funciones de hash. La primera obtiene a partir de la llave la dirección de base, en la cual el registro será ubicado. De producirse overflow, se utilizará la segunda función de hash. Esta segunda función no retorna una dirección, sino que su resultado es un desplazamiento.
- área de desborde separado: aquí se distinguen dos tipos de nodos: aquellos direccionables por la función de hash y aquellos de reserva, que solo podrán ser utilizados en caso de saturación pero que no son alcanzables por la función de hash

HASH CON ESPACIO DE DIRECCIONAMIENTO ESTÁTICO

necesita un número de direcciones fijas, virtualmente imposible. Cuando el archivo se llena hay una saturación excesiva y redispersar con una nueva función llevaría muchos cambios

HASH CON ESPACIO DE DIRECCIONAMIENTO DINÁMICO

(según Wikipedia xd) las tablas hash se presentaron como una alternativa hacia las estructuras tipo árbol porque permitían el almacenamiento de grandes volúmenes de información y algoritmos eficientes para la administración sobre estas estructuras (inserción, eliminación y búsqueda).

sin embargo, presentan dos grandes problemas:

- no existen funciones hash perfectas que permitan asegurar que por cada transformación de un elemento habrá una única correspondencia en la clave que contiene este elemento
- son estructuras estáticas que no pueden crecer ya que necesitan un tamaño fijo para el funcionamiento de la estructura

para solucionar el segundo problema se implementa la utilización de métodos totales y métodos parciales. Convirtiendo la tabla hash en una estructura dinámica capaz de almacenar un flujo de información y no una cantidad fija de datos

es la solución del direccionamiento estático, reorganiza tablas sin mover muchos registros y usando técnicas que asumen bloques físicos que pueden usarse o liberarse

posibilidades de espacio dinámico

- hash virtual
- hash dinámico
- hash extensible: adapta el resultado de la función del hash de acuerdo al número de registros que tanta el archivo y de las cubetas necesitadas para su almacenamiento. La función genera una secuencia de bits, normalmente 32

HASH EXTENSIBLE

el método de hash extensible es una alternativa de implementación para hash con espacio de direccionamiento dinámico. El principio del método consiste en comenzar a trabajar con un único nodo para almacenar registros e ir aumentando la cantidad de direcciones disponibles a medida que los nodos se completan

NO USA DENSIDAD DE EMPAQUETAMIENTO y su principal problema es que las direcciones de nodos no están prefijadas a priori por lo tanto la función de hash no retorna una dirección fija por lo tanto debo cambiar la política de función de dispersión. Para el método extensible, la función de hash retorna un string de bits. La cantidad de bits que retorna determina la cantidad máxima de direcciones a las que puede acceder el método.

cómo trabaja

- usa solo bits necesarios de acuerdo a cada instancia del archivo
- los bits tomados forman la dirección del nodo que se usará
- si se intenta insertar a una cubeta llena deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nodo nuevo. Para ello se toma un bit más
- la tabla tendrá tantas entradas como 2^i , siendo i el número de bits actuales para el sistema

ejemplo

Clave	Secuencia de bits
Alfa 0011 0011
Beta 0110 0101
Gamma 1001 1010
Epsilon 0111 1100
Delta 1100 0001
Tita 0001 0110
Omega 1111 1111
Pi 0000 0000
Tau 0011 1011
Lambda 0100 1000
Sigma 0010 1110

UNLP - Facultad
de Informática

/* ver explicación de inserción en "Introducción a las bases de datos- Bertone, Thomas, página */

CONCLUSIONES

- métodos de búsqueda de información en un archivo presentan ventajas y desventajas en cada caso
- el mayor porcentaje de operaciones sobre los archivos son de consultas
- los árboles balanceados representaron una solución aceptable en términos de eficiencia, reduciendo el número de accesos a disco. Para árboles balanceados se discutieron 3 alternativas: B, B* y B+. Las dos primeras plantean el mismo comportamiento pero se diferencian en que B* completa más los nodos. La variante B+ presenta una ventaja adicional y es que además de permitir la búsqueda eficiente, también permite el recorrido secuencial del archivo
- la dispersión con espacio de direccionamiento estático asegura encontrar los registros buscados con un solo acceso a disco
- si se desea asegurar siempre un acceso a disco para recuperar la información, la variante de hash extensible, que usa espacio de direccionamiento dinámico, lo logra. El costo que debe asumirse con esta técnica es mayor procesamiento cuando una inserción genera overflow

ELECCIÓN DE ORGANIZACIÓN

la elección tendrá como objetivo acomodar datos para satisfacer rápidamente los requerimientos

organización	acceso a un registro por clave primaria	acceso a todos los registros por clave primaria
ninguna	muy ineficiente	muy ineficiente
secuencial	poco eficiente	eficiente
secuencial indizada	muy eficiente	el más eficiente
hash	el más eficiente	muy ineficiente

cosas para tener en cuenta a la hora de hacer la elección

- captar los requerimientos de usuario
- examinar características del archivo como número y tamaño de registros
- requerimientos del usuario, como tipos de operaciones, números de accesos a archivos
- características del hardware: tamaño de sectores, bloques, pistas, cilindros, etc.
- parámetros
- tiempo necesario para desarrollar y mantener el software para procesar archivos
- uno promedio: # registros usados / # registros