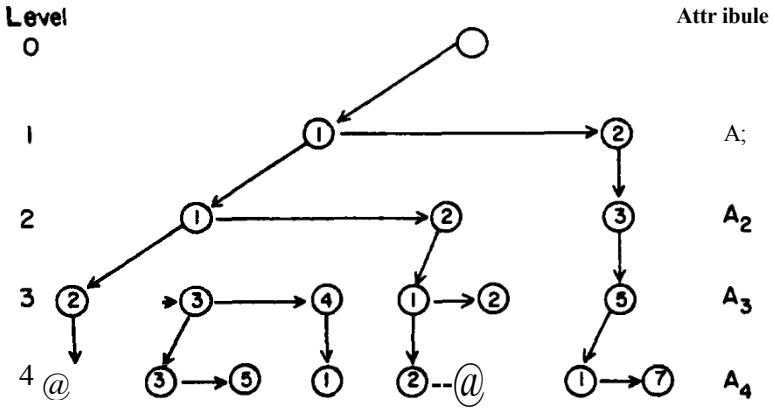


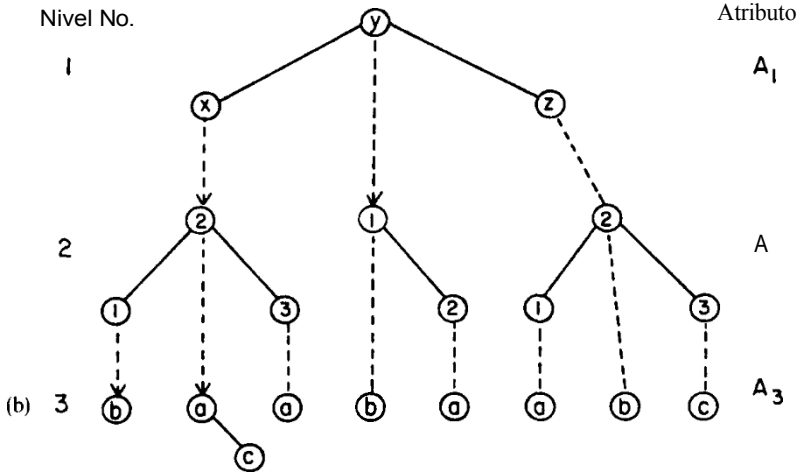
FIG. 7. Árbol doblemente encadenado para los datos de la Fig. 6a.

El complejo BST para un fichero de es similar al MAT. En el complejo BST, cada conjunto filial se mantiene como un árbol binario de altura equilibrada o un árbol AVL (en un árbol AVL, la diferencia entre las alturas de los subárboles izquierdo y derecho de cada nodo es como máximo uno). Esto garantiza un tiempo de búsqueda logarítmico para una consulta de coincidencia completa. La Fig. 8b muestra el complejo BST para el fichero de datos de la Fig. 8a. Las hojas llevan punteros a los números de página. Por lo tanto, si el número de valores clave distintos en el nivel i es r(i), el tiempo necesario para que el complejo BST responda a una consulta de coincidencia exacta es como máximo Z'-
log(r(i))ñ k log *N.* A pesar de su buen rendimiento para la coincidencia completa, la BST-complex no admite eficientemente las consultas de coincidencia parcial y de rango. El árbol B multidimensional **(MDBT**) de Ouskal y Scheuermann (1981) garantiza una búsqueda eficiente para diversas consultas asociativas en entornos de bases de datos. La idea central es el uso de árboles B para mantener los conjuntos filiales en una estructura similar a MAT. Esta estructura, denominada MDBT, se propone como una alternativa viable para un entorno con un grado de complejidad de consulta bastante elevado. Las estrategias para mantener el **MDBT** también se discuten en Ouksel y Scheuermann (198 l). Las estimaciones analíticas para la **MDBT** aleatoria vienen dadas por 0(log *N), O(N* log *N'JN')* y 0(log *N)* como complejidades para las consultas de coincidencia exacta en el peor de los casos, coincidencia parcial en el peor de los casos y rango medio restringido, respectivamente. *N* denota el producto de los tamaños medios de conjuntos filiales de niveles no especificados para una coincidencia parcial. Cabe señalar que se supone que la MDBT tiene una forma "media". De nuevo, la consulta de rango restringido es una forma simplificada de una consulta de rango generalizado. Se supone que cada atributo no especifica más de dos valores de atributo. Sin embargo, una consulta de rango generalizado tendrá sub-
complejidad sustancialmente mayor en el **MDBT** que *O(log N).*

Otra estructura de índice arborescente que se basa en los mismos conceptos que la MDBT es el árbol B multidimensional (árbol kB) propuesto por Gueting y

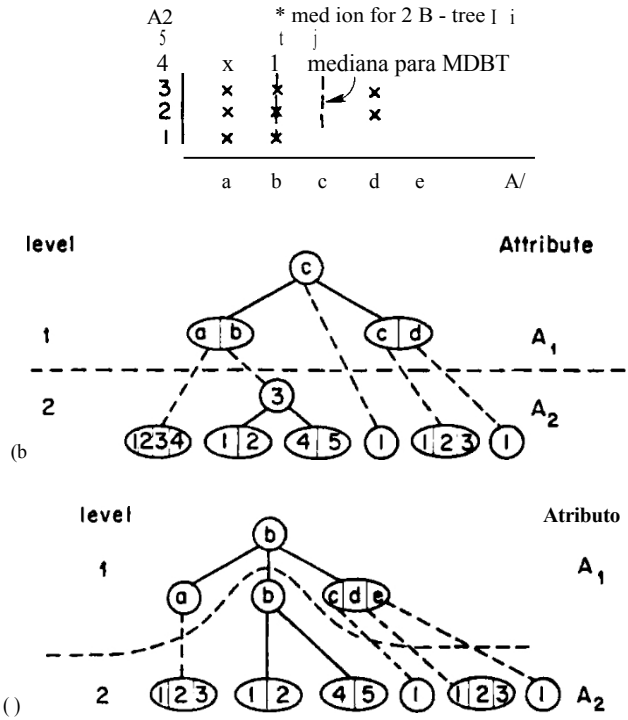
	A/	A_2	Ag	Paęe	No.
	x	i	b	i	
	y	i	b	2	
	y	2	a	4	
	x	2	c	5	
	x	3	a	6	
	z	2	b	7	
	z	3	c	9	

(a)



Fic. 8. (a) lnpul data fife; (b) BST-complex for data in (a).

Kriegel (1980, 1981). Bajo el supuesto de una distribución uniforme e independiente de los atributos, el MDBT tendrá una altura máxima de 1og *N + k.* Pero en los datos que no satisfacen estas propiedades, la profundidad de búsqueda podría ser tanto como *O(k* log *N).* El novedoso método de "equilibrio entre todos los componentes" del árbol kB garantiza que la profundidad sea 0(logo+ t *N*+ *k j,* donde m es el orden del árbol kB. Por lo tanto, la consulta de coincidencia completa, la inserción y las supresiones restringidas a una ruta tienen una complejidad de tiempo logarítmica. Presentaremos un ejemplo de Kriegel (1984) para ilustrar la diferencia entre el MDBT y el árbol kB. La figura 9a muestra registros bidimensionales como puntos en un espacio bidimensional. La figura 9b muestra el correspondiente. En el



Fret. 9. (a) Puntos de datos In put; (b) MDBT para los datos de (a); (c) kB-tree para los datos de (a).

MDBT cada nivel corresponde a un atributo, y cada conjunto filial se mantiene como un árbol B en el mismo nivel. En el nivel 1, la mediana de a, b, c, d, *e,* es c, ya que el equilibrio se basa únicamente en el atributo l. La figura 9c muestra el árbol GB. En el nivel 1, se selecciona *b* como mediana, ya que el equilibrio se basa en todos los registros. Esta noción de equilibrio garantiza la profundidad 0(log *N*+ *k)* para el árbol kB. Aunque el concepto de equilibrio sobre todos los componentes puede encontrarse en Lee y Wong (1977), la principal contribución de los árboles kB es su mantenibilidad en entornos dinámicos. Sin embargo, tanto el MDBT como el árbol kB tienen un defecto: ninguno de los dos admite el procesamiento secuencial en cada nivel. Esta propiedad es muy importante para responder a consultas de coincidencia parcial y de rango. Existen otros refinamientos de éstos sobre el árbol #B, *el kB' -ttee,* y *(k*+ l)B'-tree, que son más eficientes en ciertos aspectos. Para más detalles, véase Kriegel (1984).

La principal diferencia entre las estructuras de datos mencionadas reside en la forma en que se organizan los conjuntos filiales. En el MAT, los conjuntos filiales se mantienen como listas ordenadas que admiten la búsqueda binaria. En el main

Esta característica se mantiene de forma natural si los miembros de los conjuntos filiales se almacenan en ubicaciones consecutivas. Por consiguiente, buscar un miembro en un conjunto filial de tamaño s cuesta casi O(log(sj). Las implicaciones exactas de este aspecto se analizan en la sección siguiente.

6.3 Generación de directorios

La estructura de datos MAT se linealiza para formar un directorio. El directorio define el modo en que se organizan los distintos niveles y el modo en que se mantienen los conjuntos filiales. Las estructuras de datos como MAT, DCT, MDBT, kB-tree, *k* B" -tree, etc. pueden implementarse utilizando punteros. Para responder a una consulta es necesario recorrer los punteros. En general, no se hace ninguna suposición sobre la organización de los punteros. Este tipo de implementación mediante punteros es eficaz cuando la estructura de datos reside en la memoria principal. Pero cuando el directorio reside en la memoria secundaria, donde la transferencia de información se realiza en forma de páginas, la aleatoriedad de la organización de los punteros provoca la recuperación de muchas páginas innecesarias. Por ejemplo, consideremos el caso en el que los nodos de un conjunto filial residen en páginas separadas de la memoria secundaria. Esta situación puede darse cuando el conjunto filial se genera mediante inserciones dinámicas de registros una vez construida la estructura estática. En este caso, la búsqueda de una vaJue en un conjunto filial de tamaño s es probable que resulte en 0(log *N)* recuperaciones de páginas de memoria secundaria. En cambio, si un conjunto filial se mantiene en una sola página o en un número constante de páginas, la búsqueda implica 0(l) recuperaciones de páginas secundarias. Una vez cargada una página secundaria en la memoria principal, se puede la búsqueda del valor deseado en el conjunto filial. Esta situación se da en organizaciones de bases de datos físicas en las que los archivos residen físicamente en el almacenamiento secundario. Las otras aplicaciones pueden incluir gráficos, computacional, etc., donde el gran tamaño de los datos hace necesario el almacenamiento de la información en la memoria secundaria.

La principal fuerza impulsora de la organización basada en MAT el almacenamiento en memoria secundaria. Aunque la estructura abstracta MAT puede implementarse utilizando punteros, nosotros consideramos una organización en la que los nodos correspondientes a un conjunto filial se mantienen juntos. La idea principal es agrupar los nodos a los que es probable que se acceda al responder a una consulta. En una consulta de coincidencia completa, se busca un valor en los nodos de un conjunto filial. En una consulta de coincidencia parcial, o bien se busca un valor en el conjunto , o bien se recupera todo el conjunto filial. En una consulta de rango, se recuperan los nodos dentro del rango especificado. Para responder a las consultas anteriores, introducimos el concepto de linealización, que nos permite almacenar la estructura de datos MAT en forma de tabla. Existen dos tipos básicos de linealización propuestos por Subas y Kashyap (1975) y Kashyap *et al.* (1977): linealización en profundidad y linealización en amplitud. Cualquiera de ellas puede realizarse de descendente o ascendente. El método descendente es más

eficiente que el ascendente. Esto se debe a que, al responder a una consulta, el método descendente permite reducir el número de nodos que hay que buscar. En la organización depth-first, los nodos correspondientes a los campos de un registro se mantienen juntos, como se ilustra en la Fig. 10a. Los números al del nodo representan el número de entrada de los nodos en el directorio. Se trata de una técnica eficaz para las consultas de coincidencia completa, ya que en el mejor de los casos la respuesta puede producirse en un solo acceso al almacenamiento secundario (suponiendo que *k* nodos puedan almacenarse en una sola página). Sin embargo, el peor de los , se necesitan *k* accesos. Pero esto no admite de forma natural las consultas de coincidencia parcial y de rango, ya que los miembros de un conjunto filial no se mantienen juntos. En la linealización breadth-first, los nodos correspondientes a un conjunto filial se mantienen juntos, como se muestra en la Fig. 10b. Este método admite de forma natural las consultas de coincidencia parcial y de rango.

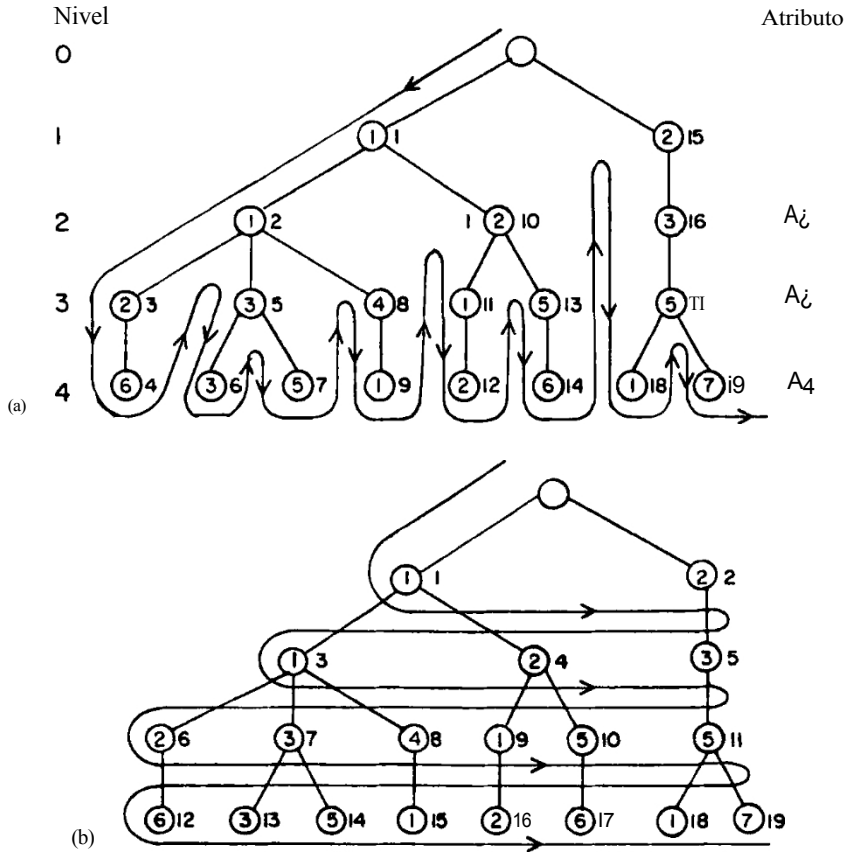


FIG. 10. (a) Dept h-primera linealización; (b) amplitud primera linealización.

En las figuras l la y llb se muestran los directorios correspondientes a las linealizaciones depth-first y breadth-first, donde cada elemento del directorio es del tipo

elemento del directorio -- **record**

node-num: I.. *M,'*

valor: *l..M;*

primogénito/hermano:

1..M,' último-hijo/primo:

1..M;

fin;

donde los campos correspondientes a un nodo T, en el nivel j, y numerado n se definen como sigue:

valor: valor del nodo *T;*

first-child: número de nodo del primer elemento
del conjunto de hijos del nodo *T;* para j = 1, 2, ... k

puntero de registro para j = *k*

last-child: número de nodo del último elemento
del conjunto de hijos del nodo *T,* para y= 1, 2, ... , *k*

: 0 para y= *k;*

brother: número de nodo del nodo contiguo a
nodo *T* en el orden para y= 1, 2, ... , *k;*

primo. número de nodo del primer nodo del
conjunto filial próximo al de
nodo *T;* para y= 1, 2, ... , *k;*

Los campos /primer-niño y *último-niño* se utilizan en la linealización breadth-first. Los campos *brother* y *cousin* se utilizan en la linealización dept h-first. Los algoritmos para generar un directorio depth-first pueden obtenerse de Kashyap *et* at. (1977), y Gopalakrishana y Veni Madhavan (1980). Damos un algoritmo para generar un directorio breadth-first.

El paso 2 de DIR-GENERATION calcula basest), / - 1, 2, ... , *(k*+ 1), tal que cualquier nodo de directorio en el nivel *i* se encuentra entre la base *bij* y la base bi + 11 1. Estos valores son necesarios para almacenar los campos value, first-child y last-child en el algoritmo DIR-GENERATION. Este algoritmo se describe a continuación.

El principal coste del algoritmo DIR-GENERATION se debe a la ordenación. Una vez ordenados los registros imput, se realizan dos pasadas sobre los registros ordenados, una

Nodo - ámbar	Vd ue	Hermano	**Primo**
2	1	i 0	16
4	6		6
6	3	7	9
		-	9
		-	1 1
		-	1 2
IO	2	-	1 6
2	2	-	1 4
i s	3	-	-
1 8	1	18	-

(a)

FIG. 11. (a) "depth-first"; (b) directorio "breadth-first".

para el algoritmo COM PUTE-BASE y una vez para las líneas 3 a 11 del algoritmo DIR-GENERATION. Si la entrada reside en la memoria principal, entonces la fase de ordenación tiene *O[kN log(kN))* comparaciones. El algoritmo COM PUTE-BASE *tiene O(kN)* comparaciones, o accesos a memoria. Esto se debe a que la línea 6 de COM PUTE-BASE y la línea 6 de CO M PUTE-BASE y la línea 6 de DIR- GENERATION implican como máximo k comparaciones para cada registro. De nuevo, la línea 8 de COM se ejecuta como mucho *k* veces por cada registro. Del mismo modo, podemos demostrar que líneas 8 y 9 de DIR-GENERATION pueden ejecutarse como máximo *k* veces por cada registro. Por tanto, la complejidad de

Número de nodo	Vo lue	Primer niño	Niño perdido
2	2	5	5
4	2	9	I O
6	2	12	t 2
8	4	i 0	1 5
IO	5	17	17
iy	6	4	
i 0	1	2	-
16	2	2	-
18	1	8	-

(b)

Traste. 11. *(Cont inueJ j*

DI R-GENERACIÓN es $O([kN\ j \log(kN) + HN) - O((kN) \log(kN))$. Si cada comparación de registros se considera de tiempo unitario, entonces la complejidad es $0[N \log N\ j$.

Si los registros de entrada residen en el almacenamiento secundario, entonces el coste se debe a los accesos a la página durante la clasificación. Las otras partes del algoritmo realizan dos pasadas secuenciales sobre los datos ordenados, y se realizan *Ohm)* accesos a páginas durante estas pasadas, donde m es el número de páginas en las que se almacenan los datos. Por lo tanto, utilizando técnicas estándar de ordenación externa, la generación de directorios puede realizarse en 0(m log m) accesos a páginas como en Horowitz y Sahni (1976).

algoritmo DI R-GENERATION;
comience
1. ordenar el fichero de datos introducido en función de *los k* atributos
2. COM PUTE-BASE;
3. leer el primer disco;
4. repita
5. leer el siguiente disco;
6. compararlo con el registro anterior para obtener L, el nivel más alto en el que los valores de los atributos son diferentes;
7. para j := L a *k âo*
8. asignar campo de valor;
9. para j:= (L+ 1) a *k* hacer
10. asignar los campos first-child y last-child;
11. **hasta que** se procesen todos los registros;

fin;

algoritmo COM PUTE-BASE;
comience
1. para i:= 1 a *[k*+ l) hacer
2. base *ij* := i;
3. leer el primer disco;
4. **repita**
5. leer el siguiente disco;
6. compararlo con el registro anterior para obtener L, el nivel más alto en el que los valores de los atributos son diferentes;
7. para j:= L hasta (k+ 1) hacer
8. base y)= base[jJ+ 1;
9. **hasta que** se procesen todos los registros;
10. para j:- 2 hasta *(k*+ 1) hacer
11. base b y)= base y)+ base / - 1 j;

fin;

6.4 Algoritmos de búsqueda

En esta sección, analizamos los algoritmos de búsqueda basados en MAT para consultas de coincidencia completa, coincidencia parcial y rango. El concepto de "agrupación" de datos se utiliza para mejorar el rendimiento medio de varias búsquedas.

algoritmos. Esto se consigue manteniendo juntos los nodos a los que es probable que se acceda al responder a una consulta determinada. El análisis del rendimiento se lleva a cabo en los dos casos siguientes:

(i) reside en la memoria principal,
(ii) reside en el almacenamiento secundario.

La linealización breadth-first asegura que los miembros de un conjunto filial son consecutivos en memoria principal y se puede realizar una búsqueda binaria sobre ellos. Cuando el directorio reside en memoria secundaria, esta agrupación ayuda a reducir el número de páginas a las que se accede. Suponemos que cada conjunto filial no ocupa más de un número constante de páginas en la memoria secundaria. También suponemos que un conjunto filial puede cargarse completamente en la memoria principal cuando se está respondiendo a las consultas. En esta sección demostramos que el **MAT** funciona casi tan bien como otras estructuras, como el árbol kB, MDBT, etc., cuando el directorio reside en la memoria principal. Nuestro principal resultado es que el MAT es una mejor opción cuando el directorio reside en la memoria secundaria.

Definimos una consulta *Q,* como sigue:

$$Q = \bigcap_{i=1}^{k} q_i$$

donde q, es el calificador del atributo *A;*. Para una consulta de coincidencia completa, q; especifica un valor para *A;, i* -- 1, 2, ... , *k.* Para una consulta de rango, q; especifica un rango [/;, Ii;d para el atributo *A;, i* -- *I,* 2, ... *k*, donde I; y /i; indican los límites inferior y superior del intervalo, respectivamente. Para una coincidencia parcial, cada nivel i. *i* = 1, 2, ... *k,* puede estar especificado o no. Para un nivel i especificado, *q;* especifica un valor para *A,*. Para niveles no especificados, q, no especifica ninguna restricción sobre *A;*. Una coincidencia completa puede visualizarse como una consulta de rango que especifica el rango de un único valor para cada atributo. Del mismo modo, una consulta de coincidencia parcial puede visualizarse como una consulta de rango que especifica un rango de tamaño uno para los niveles especificados, y un rango del dominio del atributo para los niveles no especificados. Normalmente, una coincidencia completa y una coincidencia parcial son más fáciles de responder que una consulta de rango. También analizamos una consulta de rango restringido, que no especifica más de n valores para cada atributo. La respuesta a una consulta puede visualizarse utilizando nociones geométricas: el conjunto de registros de *k* atributos describe un espacio k-dimensional. Una consulta de rango describe un hiperrectángulo rectilíneo en *k* dimensiones. Responder a una consulta de rango corresponde a la recuperación de los puntos contenidos en el hiperrectángulo especificado por la consulta de rango. En la organización basada en MAT, al responder a una coincidencia completa, a una coincidencia parcial o a una consulta de rango, el procesamiento de cada atributo reduce en uno la dimensionalidad del espacio que se va a buscar.

La noción básica de una búsqueda basada en MAT es la de "descenso jerárquico", en la que la consulta se procesa nivel por nivel empezando por el nodo raíz. En cada nivel /, j = 1, 2, ... , *k,* se agregan los nodos que satisfacen la consulta "parcial" {- , q,. Estos nodos se denominan *nodos cualificados.* Los nodos cualificados del nivel *k* contienen punteros a los registros físicos que satisfacen la . Limitamos nuestro análisis a la linealización breadth-first. Encontrará más información al respecto en Rao *et* at. (1984). La discusión sobre la linealización en profundidad puede encontrarse en Gopalakrishana y Veni Madhavan (1980).

E n el algoritmo MAT-SEARCH, la línea 10 corresponde a la llamada inicial. Cuando el directorio reside en almacenamiento secundario, la ejecución de las líneas 2 y es muy crítica para decidir el número de accesos a la página. El conjunto qnodos de MAT-SEARCH se ordena y se accede a él en orden. Esto resulta en acceder a los conjuntos filiales consecutivos de izquierda a derecha en la estructura MAT. Este tipo de acceso es eficiente cuando los conjuntos son de tamaños más pequeños, de modo que varios de ellos pueden alojarse en una sola página. Este escenario se da al responder a una consulta de rango. De hecho, utilizando la ordenación breadth-first y un conjunto qnodes ordenado, los accesos secuenciales del también pueden soportarse de forma eficiente.

```
algoritmo MAT-SEARCH(nivel, qnodos);
comience
1.    tempset:= Q;
2.    para cada n e qnodos hacer
3.       añadir a tempset todos los hijos de n que
         Satisface q",e,;
4.    si nivel< k
5.    entonces MAT-SEARCH(nivel+ 1, tempset)
6.    si no
      comience
7.       para cada n e tempset do
8.    recuperar puntero de registro;
9.    fin;
fin;
10.   MAT-SEARCH(1, (raíz));
```

a) Consulta de correspondencia completa. Para una consulta de correspondencia completa, en cualquier nivel *j,* sólo hay un nodo que satisface la consulta parcial *_ , q, ya que *- , *q,* especifica una combinación única de valores de atributo en el MAT "parcial" reducido a los primeros y niveles. Por lo tanto, la línea 3 del algoritmo MAT-SEARCH se ejecuta sólo una vez para cada nivel. En cada ejecución de la línea , se realiza una búsqueda binaria en el conjunto filial para el valor de atributo especificado. Por lo tanto, el peor

la complejidad del caso de la consulta de coincidencia completa viene

dada por 0(log(r,)+ log(r,)+ + log (r,)) = *Roble* log *N).*

Si existe "uniformidad" en el perfil de MAT, de forma que el tamaño medio del conjunto hlial de nivel y es *sj*, el coste medio de la consulta de coincidencia completa pasa a ser

O(log *(s,*) log (- 1+ - -+- log *(s*)) -- O(log *N j.*

Esta estimación es adecuada para datos aleatorios que satisfacen determinadas distribuciones. La complejidad de 0(log *N)* para una coincidencia completa de datos multidimensionales puede alcanzarse fácilmente tratándolo como un problema unidimensional equivalente. La clave correspondiente a cualquier registro en el problema equivalente se obtiene concatenando los valores de los atributos correspondientes. Se puede construir fácilmente un árbol binario equilibrado que garantice una complejidad 0(log *N j* para buscar cualquier clave correspondiente a la consulta de coincidencia completa. Sin embargo, esta estrategia no es adecuada para tratar las consultas de coincidencia parcial y de rango.

Si el directorio se mantiene en almacenamiento secundario, en cada nivel no recuperamos más de c páginas correspondientes al conjunto filial del nodo calificado en el nivel anterior. Por lo tanto, el número de accesos a páginas para una coincidencia completa es *O(' k)* -- *O(k j.* En la organización breadth-first siempre se recuperan ck páginas para cualquier consulta de coincidencia completa. Para la linealización en profundidad, en el mejor de los casos, un acceso a una página será suficiente para responder a una consulta de coincidencia completa. En este caso, todos los nodos cualificados correspondientes a una consulta de coincidencia completa se encuentran en una única página.

página. Pero en el peor de los casos pueden recuperarse 0(st+ s2 * + s) páginas. En este caso, los nodos calificados son los últimos de sus conjuntos filiales. Utilizando las caracterizaciones medias del MAT, el número de accesos a páginas en el peor de los casos es

O(kN '!k) para una consulta de coincidencia completa. Es observar que se necesitarán tantos accesos a páginas cuando el MAT se implemente utilizando punteros y no se asuma ningún agrupamiento en los nodos.

b) Consulta de coincidencia *parcial.* El número máximo de registros que satisfacen una consulta de coincidencia parcial que especifica *t* niveles viene dado por

$$t_1 t_2 \ldots t_k$$

donde

$$\_ \begin{cases} 1 & \text{si se especifica } A \\ \text{u;} & \text{si no se especifica } A \end{cases}$$

Utilizando las caracterizaciones medias de la Sección 6.1, podemos concluir que el número de registros que satisfacen una consulta de coincidencia parcial dada es *O(N' '!')* para datos uniformemente distribuidos. En la línea 3 del algoritmo MAT-SEARCH, para

niveles especificados, una búsqueda binaria en los conjuntos filiales. Para niveles no especificados, calificamos todo el conjunto filial. En el peor de los casos para nuestro algoritmo, los atributos hrst (k - t) no están especificados, y cada nodo calificado en el nivel *(k* - i) da lugar a un nodo calificado en cada nivel posterior. Por lo tanto, para cada nivel y, *tk - t*+ l) < j < *k,* tenemos *O(N'* ') conjuntos en los que buscar, y en cada nivel puede haber *OWN' !')* nodos cualificados. Por lo tanto, el coste de responder a una consulta de coincidencia parcial viene dado por

$$0(5/so- \quad s/ \quad (\log(ste+, \quad t) +---+ \tog(st))) = 0(tN' \,''''\, \log(/V'*'))$$

Observamos que la expresión anterior se aproxima a 0(log *N 1* a medida que se especifican más y más atributos. Pero, a medida que dejamos algunos de los atributos sin especificar, la complejidad aumenta. Esto se debe a que cada vez más nodos podrían ser calificados para una coincidencia parcial a medida que se especifican menos atributos.

Si el directorio se mantiene en almacenamiento secundario, el número de accesos a páginas correspondientes a los primeros *(k* - r) niveles no especificados es *O(N' '' '''* - un acceso a página por cada conjunto filial accedido. El número de páginas a las que se accede en los siguientes f niveles especificados es *O(tN' !'* '' '). Por lo tanto, el número total de accesos a páginas es *O'tN'* ' '"). A medida que se especifican más y más atributos, este coste se aproxima *a O(kN'* '). Si no se asume la linealización, cada nodo cualificado puede dar lugar a un acceso a la página, y por tanto el número de accesos a la página es *O(tN '!'* Iog *(N'!'))*. Si se utiliza la ordenación depth-first para la linealización, se podría demostrar que el número de accesos a la *página es O(tN'* ' '). El rendimiento de la linealización amplitud-primero es ligeramente mejor que el de la linealización profundidad-primero. Sin embargo, ambas linealizaciones son más eficientes que la implementación aleatoria. Hay que tener en cuenta que existe un cierto grado de agrupación de nodos en las páginas secundarias, y la suposición de aleatoriedad completa en la asignación de nodos no es válida. Sin embargo, proporciona un límite teórico superior en el número de accesos a páginas.

c) *Consulta de rangos.* El análisis del rendimiento de una consulta de rango basado en el peor de los casos tiene principalmente un significado teórico más que un gran valor práctico. En concreto, en el peor de los casos, una consulta de rango puede recuperar todos los *N* registros, lo que no es muy realista, especialmente en entornos de bases de datos de gran tamaño. Normalmente, el número de registros recuperados es pequeño, y las consultas que recuperan un gran número de registros son poco frecuentes. Por tanto, una estimación aproximada del número de registros recuperados cercana al valor medio es más adecuada para describir el rendimiento del sistema. Sin embargo, el análisis del caso medio es difícil porque no se conocen los patrones exactos del caso medio de una consulta y, además, el caso medio varía de un problema a otro. En esta sección, primero se consideran las consultas de rango generalizado y, a continuación, se analizan las consultas de rango restringido.

Normalmente, la complejidad temporal de una consulta de rango se expresa de la forma

$$0(f(N\,j+\quad K),$$

donde *k,* es el número de registros recuperados y *//(N*) da el coste que supone la búsqueda de los límites que delimitan los punteros de registro cualificados. Los principales esfuerzos en el diseño de algoritmos eficientes para la búsqueda de rangos se dirigen a disminuir *//(N* sin un aumento considerable de los costes de almacenamiento y preprocesamiento. Para una consulta de rango, el paso 3 del algoritmo MAT-SEARCH se modifica para buscar el límite superior e inferior del rango especificado para ese nivel. Los nodos que se encuentran dentro de los límites se recuperan y se añaden al tempset. Esta característica está bien soportada por la linealización breadth-first.

En el peor caso de la consulta de rango, todos los nodos de los conjuntos fiiales buscados están calificados. Por lo tanto, para un conjunto filial de tamaño *s,* el coste de la búsqueda de límites es 0(log(s)) y la recuperación del nodo calificado cuesta *0(s).* Utilizando la caracterización media del MAT, el coste de búsqueda en el nivel j calcularse como ((*, s;)* log(s;)). Dado que, en el peor de los , todos los nodos de un nivel están cualificados, el coste de procesar una consulta de rango para el nivel y es 0((s;) *log(s)+ ,'_ , s,).*

Por lo tanto, la complejidad temporal de una consulta de rango viene dada por $I_{,-}$ ' (, *st)+* $(\prod_{i=1}^{k-1}$ (.s;) log(s,) + K_1 donde el primer término corresponde al coste de procesar los primeros *(k* - 1) niveles. El segundo término corresponde al coste de la búsqueda de los límites de alcance del último nivel. El tercer término, *K ,*
corresponde al número de nodos calificados en el nivel final. El coste de una consulta de rango puede simplificarse y darse como sigue:

$$O(N'\quad !'\,log(N'^{k})+K\quad).$$

Si el directorio reside en almacenamiento secundario, se pueden utilizar argumentos similares a los anteriores para demostrar que el número de accesos a páginas es $O(N'\ '!^{(k)})$. Si no se utiliza la linealización y se supone una ubicación aleatoria de los nodos entre las páginas de almacenamiento secundario, entonces el número de accesos a páginas es $O(N'\ !^{k}+K)$. Sin embargo, como ya se ha señalado, esta suposición no es muy práctica. Una vez más, es interesante observar que la ordenación por profundidad también tiene una complejidad de búsqueda de rango de $O(N\ '+K)$ accesos a páginas.

Hasta ahora sólo hemos considerado las estimaciones del peor caso para una consulta de rango. La suposición de que todos los miembros de los conjuntos filiales satisfacen la consulta no es realista. Si imponemos la restricción de que para una consulta de rango no se califiquen más de nodos en cada conjunto filial buscado, entonces la complejidad de una consulta de rango puede darse como:

$$\sum_{j=1}^{k-1}\prod_{i=1}^{j}a^{i}+\left(\prod_{j=1}^{k-1}a^{j}\right)\log(s_k)+K_1$$

que puede aproximarse (por término medio) mediante

$$\boldsymbol{O(a'''(N''')}\ \text{-I- } Kg\ ).$$

Definamos p, denominada *fracción cualificada*, como

$$p = \frac{a + a^2 + \cdots + a^k}{N}$$

$$\frac{a(a^k - 1)}{(a - l)N}$$

$$= \frac{a^k}{N}, \quad \text{para o razonablemente grandes.}$$

La fracción calificada da una estimación media del valor máximo de la fracción de los nodos que se califican de un conjunto filial. Ahora el coste de la consulta de rango medio viene dado por

$$Q[\ p(N)''''\ \text{'''}(6'^{*}\ )+ K\quad)\ a\ 0(pN \qquad\qquad '^{*}'\ \log(/'/'\ ')+ Kg)$$

Por lo tanto, el valor de p determina el coste de la consulta de rango. Cuando el directorio reside en almacenamiento secundario, el número de accesos a páginas viene dado por $0[pN'\ '\ ')$.

La intuición para este perfil medio de una consulta de rango dada por Rao e Iyengar (1986) es que, en promedio, una consulta de rango define un "sub-MAT" que tiene un tamaño de conjunto filial de *ps,* en el nivel i, i= 1, 2, . , #. Esta visualización permite comprender mejor el proceso de respuesta a una consulta de rango en MAT. La tabla I resume el orden del coste de procesamiento de una consulta de rango para distintos valores de *p.*

El rendimiento del MAT mejora a medida que se reduce el número de registros recuperados por una consulta de rango. Este suele ser el caso de las aplicaciones

TABLA 1

ANÁLISIS DEL COSTE MEDIO DE LA CONSULTA ANGE DE BY PE @ - Qi

(N, indica el número de registros cualificados) "

Fracción cualificada *p*	Coste de consulta memoria principal	Coste de consulta memoria secundaria
$\left(\frac{N}{N}\right)$	$O(N'\ \ '!'\log(N'\ ')+ K,$	$O(N'\ '!')$
$f^{\frac{log}{N\ ]}}$	$0(\log N+\ \ )$	$0(\log N)$
$\left(\frac{\text{regis}\log N}{N}\right)$	$0\{log(N\ '''+ \log N)+ \text{K t})$	$0(\log(N\ '''+ \log N))$
, c= constante	$O(\log (N\ '''+ K,)$	$O(log(N\ '''))$

que implican grandes cantidades de datos, como las grandes bases de datos, los sistemas de tratamiento de imágenes, etc.

Concluimos esta sección señalando que, cuando se utiliza almacenamiento secundario para el directorio, la idoneidad de los TMA aumenta resultado de mantener los nodos de los conjuntos filiales juntos en páginas de almacenamiento secundario.

6.5 Comparación de resultados

En esta sección, comparamos el rendimiento de la estructura de datos MAT con otras estructuras de datos multidimensionales. La base de la comparación son los costes de consulta, preprocesamiento y almacenamiento cuando el directorio reside en la memoria principal. La comparación de la MAT con estructuras de datos como MDBT, kB-tree, BST-complex, etc. ya se ha realizado anteriormente. Todas tienen casi las mismas medidas de rendimiento, como se ha observado en las últimas secciones. A continuación, consideramos las estructuras de datos que no se parecen mucho a los TMA en los aspectos estructurales. Entre ellas se incluyen el escaneo secuencial, la proyección, el quadtree, el árbol #-d, el árbol de rangos, los rangos k no solapados y los rangos k solapados. Un excelente estudio de estas estructuras de datos para la búsqueda de rangos puede encontrarse en Bentley (1980).

Una consulta de coincidencia completa es de complejidad 0(log *N)* en la estructura de datos MAT, quadtree, £-d tree, etc. Del mismo modo, una consulta de coincidencia parcial puede responderse en *OWN' !k* log *(N '!j)* en el MAT. El escaneo secuencial tiene una complejidad de *O(N)* tanto para la consulta de coincidencia completa como para la parcial. La estructura de datos de proyección tiene complejidades de *Out* log *tN ' '!),* y *O(kN* ' ' ') para las consultas de coincidencia parcial y completa respectivamente. El árbol k-d y el quadtree tienen complejidades temporales de O(log *N)* y *OWN ' '!j* para consultas de coincidencia completa y parcial. Las demás estructuras de datos, como los árboles de rangos, el árbol d-fold, etc., están diseñadas específicamente para de rangos y, en general, no son adecuadas para consultas de coincidencias completas y parciales.

En la Tabla II se muestra la comparación rendimiento de varias estructuras de datos. Básicamente, hay cuatro tipos de estructuras de datos desde el punto de vista de la búsqueda de rangos. El primer tipo tiene una complejidad de *OWN* ' + *K,*), y el segundo tipo tiene una complejidad de *O(log+K,).* El tercero y el cuarto tienen complejidades de búsqueda de rango de 0(log *N + K)* y *0(N')* respectivamente. La estructura de datos MAT pertenece al primer . Cabe señalar que la reducción de un coste conlleva un aumento de los demás costes para cualquier estructura de datos. En el segundo tipo de estructura de datos, la reducción del coste de consulta a 0(log' + <) es el resultado del aumento de los costes de preprocesamiento y almacenamiento a *O(N* log' *' N)* y *O(N* log' *' N)* respectivamente. Este efecto es más prominente en los rangos k solapados. En esta estructura de datos, la reducción adicional de los consulta a O(log *N+ K)* da lugar a los elevados costes de preprocesamiento y consulta dados por *OWN* ' '), donde e es un número positivo. Se observa el efecto inverso

TABLE II

Corl r **RISON O PERFORM ANCE O ALGUNAS ESTRUCTURAS QATA K ULTIDIMENSIONALES**

Estructura de datos	Preprocesamiento COSt	Coste de almacenamiento	Consulta de la gama COSt
Exploración secuencial	*O[Nj*	*0(N)*	*0(N)*
Proyección	*O(N* log *N)*	*0(N)*	*0(N'* " "+K,)
Quadtree Árbol k-d	*0(N* log *N)*	*0(Nj*	*PROPIO'* " *+K,)*
MAT	*O(N* log *N)*	*PROPIA)*	*O(N ' '!'* log{N*'!')+ K)*
Árbol de rangos Árbol de d-pliegues Super B-árbol Quintanar y árbol	*O(N* log' *'N)*	*O(N* log' *'N)*	0(log' *N+ K,)*
árbol de k pliegues	*O(N* log' *'N j*	*O(N* log' 'N)	0(log' *'N+ K)*
Superposición de rangos k	*O(N ')*	*O(N ' -)*	O(log *N+ K)*
Sin solapamientos rangos k	*O(N* log *N)*	*0(N)*	*O N' -- K,)*

en Jr-rangos no solapados que tienen un coste de consulta de *O(N+ K)* y unos costes de preprocesamiento y almacenamiento *de O(N* log *N)* y *O(N)* respectivamente. Esto parece indicar que existe una estrecha relación entre los costes de consulta, preprocesamiento y almacenamiento.

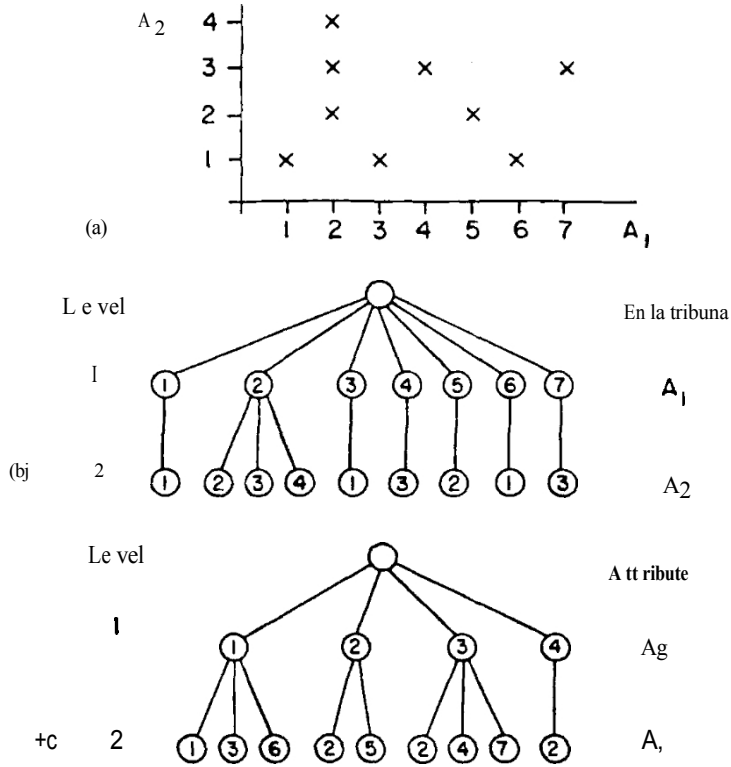
Cuando el MAT se organiza en almacenamiento secundario, su rendimiento debe compararse con el de estructuras como el archivo de cuadrícula, el hashing extendido, etc. Sin embargo, el análisis de la mayoría de estas estructuras no se lleva a cabo de la forma en que se hace para el . Esto dificulta mucho la comparación. Concluimos esta sección señalando que la MAT pertenece a la familia de estructuras de datos que contiene árboles k-d, quadtrees, etc. A pesar de su similitud en la complejidad en el peor de los casos, cada una es mejor que la otra en el caso medio de determinadas aplicaciones. Hay que estudiar a fondo las propiedades del problema para elegir una estructura de datos sobre la otra. Los aspectos que hacen que la estructura de datos MAT sea eficiente en determinadas aplicaciones se tratan en el sección neKta.

7. Paradigmas para la estructura de datos MAT

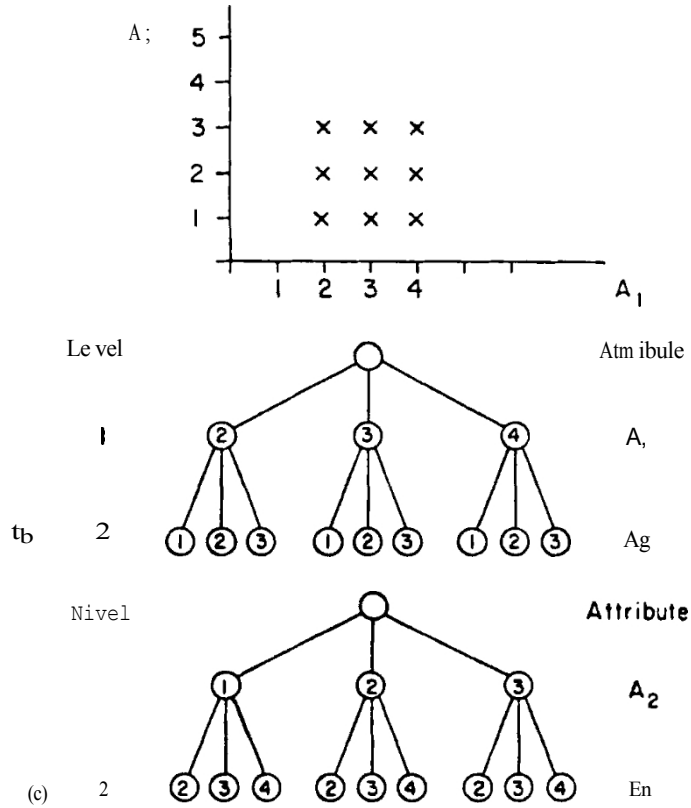
Aunque el MAT pertenece a la clase de los quadtrees y k-d-trees, su rendimiento en el peor de los casos puede ser mejor que el de otros en determinadas aplicaciones. Se pueden utilizar las propiedades de datos y las consultas para mejorar el

rendimiento de MAT en muchas aplicaciones de la vida real. Las propiedades de los datos, como el tamaño de los distintos conjuntos filiales, el orden de los atributos, etc., deciden el perfil del . Esto, a su vez, determina la complejidad de varias consultas. Las propiedades de consulta, como el tipo de consulta, la probabilidad de aparición de distintos atributos en una consulta, el número de registros que satisfacen la consulta, etc., deciden el rendimiento de las distintas consultas en la MAT. En esta sección se analizan brevemente estos dos aspectos en forma de paradigmas.

Paradigma I. Para el mismo número de registros, un TMA con un perfil que tiene menos conjuntos filiales grandes tiene mejores resultados que un TMA con más conjuntos filiales pequeños. Las figuras 12b y 13b muestran los TMA para los datos de las figuras 12a y 13a respectivamente. En general, los datos con "grandes conjuntos" tienen una estructura de MAT más eficiente que una MAT con datos "dispersos". Para cualquier dato dado, el perfil de



Flo. 12. (a) Datos de muestra; (b) MA T con ordenación al tributo *A , A z ; (c j* MA T con ordenación al atri tuto A_2, A .



Flci. 13. (a) Datos de muestra; (b) MAT con ordenación de atributos *A , A : (c)* MAT con ordenación de atributos A_2, *A* .

la MAT depende también de la ordenación de los atributos. La figura 12a ilustra que intercambiando los atributos *A* y A_2 se obtiene un MAT más compacto. La ordenación de los atributos de forma que el tamaño medio del conjunto filial disminuya hacia el
inferior, se demuestra que es óptimo para la respuesta media de los casos de una consulta de rango en Rao y lyengar (1986). Otro criterio para ordenar los atributos es minimizar el tamaño medio los conjuntos filiales en todos los niveles. Estos dos requisitos pueden no cumplirse simultáneamente. Es necesario llegar a un compromiso si ambos requisitos entran en conflicto.

Paradigma 2. Si los atributos que aparecen con más frecuencia se colocan en los niveles superiores, se realizará un mayor cribado de los nodos en los niveles superiores. El resultado habrá que buscar menos nodos en los inferiores. Este enfoque es adecuado para

consultas de coincidencia parcial. Para una consulta de rango, los atributos que probablemente den lugar a menos nodos cualificados pueden colocarse más arriba en la estructura **MAT**. El crecimiento del número de nodos a buscar se verá restringido como consecuencia de esta ordenación.

Los paradigmas anteriores sugieren que MAT es una estructura indexada prometedora para aplicaciones de recuperación de información.

$. Of'tCIU9ÎORS

El diseño y análisis de estructuras de datos multidimensionales y al- goritmos ha adquirido una importancia creciente en los últimos años, especialmente en las áreas de la geometría computacional, la robótica, la recuperación de información y la organización de bases de datos físicas. Como resultado, se ha propuesto y estudiado un gran número de estructuras de datos multidimensionales en diversas áreas de aplicación. En este artículo se presenta un breve estudio de estas estructuras de datos existentes junto con algunos paradigmas nuevos para equilibrar árboles de búsqueda multidimensionales. Además, hemos explorado el potencial de los Árboles de Atributos Múltiples como una prometedora estructura de datos multidimensional para su aplicación en problemas de consulta de rango. Este estudio no es en absoluto exhaustivo, y la omisión de cualquier estructura de datos no es intencionada.

Aquí hemos intentado ilustrar y destacar los siguientes aspectos:

1. La necesidad de desarrollar y analizar estructuras de árbol de búsqueda simples, eficientes, equilibradas y auto-organizadas que sean útiles para aplicaciones en tiempo real.
2. Un análisis relativo de las técnicas existentes para la extensión de árboles de búsqueda equilibrados a dominios multidimensionales.
3. Un modelo de árboles equilibrados multidimensionales y ponderados mediante paradigmas bien definidos.
4. Una descripción introductoria de los Árboles de Atributos Múltiples y el análisis de su rendimiento para la consulta de coincidencia completa, la consulta de coincidencia parcial y la consulta de rango.

Se espera que el estudio dé lugar a una mejor comprensión del funcionamiento, así como al desarrollo de nuevas características de rendimiento de las estructuras de datos multidimensionales. En vista del Proyecto de Computadoras de Quinta Generación en curso y del creciente énfasis en el uso de computadoras de alta velocidad, un estudio detallado de las técnicas para el diseño y análisis de estructuras de árbol de búsqueda multidimensional auto-organizadas equilibradas se vuelve bastante importante.

Las nociones de complejidad temporal y espacial son muy útiles a la hora elegir una estructura de datos adecuada para el cálculo de alta velocidad. Pero es

difícil analizar la estructura de datos del cálculo a la estrecha intercalación y la indeterminación de los costes de asignación.

El trabajo de Vaishnavi b1984, 1987) descrito en la Sección 3 presenta un marco para implementaciones de árboles multidimensionales y ponderados equilibrados y autoorganizados. Estas estructuras deben tener un rendimiento eficiente en el peor de los casos por operación. Los árboles de atributos múltiples son una estructura de datos multidimensional con especial relevancia en la organización física de bases de datos. Un tema importante de nuestro trabajo es la exploración de nuevos paradigmas para las estructuras mencionadas.

Podemos considerar los paradigmas aquí tratados a dos niveles diferentes: un nivel abstracto y un nivel de aplicación.

A nivel abstracto, los paradigmas debatidos pueden dividirse en dos categorías:

1. Estrategias para soportar las "violaciones de estructura" en los "árboles ponderados multidimensionales equilibrados generalizados".
2. Estrategias para explotar las propiedades estructurales de los árboles de atributos múltiples.

A de aplicación, los paradigmas de los árboles ponderados multidimensionales equilibrados pueden dividirse en cuatro categorías:

1. Las violaciones de la estructura deben ser "soportadas" por cada uno de los subárboles "hermanos" (de dimensión inmediatamente inferior) del nodo. Denominamos a este paradigma *estrategia apoyo local*.
2. En la siguiente estrategia, denominada *estrategia soporte global,* las violaciones de la estructura deben ser soportadas por cada uno de los subárboles "adyacentes" de los nodos causantes de la violación de la estructura.
3. En la *estrategia apoyo a vecinos*, las violaciones de estructura deben ser apoyadas por al menos uno de los subárboles "colindantes" de los nodos causantes de la violación de estructura.
4. Aprovechando las propiedades computacionales de la estructura de datos.

Estos paradigmas se han fundamentado con ejemplos adecuados. Nos gustaría destacar aquí que no existe un método universal para diseñar estructuras de datos multidimensionales eficientes. El truco está en identificar la formulación correcta del problema computacional seguida de una descomposición adecuada para una implementación eficiente. Las técnicas presentadas en este artículo pueden utilizarse para lograr la descomposición mencionada.

A CK NOW LEDGEMENTS

Los autores desean agradecer la ayuda recibida de los profesores D. Kraft y M oitra en cuanto a comentarios editoriales.

REFERENCIAS

Adel'son-Velskij, G. M., y Landis, Y. M. (1962). An algorithm for the Organization of lnformation. *Sonic I Maihrmaii' Dokl'iJ y 3* 1259 - 1263.

Aho, A. V., Hopcroft, J. E., y Ullman, J. D. (1974). "El diseño y análisis de algoritmos informáticos". Addison-Wesley, Reading, MA.

Alt, H., Mehlhorn, K., y M unro, J. I. (1981). Partial match retrieval in implicit data . *Prof'. ltllh Symp. Matlt. Foutiâalions oJ Comp. Sci,* pp. 156 - 161.

Avid, Z., y Shamir, E. (1981). A direct solution to range search and related problems for product . *Prof'. 22nd Annuul S) nip. on Foundation.s of Comp. Sci.,* pp. 1 23 -1 26.

Barnes, M. *C.,* y Collens, D. S. (1973). "Almacenamiento de estructuras jerárquicas de bases de datos en forma transpuesta". Datafair.

Bato ry, D. S. (1979). On searching transposed fi les. *ACM* Pr'sn.i. *Database Sv.st. 4,* 531 - 544. Bayer, R.

y McCreigh t, E. (1972). Organización y mantenimiento de grandes índices ordenados. *A"tu Inf!orinalica I,* 173 - 189.

Bens, S. W. (1982). Dynamic Weighted Data Structures. Tesis doctoral, Universidad de Stanford, Informe nº STAN-CS-82-916.

Bent, S. W., Sleator, D. D., y Tarjan, R. E. (I 980). Biased 2-3 Trees. *Proc.* 2/sr *Annu. IEEE Symp. on Fouitdalions of Corny. Sci.,* 248 - 254.

Bent. S. W., Sleator, D. D., y Tarjan, R. E. t1985). Árboles de búsqueda sesgados. *SIAM* 7. Competing 14, 545 - 568.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. c "'n. AcMin T), 509 - 517.

Bentley, J. L. (1977). Solutions to Klee's rectangle problems. Carnegie-Mellon University, Department of Computer Science, texto mecanografiado.

Bentley, J. L. (1979a). Problemas de búsqueda descomponibles. *Inf. Proces.s. Lei i.* 8 (5), 133 - 136.

Bentley, J. L. (1979b). M ultidimensional binary search trees in database applications. *IEEE Trans.* So/iit'. *E'uj.* SE-5 (4), 333 - 340.

Bent ley, J. L. (1980). M ultidimensional divide y vencerás. *Commun. ACM* 23 (4), 214 -229. Bentley, J.

L., y Burk hard, W. A. (1976). Heuristics for partial match retrieval data base design. *Inf: Pro"e.s.s Lett.* 4 (5), 132- 135.

Bentley, J. L., y Friedman, J. H. t1979). Estructuras de datos para la búsqueda de rangos. *ACM Comyut. Suri'r i .s* I I (4), 397 -409.

Bently, J. L., y McGeoch, C. C. (1985). A mortized analysis of self-organizing sequential search heuristics. *Common. ACM* 28, 404-41 1.

Bentley. J. L., y Ott mann, T. H. (1979). Algorithms for repo rling and counting geometric intersections. *IEEE Truns. Coritp.* C-28, 643 - 647.

Bentley. J. L., y Saxe, J. B. (t979). Algorithms on vector sets. *SIG ACT News 11,* 36 - 39.

Bentley, J. L., y Saxe, J. B. (1980). Decomposable searching problems 1: static-to-d ynamic t. *Journal of Algorithm.s* 1, 571 - 577.

Bentley, J. L., y Shamos, M. I. (1977). A problem in multivariate statistics: algorithm, data struct, and application. *Prof'. l5ih A Herron C"n/. on* Commun. *Control and Comyut,* 193 - 201.

Bentley, J. L., y Stanat, D. F. (1975). Analysis of range searches in quad trees. *Inf. Process. Lei i.* 4 (5), 170- 173.

Bolour, A. (1981). Optimal retrieval algorithms for small region queries. *SIAM J. Comput.* 10, 721- 741.

Burk hard, W. A. *(* 1983). I nterpolation based index maintenance. *Proc. ACM Symy. Principle.s of Daio/in.*se St'.stem.s, 76 - 89.

Cárdenas, A. F., y Sagamang, J. P. (1974). Double-chained tree data base organization -analysis and st rategies. I BM research report RJ 1374.

Casey, R. G. (t973). Design of tree structures for efficient querying. *Commun. ACM 16* (9), 546-556.
Chang, J. M., y Fu, K. S. (1981). Extended k-d tree database organization -a dynamic multiattribute clustering method. *IEEE Trans. Softw'. Enq.* SE-7 (3), 284-290.
Dobkin, D. P., y Lipton, R. (1976). Problemas de búsqueda multidimensional. *SIAM J.* C'omyof. 5, 181-1 86.
Edelsbrunner, H. (1980a). Búsqueda dinámica de intersección de rectángulos. Technische Universität Graz, Institut für Informationsverarbeitung, Informe F47.
Edelsbrunner, H. (1980b). Estructura dinámica de datos para consultas de intersección ortogonal. Informe nº 59 de la Technische Universität Graz.
Edelsbrunner, H. (1981). A note on dynamic range searching. *Boletín de los CT de la EA* 15, 34-40.
Edelsbrunner, H. (1983). A new approach to rectangle intersections. *lui. J. Comput. Math.* 13 (3 & 4), 209-229.
Edelsbrunner, H. (1984). Key-problems and key-methods in computational geometry. *STA CS84 Sytitp. Allí. Aspert.' o(Goniy. Sci.,* París, abril de 1984, Lect. Notes Comp. Springer-Verlag,

Edelsbrunner, H., y Maurer, H. A. (1981). Sobre la intersección de objetos ortogonales. *Inf. Pio"ess. Leti. 13,* 177 -1 81.
Edelsbrunner, H., y van Leeuwen, J. (1983). M ultidimensional data structures and algorithms: Bibliografia. IIG, Technische Universität Graz, Austria, Rep. 104.
Fagin, R., Nievergelt, J., Pippenger, N., y Strong, H. R. (1979). Extendible hashing-a fast method for dynamic files. *ACM Trans. Database Sy.st.* 4 (3), 315 - 344.
Feigenbaum, J. y Tarjan, R. E. (1983). Dos nuevos tipos de árboles de búsqueda sesgados. *Bell S y.st. Tech.* 7nurra/ 62, 3139- 3158.
Finkel, R. A., y Bentley, J. L. (1974). Quad trees -a data structure for retrieval on composite keys. *A' ta ln|'ormatica* 4 (1), 1 -9.
Fredkin, E. (1960). Trie Memory. Commen. *ACM* 3, 490-499.
Fredman, M. L. (1981a). The spanning lower bound on the complexity of orthogonal range queries. J. *Alyoriihm.s* 1, 77 -87.
Fredman, M. L. (1981 b). A lower bound on the complexity of orthogonal range queries. *J. ACM* 28, 696- 705.
Fredman, M. L. (1981). Lower bounds on the complexity or orthogonal range queries. *SIAM J. Comyut.* 10, 1-10.
Gopalakrishna, V., y Veni Mad havan, C. E. (1980). Performance evaluation attribute-based tree organization *ACM Trane. Datahuse. Sysi. 6* (1), 69- 87.
Gotlib, C. C., y Gotlib, L. R. (1978). Tipos de datos y . Prentice-Hall, Englewood Cliffs, N. J.
Gueting R. H. y K riegel, H. P. (1980). M ultidimensional B-tree: an efficient dynamic file structure for exact match queries. *Prof'. l0ih GI Annu. Conf.,* Informatik Fachberichte, Springer Verlag, 37-5 388.
Gueting R. H. y Kriegel H. P. (1981). Dynamic k-dimensional multiway search under time- varying access frequencies. *Proc. 5ih GI Can(. on Theoretical Comp.* Lecture Notes in Computer Science 104. Springer-Verlag, New York. Springer-Verlag, Nueva York, 135 -145.
Hart, J. H. (1981). Optimal Two dimensional range using biiiary range lists. Tech. Report 76-81, Dept. Computer Science, Unix. Kentucky.
Hirschberg, D. S. (1980). On the complexity of searching a set of vectors. *SIAM J. Comput.* 9, 126-129.
Horowitz, E., y Sahni, S. (1976). "Fundamentos de las estructuras de datos". Computer Science Press.
H, S., y Mehlhorn, K. (1982). Una nueva estructura de datos para representar listas ordenadas. *Aciu Inforniytica* 17, 157 -184.

I yengar, S. S., y Raman, V. (1983). Properties and applictions of forest of quadtrees for pictorial information. *BIT* 23, 472 -486.

Jones, L., y 1yengar, S. S. (1984). Space and time efficient virtual quad trees. *IEEE Tran.s. Puitern Aitul. Ma"li. Iniell.* PAM1-6 (2), 244- 247.

Kash yap, R. L., y Su bas, S. K. C., y Yao, S. B. (1977). Analysis of multiattribute tree database organization. *IEEE Tran,s. Soh* ir. *Eng.* SE-2 (6), 451 -467.

Kirk Patrick, D. G. (1983). Búsqueda óptima en subdivisiones planares. *SIAM J. Compui.* 12, 2-8 35.

Kn, D. E. (1974). "The art of computer program ming, Vol. 3: Sorting and searching". Addison- Wesley, Reading, Mass.

K osaraju, R. (198 I). Búsqueda localizada en listas ordenadas. *Proc. 14Ih Synip.* or *Theory o/ Computing,* 62 -69.

K riegel, H. P. (1951). Variants of multidimensional B-trees as dynamic index structure for associative retrieval database systems. *Prof. 7lh Conf. Graph Theoreti" Concept.' en Conifiut. S"i.,* Linz, Austria.

K riegel, H. P. (1984). Performance comparison of index structures for multi-key retrieval. *Prof'. Annu. Mertiny de SIG MOD.* 186-195.

K riegel, H. P., y Vaish navi, V. K. (198 la). M ultidimensional B*-trees. manuscrito inédito. K riegel, H. P., y Vaishnavi, Y. K. (198 I b). Weighed t multidimensional B-trees used as nearly optimal dynamic dictionaries. *Proc. I0ih Int. S vmp. on Math. Found. of Camf'ui. S"i.* 118, 410-417,

Kriegel, H. P., y Vaishnavi, V. K. (1981c). A nearly optimal d ynamic tree structure for partial- match queries with timevarying frequencies. *Prof'. Conf:* on *Information Sciences ancl Syst'n1.s,* 19-3 197.

Lee, D. T., y Preperata, F. P. (1984). Computational geometry -a survey. *IEEE Trans.* Cm'ilmers C-33 (12), 1072 -1 101.

Lee, D. T., y Wo rig, C. K. (1977). Worst case analysis of region and partial region searches in m ultidilnensional binary search trees and balanced quad trees. *Aera fnfurmaf i"s* 9, 23 -29.

Lee, D. T., y Wong, *C.* K. (1980). Quintary trees: a fi Ie structure for multidimensional database sistemas. *ACM Trun.s.* Jriiafias' *Sysl.* 5 (3), 33-9 353.

Lee, D. T., y Wong, C. K. (1981). Finding intersections of rectangles by range search. *J. Algorithm.s* 2, 33-7 347.

L'en, E. Y., Taylor. E. C.. Driscoll, R. J., y Reynolds, L. H. (1975). Binary search tree complex- towards the implementation, *Proc, Ver y Lurye Data Base Conf.* Framingliam, Mass, 540- 542. Liou, J. H., y Yao, S. B. (1977). Multi-dimensional clustering for data base organization. *Inf.* Sr.i/, 2, I S7 -195.

Litwin, W. (1980). Linear hashing. a new tool for fi Ie and table add ressing. *Proc. 6th Int. Conf: on Ver)' Lurt)e Dula Buses,* 212 -223.

Lueker, G. S. (1978). A data structures for ort hogonal range queries. *Prof. l9lli Annu. S temp. Fouii'iutioiis de Comput. S"i.,* 28 -34.

Lueker, G. S. (1979). A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems. Tech. Report # 129, Dept. 1nformation and Computer Science, Univ. California, Irvine.

Lueker, G. S. (1982). A data structure to r d ynamic range queries. *Enf.* f'rn' . *Left.* 15, 209- 213. Lum, V. Y. (1970). M ulti-attribute retrieval with combined indexes. Cr'mmon. *ACM* 13, 660 -665.

McC reight, E. M. (1980). Efficient algoril h ms for enumerating intersecting intervals and rectangles. Informe de Xerox Parc, CSL-80-09.

Meh lhorn, K. (1979). Búsqueda binaria dinámica. *SIAM J. Computing* 8, 175 -195.

Mehlhorn, K. (1981). Lower bounds on the etficiency of t ransforming static data structures to estructuras de datos dinámicas. *Math. Sy.s. Theo.* 15, 1 -11.

Meh1horn, K. (1984a). "Estructuras de datos y algoritmos 1: Ordenación y búsqueda". Springer-Verlag.

Mehlhorn, K. (1984b). "Estructuras de datos y algoritmos 3: Estructuras de árbol multidimensionales y geometría computacional". Springer-Verlag.

Mehlhorn, K., y Overmars, M. H. (1981). D ynamiZación óptima de problemas descomponibles. *Inf. Pro"cs.i. Leit.* 12, 93 -98.

Nievergelt, J., y Reingold, E. M. (1973). Árboles de búsqueda binarios de equilibrio acotado. *SIAM J. Comput.* 2, 33 -43.

Nievergelt, J., Hinterberger, H., y Sevcik, K. C. (1984). The grid-file: an adaptable, symmetric multi-key file structures. *ACIvf Tran.s.* on *Database Sys.* 9, 38-71.

Orenstein. J. A. (1982). TRI multidimensional Es utilizado para la búsqueda asociativa. *Inform. Proc.* Mn. 4 (4), 150-158.

Ouskel, M., y Scheuermann, P. (1981). M ultidimensional B-trees: analysis of dynamic ' behaviour. *BIT* 21, 401 -418.

Overmars, M. H. (t981). Dinamización de problemas de conjuntos descomponibles por orden. J. *Algorithms* 2, 245-260.

Overmars, M. H. (1984). "El diseño de estructuras de datos dinámicas". Lec/. *Nores iii Comput. Sci.* **156,** Springer-Yerlag, Berlín.

Overmars, M. H., y van Leeuwen, J. (1982). Dynamic multidimensional data structures based on quad- and k-d trees. *Acto lnformatica* 17, 267-285.

Pealtz, J. L., Berman, W. J., y Cagley, E. M. (1950). Partial match retrieval using indexed descriptor files. Common. *ACM* 23, (9), 522 -528.

Preparata, F. P., y Shamos, M. I. (1985). "Geometría computacional". Springer-Verlag, Nueva York.

Rao, N. S. V. (1984). Multiple attribute tree based search and dynamization algorithms. M. E. Thesis, School of Automation, Indian Institute of Science, Bangalore, I ndia.

Rao, N. S. V., y I yengar, S. S. (1986). Optimal attribute ranking problem in MAT data structure. *Int. J. of' Compui. Math. 21,* 1 -12.

Rao, N. S. V., Vaishnavi, V. K. e Iyengar, S. S. (1987). On the dynamization of data structures. Tech. Report # 85-031, Dept. of Computer Science, Louisiana State Univ., to appear in *Bl T.* Rao, N. S. V" Veni Madhavan, C. E., and Iyengar, S. S. (1984). Range search and dynamization algorithms in multiple attribute tree. Tech. Report, School of Automation, Indian Institute of Science, Bangalore, I ndia.

Rao, N. S. V., Yeni Madhavan, C. E., y lyengar, S. S. (1985). A comparative study of multiple attribute tree and inverted file structures for large bibliographic files. *Inf. Process. and M ymi.* 18 (S) 200-2I3.

Rivest, R. L. (1976). Algoritmos de recuperación de coincidencias parciales. *SIAM J. Comput.* 5, 19- 50.

Rothnie, J. B., y Lozano, T. (1974). Organización de ficheros basada en atributos en un entorno de memoria paginada. C'ommun. *ACM* 17 (2). 63 -69.

Samet, H. (1983). A quadtree medial axis transform. C'ozrimun. *ACM* 26 (9), 680-693.

Samet, H. (1984). The quadtree and related hierarchical data structures. *Compui. Surgeys* 16 (2), 187 -260.

Scheuermann, P. y Ouksel, M. (1982). Multidimensional B-trees for Associative Searching in Database Systems. *Inf. SV.sI.* 7, I 23- 137.

Shamos, M. I. (1978). Computational Geometry. Ph.D. Thesis, Dept. Comput. Sci., Yale Univ. Shamos, *M.* I., and Hoey, D. (1975). Closest point problems. *Proc. lbih Annu. IEEE Syniy. FounJations of Comput. S"i., I* 51-162.

Sleator, D. D., y Tarjan, R. E. (1983a). Árboles de búsqueda binarios autoajustables. *Proc. 15th Annu. Symp. on Th'-cry of Comyuiing,* 235-245.

Sleator, D. D., y Tarjan, R. E. (1983b). A data slructure for dynamic . J. Comput. end Syst. *Sci.* 26, 362-391.

Sleator, D. D., y Tarjan, R. E. (1985). Árboles de búsqueda binaria autoajustables. J. ACM 32(3), 652 -686.

Subas, S. K. C. y K wish yap, R. L. (1975). Database organization -an access and storage method. Tech. Report # TR EE 75-32, School of Engineering, Purdue Univ.

Sussengut h, E. H., I r. (1963). Utilización de estructuras arbóreas para el tratamiento de ficheros. C"mmiin. *ACM 6* (5) 272 -279.

Tann minen, M. (1982). The extendible cell method for closest point problems. *BIT* 22, 27 -41.

Tsakalidis, A. K. {1984). Una implementación óptima para la búsqueda localizada. Informe técnico A S4/0fi, Universidad de Sarlandes, Alemania Occidental.

Vaishnavi, V. K. (1982). Computing point enclosures. *IEEE Trans. Comput.* C-31 (1), 2-2 29.

Vaisfinavi, V. K. (1953a). On the worst-case efficient implementation of weighted dynamic dictionaries. *Prof'. 21st Annu. A Herron ConJ: on Communication, Control, and Computing,* 647 -655.

Vaishnavi, Y. K. (1983b). Cómo equilibrar un árbol de búsqueda multidimensional. *Pro". 3rJ Conf. on Ft'uiiJulion.s of Soft" care Te"hnoloy y and Theor'- lirul Computer Science,* Bangalore, India.

Vaishnavi, V. K. (1954). M ult idimensional heigh t-balanced trees. *IEEE Trans. Comput.* C-33 (4), 334 -343.

Vaishnavi, V. K. (1956). On the height of multidimensional heigh t-balanced trees. *IEEE Trans. Comput.* C-35, 773 - 750.

Vaish navi, V. K. (1987). Weighted leaf A VL-trees. *Sl A M J. Comput.,* 16 (3) 503 - 537.

Vaish navi, V. K., y Wood, D.{1980). Data structures for rectangle containment and enclosure problems. *Computer Grophi"s nnd Image Pierces.* 13, 372 - 384.

Veni Mad havan, C. E. (1984). Secondary retrieval using tree data structures. *Theor. Comyut. S"i.* 33, 107 - 116.

Willard, D. E. (1979a). El algoritmo del superárbol B. Tech. Report TR-03-79, Aiken Computation Lab., Harvard *Unix.*

Willard, D. E. (1979b). "Algoritmos de búsqueda en bases de datos orientados a predicados". Garland Pub. Company, Nueva York.

Willard, D. E. (1982). Polygon retrieval. *SIAM J. Coriiyut.* 11 (1), 149-165.

Willard, D. E. (1985). Nueva estructura de datos para consultas ortogonales. *SIA M J. Compui.* 14 (1) 232 -253.

Willard, D. E., y Lueiter, G. S. (1985). Adición de la capacidad de restricción de rango a los datos dinámicos estructuras. J. *AC M* 32 (3), 597 -617.