Un Sistema de Almacenamiento y Recuperación de Información Para Facilitar Tanto Procesos Puntuales Como en Orden Lexicográfico

F. AURTENECHEA *

RESUMEN

Se describe la estructura de hashing extendible para el almacenamiento y recuperación de información en archivos dinámicos. Se presenta una proposición de un esquema de almacenamiento mediante hashing extendible para permitir almacenar y procesar identificadores en orden lexicográfico, sin las restricciones de tiempo incurridas con hashing tradicional.

La factibilidad de este sistema se analiza basicamente con respecto a las necesidades de almacenamiento que la estructura requiere. El problema fundamental consiste en que, en general, no es posible encontrar una función de hashing que distribuya las claves en orden lexicográfico y al mismo tiempo distribuya el espacio de almacenamiento en forma uniforme. Se discute una solución a este problema, que se basa fundamentalmente en particionar el espacio de almacenamiento en bloques de tamaño variable.

ABSTRACT

The extendible hashing structure for information retrieval of dynamic files is described. The paper presents a proposal of a system for structuring data using extendible hashing, to allow keys to be stored and processed in lexicographic order, without the restrictions of traditional hashing.

This system is analyzed, basically, according to the requirements of primary and secondary storage needed for the structure. Tha basic issue is that, in general, it is difficult to find lexicographic order-preserving hash functions that distribute the key space uniformly over an address space. A solution to this problem is presented, which is based in partitioning the address space into variable-length blocks

* Ingeniero Civil Industriel (PUC Chile 1977); Master of Philosophy (M. Phil) University of Edinburgh (1982); Redes de Comunicación de Datos, Ingeniería de Software, Bases de Datos; Departamento de Ciencia de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, casilla 6177, Santiago, Chile.

I Introducción

Los sistemas destinados a almacenar grandes volúmenes de información se pueden evaluar, básicamente, de acuerdo a tres criterios fundamentales: i) el tiempo promedio de respuesta a una determinada consulta, ii) las necesidades de almacenamiento que requieren, y iii) la facilidad de adaptar su estructura en respuesta a demandas externas de inserciones y eliminaciones de registros.

Una estructura muy utilizada como método de almacenamiento y recuperación de información es la estructura de hashing. Cuando se requiere de accesos puntuales a la información, esta estructura es una de las más eficientes con respecto al primero de los criterios anteriormente mencionados (tiempo de respuesta). Sin embargo, esta eficiencia se logra a expensas de sobredimensionar el archivo o tabla de almacenamiento (tabla de hashing). Además, su estructura es básicamente estática, ya que una vez definida la dimensión de la tabla de hashing, sólo se pueden alterar valores de ésta, pero la tabla misma no puede ser modificada. Por lo tanto, si la dimensión de la tabla se subestima con respecto a los requerimientos reales de almacenamiento (por ejemplo, se logra ocupar más del 90% de la tabla), se debe proceder a reconfigurar completamente el archivo que contiene los datos.

En la búsqueda de soluciones para este problema, se han realizado investigaciones en nuevas estructuras de tipo hashing, denominadas hashing extendible o hashing dinámico (Fagin et al,1977 y Larson,1978). Este esquema de hashing, además de ser eficiente en términos de tiempo de respuesta ante consulta de sus elementos (máximo dos accesos al disco), es una estructura dinámica. Esto es, facilita el almacenamiento de la información de acuerdo a las reales necesidades, al permitir que las tablas de hashing puedan expandirse o contraerse dinámicamente.

Hashing extendible es una estructura orientada al almacenamiento de datos en dispositivos de acceso secundario (por ejemplo, discos magnéticos), donde la información se almacena mediante un esquema de páginas de tamaño fijo. El sistema permite mantener relativamente uniforme el factor de ocupación de las páginas, mediante esquemas de reestructuración que involucran un mínimo de costo adicional.

Sin embargo, con este método de almacenamiento, al igual que con el método de hashing tradicional, normalmente no es posible acotar en tiempo real la realización de procesos secuenciales, con respecto al orden lexicográfico de la clave de acceso a la información. En otras palabras, dada una clave cualquiera K_i, el costo de encontrar la clave que precede o sucede a K_i (K_{i-1}, K_{i+1}) en secuencia natural es, en general, difícil de soportar (Knuth,1974, Severance,1976).

La estructura de hashing extendible tiene dos propósitos fundamentales: (i) facilitar el almacenamiento de archivos dinámicos permitiendo que las tablas de hashing puedan expandirse o contraerse dinámicamente, y (ii) incorporar un esquema de páginas de almacenamiento, manteniendo uniformidad en la cantidad de datos existente en cada una.

La estructura incorpora dos niveles de almacenamiento. El primer nivel, es un directorio que contiene índices al segundo nivel, siendo este último una estructura de páginas u hojas. El directorio consiste de una tabla de acceso directo, cuyo espacio de direccionamiento D, se obtiene transformando el espacio de claves K mediante una función de hashing h(K) (Carter y Wegman, 1977). Cada entrada en el directorio representa un puntero que direcciona a una página de información, la que agrupa a un cierto número de registros con características similares.

(h(K) ---> D =====> Paginas)

La función de hashing establece una transformación del espacio de claves K en un espacio D de direcciones de gran tamaño. El espacio D se separa en una partición P de m bloques, en que cada bloque tiene una página asignada; el directorio establece la correspondencia entre bloques y páginas. La partición P define m+1 límites (a_0, a_1,..., a_m), en que la página P_i contiene todas las claves K tal que, $a_{i-1} \leq h(K) < a_i$.

Si se llena la página Pi, se puede cambiar la partición, por ejemplo moviendo el límite ai en uno y reorganizando sólo aquellas claves afectadas por esta reestructuración local. Luego, la idea de hacer extendible la tabla de hashing consiste en variar la partición P en un espacio D muy grande, sin variar la función de hashing.

El esquema anteriormente descrito se basa fundamentalmente en el Sistema Buddy para el manejo de información (Knuth 1974).

En este sistema se escoge D = (0,1,...,2^n-1), para algún n fijo. Entonces a_0=0 <a_1 < ... <a_m=2^n-1 son los límites de una partición de un sistema buddy, si y solo si todo intervalo (a_{i-1},a_i) puede ser obtenido por sucesivas divisiones de intervalos en D.

A continuación se describe un esquema específico de hashing extendible (ver Fagin et al,1979), y en el cual se basa nuestro análisis. Se escoge una función de hashing h fija con la cual se obtiene, por cada clave K, una pseudo-clave K'=h(K). El espacio de direccionamiento D, del directorio, es igual a 2^{n-1}, donde n es el tamaño máximo de la pseudo clave, en bits. Así por ejemplo, con 32 bits para definir el espacio D, se pueden direccionar aproximadamente 10^{10} páginas.

Se define como profundidad p del directorio, a los primeros p bits de la pseudo clave K', los que se usan para determinar la página. El directorio tiene 2^p punteros, no necesariamente distintos. El primero indica a la página que almacena todas aquellas claves cuya pseudo-clave comienza con p ceros consecutivos. El segundo puntero identifica a todas las claves cuya pseudo clave es 0...01 en sus primeros p bits; el tercero identifica a aquellas claves cuya pseudo clave es 0...010 en sus primeros p bits, y así sucesivamente en orden lexicográfico. Finalmente, el último puntero identifica a todas las claves cuya pseudo clave comienza con p unos consecutivos. La Fig. 1 muestra un posible esquema de almacenamiento para p = 3.

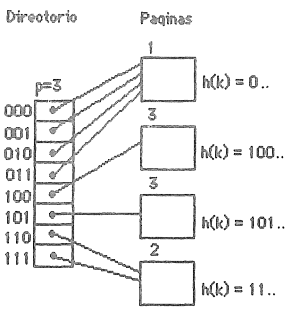
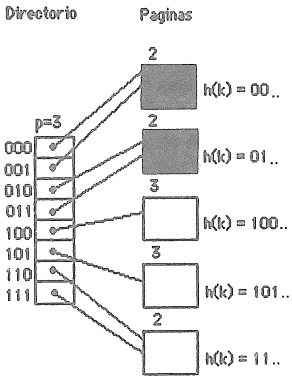


Fig. 1 Un Esquema de Hashing Extendible

Siguiendo el puntero 111 de la Fig. 1, se llega a una página cuya profundidad local (p') es igual a 2. Esto significa que esta página contiene no sólo aquellas claves cuyos primeros 3 bits son 111, sino también aquellas que comienzan con 11. Luego, el puntero 110 también señala a esta página.

Si se llena una página (o por ejemplo, se sobrepasa el 90% de ocupación) en la cual p' < p, ésta se separa en dos páginas, aumentando cada una su profundidad local en 1. Considerando la Fig. 1, tal caso ocurriría si se llena la página cuya pseudo clave comienza con 0.., (su p' es igual a 1). Después de producirse la reconfiguración, como se muestra en la Fig. 2, una página contiene a todas las claves cuya pseudo-clave comienza con 00, y la otra a todas las claves cuya pseudo-clave comienza con 01. Ambas páginas con profundidad local igual a 2.



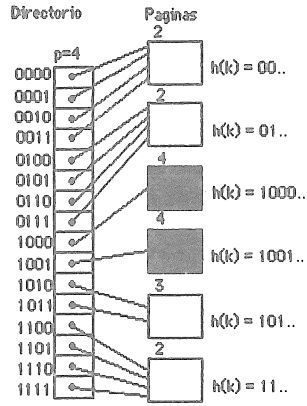


Fig 2 Rebalse de pagina sin duplicar directorio Fig 3. Rebalse de pagina duplicando directorio

Si se llena una página i cuya profundidad local es igual a la profundidad del directorio ($p=p'_1$), entonces éste debe aumentar su profundidad local en 1, lo que significa duplicar su tamaño. Por ejemplo, si se llena la página indicada por el puntero 100, de la Fig. 2, además de separar su información en dos páginas, se debe proceder a duplicar el directorio (ver Fig. 3).

Nótese que a lo más hay dos accesos al disco para encontrar una determinada clave. Uno para localizar la página específica en el directorio, y otro para localizar la página donde se encuentra la clave.

IV Secuencialidad

La distribución del espacio de claves es generalmente no uniforme. Escogiendo alguna fución de hashing adecuada (Carter y Wegman, 1977), cualquier distribución no uniforme de claves, puede ser convertida en otra cuya distribución es aproximadamente uniforme. Esto es muy difícil de lograr si se desea que el esquema de almacenamiento de las claves sea tal que acepte el efectuar procesos secuenciales con respecto a su orden lexicográfico. En otras palabras, dada una clave cualquiera K_i, el costo de encontrar la clave que precede o sucede a K_i (K_{i-1}, K_{i+1}) en la secuencia natural, es difícil de soportar.

En este documento presentamos una proposición de un esquema de almacenamiento de las claves en orden lexicográfico, mediante hashing extendible. Para ello, se debe escoger una simple función de hashing h tal que si $h(K_1) < h(K_2)$, entonces $K_1 < K_2$ Una forma trivial de lograr esto es usando simplemente el valor de la clave, directamente como función de hashing. Esto sería equivalente un sistema de acceso directo por la clave a almacenar, lo que es difícil de soportar dado que el

tamaño del directorio tendría que ser igual al tamaño del espacio de claves. Además, la agrupación de la información en páginas carecería de sentido.

Mediante la agrupación de claves en páginas, es posible acotar el tamaño del directorio, usando como función de hashing los primeros p bits de la clave. el valor de p dependerá de factores tales como: el tamaño de página a usar, y la distribución del espacio de claves. Las cotas inferior y superior de este espacio no son, en general, posibles de determinar a priori. Sin embargo, los valores máximo y mínimo de las claves, en cualquier instante son datos que si se conocen. Dado que hashing extendible permite modificar dinámicamente el tamaño de directorio, éste se podría ajustar al espacio de direccionamiento de las claves, en cualquier instante.

Para atacar el problema de la no uniformidad del espacio de claves, habría que particionar el espacio de direccionamiento en el directorio, en bloques de tamaño variable. Por ejemplo, en regiones donde se agrupen muchas claves, habría que escoger tamaños de página mayor que en regiones donde haya escasa agrupación de claves. El factor tamaño de página depende básicamente del hardware del computador a usar: tamaño de la pista, número de sectores por pista, etc.

A continuación se propone un esquema de almacenamiento usando hashing extendible, cuya factibilidad se analiza en función de los factores anteriormente señalados. Básicamente, el directorio es administrado en base a los indicadores: p, p'(i) y pm. Como se describió anteriormente, p es la profundidad del directorio en un instante cualquiera, y p'(i) es la profundidad local de la pagina i. El indicador pm representa el número de bits correspondiente a la mayor clave posible de almacenar en la estructura, en un instante cualquiera. Este valor es función de la mayor clave que ha sido almacenada (Km) hasta ese instante, en la forma:

$$pm = \langle \log_2(km) \rangle + 1 \qquad (1)$$

donde, $\langle x \rangle$ es la parte entera de x.

Luego, la mayor clave posible de almacenar en la estructrura es $Kmax = 2^{pm} - 1$. Entonces, para una clave cualquiera k, se usa la función de hashing

$$h(k) = k >> (pm - p) \qquad (2)$$

donde, x >> y es el valor que resulta despues de ajustar x, en y bits hacia la derecha.

Con esta transformación, se cumple con la condición de secuencialidad en el almacenamiento de las claves. Además, para una página i, el valor $2^{pm-p'(i)}$ corresponde al potencial de claves a almacenar en dicha página. Considerando la función (2), con pm=10 y p = 3, la Fig. 4 muestra la distribución de almacenamiento en las páginas, para el mismo esquema de la Fig. 2.

En este caso la estructura, en ese instante, tiene capacidad para almacenar un máximo de 1024 claves (desde la clave 0 hasta la clave 1023). El mínimo número de claves en una página i, se produce cuando p'(i) = p. Esto ocurre con las páginas apuntadas por los punteros 100 y 101; cada una de ellas puede almacenar $2^{(10-3)} = 128$ claves.

Qué pasa si se desea almacenar una clave k que es mayor que la mayor clave (Kmax) posible de almacenar en la estructura? En este caso, se debe duplicar la profundidad del directorio tantas veces como sea la diferencia ($\langle \log_2 k \rangle + 1$) - pm. Los punteros hasta antes de la duplicación, permanecen inalterados; solo se crean nuevos punteros a partir del último (los que señalan a la nueva página). Además, el valor de pm debe incrementarse en la misma cantidad en que fue incrementado p.

Considerando la Fig. 4, supongamos que se desea insertar la clave 3540. En este caso, 3540 > 1023, y como $\langle \log_2 3540 \rangle + 1 - 10$ es igual a 1, la profundidad del directorio y el número de bits de la clave mayor deben aumentarse en 1. La Fig. 5 muestra este nuevo esquema. Nótese que ahora la profundidad local de cada página, disminuyó en uno con respecto a la profundidad del directorio, ya que sólo se agregaron nuevos punteros a la nueva página (p' para ésta página vale cero). Luego, debe llevarse un contador por cada duplicación del directorio en estas circunstancias, para corregir el valor de p'.

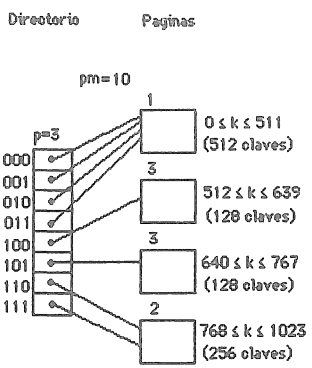


Fig 4 Almacenamiento Secuencial

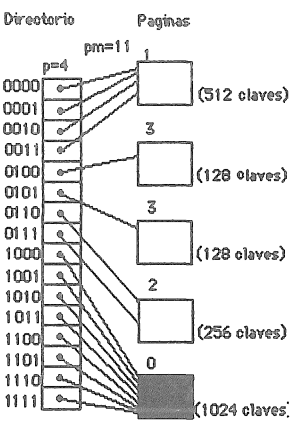


Fig 5 Duplicacion del directorio (K > Kmax)

En la práctica, la duplicación del directorio, del esquema de la Fig 5, podría posponerse, si la página anterior (en el orden lexicográfico de claves) tuviera suficiente capacidad (por ejemplo, su factor de ocupación es menor que 60%). Así, la nueva clave se puede almacenar en dicha página, junto con toda nueva clave a insertar que sea mayor que Kmax (esto debe considerarse en la función de hashing). Si el factor de ocupación de esta página aumenta demasiado, entonces se debe proceder a duplicar el directorio como se indicó anteriormente, distribuyendo las claves mayores que Kmax en una nueva página. De este modo, se evita que valores ocacionales del espacio de claves hagan aumentar el directorio en forma desmedida.

Tal como se mencionó anteriormente, el directorio también se duplica cuando se llena una página cuya profundidad local es igual a p (la profundidad del directorio). En este caso, p se incrementa en 1 pero pm permanece inalterado. Mientras menor sea la diferencia entre pm y p, menor será la cantidad de claves posibles de almacenar en

cada página. En el caso extremo en que pm sea igual a p, el espacio de claves se hace igual al espacio de direccionamiento, y toda página i en que p'(i) = p, limitaría su potencial de claves a 1. Esta situación se produce cuando el espacio de claves tiende a localizarse en torno a un cierto número reducido de páginas. En este caso, además de la no homogeneidad en el factor de ocupación de las páginas, se puede producir un crecimiento desmedido del directorio (por sucesivos rebalses en las páginas). Para evitar este fenómeno, habría que escoger un valor límite (DI) para la diferencia (pm - p). Entonces, al producirse un rebalse en una página i, y ademas se cumple que, p'(i) = p, y (pm-p) = DI, hay que procurar un espacio de almacenamiento para todo nuevo elemento a insertar en dicha página, sin duplicar el directorio.

Una solución podría consistir en ligar otra página a continuación de la página i en cuestión. Esta solución puede conducir a un deterioro importante en el tiempo de acceso promedio a los elementos, si la no uniformidad del espacio de los identificadores es excesiva.

La solución que se propone para evitar el encadenamiento de páginas, consiste en sobredimensionar ligeramente al directorio de modo que éste pueda diferenciar entre páginas de tamaños diferentes. De este modo, al producirse un rebalse en la página i, en las condiciones descritas anteriormente, se traspasa la información de dicha página a una de mayor tamaño, indicando su localización en el directorio (por ejemplo con los primeros d bits de cada puntero, siendo 2^d mayor o igual al número de tamaños distintos de página). Así, se mantiene el tiempo máximo de respuesta al equivalente de dos accesos al disco.

El problema radica en qué y cuántos tamaños de páginas escoger. Con respecto al tamaño de página, hay que considerar que éste es, en general, dependiente del sistema operativo y la configuración de los dispositivos de disco utilizados (tamaño del sector, pista etc.).

Como se mencionó anteriormente, $2^{pm-p'(j)}$ representa el potencial de ítems a almacenar en la página j. Entonces, cuando se cumpla que, DI=(pm-p'(j)) y (p'(j)=p), la información de la página j irá emigrando (por posibles rebalses) a páginas de mayor tamaño hasta un cierto límite superior de tamaño de página (T_{max}). Luego, para permitir la ocupación de esta página y no desperdiciar espacio de almacenamiento, el valor de DI deberá escogerse como el mínimo valor tal que:

$$2^{DI} \geq \langle T_{max} / Ti \rangle \qquad (3)$$

donde Ti es el tamaño de cada ítem a almacenar (asumiendo que es fijo)

Considerando que normalmente el acceso físico al disco es por sector, el tamaño de página inicial (página más pequeña), podría escogerse como el tamaño de un sector (o un submúltiplo de éste). Así, dado que se espera una no uniformidad en el espacio de direccionamiento, se evita un sobredimensionamiento de páginas en regiones con escasa agrupación de ítems. En caso de rebalse en una de estas páginas, su información se puede traspasar a otra página cuyo tamaño podrá ser un múltiplo del tamaño de un sector. Así sucesivamente hasta alcanzar el valor T_{max}.

Para optimizar el tiempo de acceso a memoria secundaria, el almacenamiento de la información en el disco debería ser tal que, cada página, se agrupe en sectores contiguos ("sector interleaving") en una misma pista (Pechura et al,1983). De este modo, mientras se procesa un sector, el siguiente podría ser accesado sin necesidad de esperar una rotación completa del disco. Entonces, podría considerarse como valor aceptable para T_{max}, el valor más cercano al tamaño de la pista (Severance y Duhne,1976).

En cuanto al número de páginas (Np) a escoger, se propone que éste sea igual a

$$Np = \langle \log_2 SPP \rangle \qquad (4)$$

donde SPP es el número de sectores por pista.

Entonces, se escoge como tamaño inicial de página 1 sector del disco, luego 2 sectores, luego 4, etc. Y así sucesivamente hasta Np tamaños. El aumento exponencial del tamaño de página evita la emigración excesiva de información a distintas páginas, en regiones donde se agrupa una gran cantidad de claves. El tamaño inicial de página podría, escogerse algo menor que el tamaño de un sector, asegurando así que aún en zonas de escasa agrupación de claves, se mantenga el factor de ocupación de las páginas por sobre el 50%.

Manteniendo un esquema óptimo de "interleaving" de sectores, se asegura que el tiempo de acceso a una página sea a lo más el equivalente a una rotación del disco. Es razonable esperar que el costo de accesar el directorio sea poco significativo, ya que normalmente estará en memoria principal, aún en aplicaciones que involucren archivos de tamaño relativamente grande (por ejemplo más de 1 millón de registros). Si el directorio no cabe completamente en memoria principal, se puede paginar en bloques de tamaño fijo, manteniendo en memoria principal las páginas de mayor uso.

V Conclusiones.

Se ha propuesto un sistema de almacenamiento y recuperación de información, basado en el esquema de hashing extendible. Nuestro sistema permite almacenar y procesar la información en orden lexicográfico, sin las restricciones de tiempo impuestas por los sistemas de hashing tradicional.

El sistema se basa fundamentalmente en particionar el espacio de almacenamiento en páginas de tamaño variable. La factibilidad del sistema se analiza en base a los requerimientos de almacenamiento en memoria secundaria, y la facilidad para mantener cada página en sectores contiguos en memoria secundaria. Mediante el esquema de particiones se permite, mantener un alto factor de ocupación de las páginas, y acotar el tiempo de acceso a una clave a un máximo de dos accesos al disco.

VI Bibliografía

1. Bayer, R. y McCreigth, E. (1972) Organization and maintenance of large ordered indexes". Acta Informatica Vol 1, No 3, 173-189.

2. Carter, J., y Wegman, M. (1979) Universal Classes of hash functions. Journal of Comput. System Science. Vol 18, No 2, 143-154.

3. Fagin R., Nievergelt, J., Pippenger, N., Strong, H. (1979) Extendible Hashing a fast access method for dynamic files. ACM Transactions on Database Systems, Vol 4, NO 3, 315-344.

4. Knuth, D. E. (1974) The Art of Computer Programming. Vol 3, Addison-Wesley, Reading, Mass.

5. Larson, P. (1978) Dynamic Hashing. BIT 18, 184-201.

6. Litwin, W. (1979) Linear virtual hashing: a new tool for files and tables manipulation. Institut National de Recherche en Informatique et en Automatique, Le Chesnay.

7. Litwin, W. (1980) Trie Hashing. Institut Natinal de Recherche en Informatique et en Automatique, Le Chesnay.

8. Pechura, M. A., Y Schoeffler, J. D. (1983) Estimating file access time of floppy disks. Communication of the ACM, Vol 26 No 10, 754-763.

9. Severance, D., y Duhne, R. (1976) A practitioner's guide to addressing algorithms. Communication of the ACM, Vol 19 No 6, 314-326.

10. Severance, D., y Carlis, J. (1977) A practical approach to selecting record access paths. Computing Surveys, Vol 9, No 4, 259-272.