Árboles Biselados

Mg. Augusto Cortez Vásquez[1, 2], Mg. Hugo Vega Huerta[1, 2]

[1]Facultad de Ingeniería de Sistemas e Informática,
Universidad Nacional Mayor de San Marcos

[2] Facultad de Ingeniería
Universidad Ricardo Palma

acortezv@unmsm.edu.pe, hvegah@unmsm.edu.pe

RESUMEN

El objetivo del presente estudio es evaluar el orden de complejidad de las operaciones de búsqueda, inserción o eliminación en un árbol biselado. Para el despliegue de un nodo se utiliza la técnica de rotaciones, muy utilizada en los árboles AVL. Para la evaluación de la complejidad de las operaciones se utiliza el método de análisis amortizado. El análisis amortizado en árboles biselados es beneficioso por cuanto se aplica cuando se realiza una sucesión de *m* operaciones de tal forma que en conjunto el tiempo de la operación sea a lo más O (m Log n), aunque individualmente cada operación pueda ser de orden O(n).

Palabras clave: Árboles de búsqueda, árbol desplegado, árbol biselado, análisis amortizado.

ABSTRACT

The objective of this study is to assess the complexity order of search operations, insertion and/or elimination in a beveled tree. For the deployment of a node the very utilized rotation technique is used in the trees AVL. For the evaluation of the complexity of operations the amortized method of analysis is used. The amortized analysis in beveled trees is beneficial insofar as is applied when a succession of m operations is realized for so that overall the time of the operation ought to be no more of O (m Log n), although individually every operation can be of order O(n).

Key words: Search tree, splay tree, amortized analysis.

1. INTRODUCCIÓN

Los árboles biselados son árboles binarios de búsqueda autobalanceables con la propiedad de que se optimizan automáticamente mientras son accesados.

Debido a que en un árbol biselado cada operación requiere un despliegue, el costo amortizado de cualquier operación está dentro de un factor constante del costo amortizado de un desplegado. De aquí que todas las operaciones sobre árboles desplegados toman un tiempo amortizado de O (Log n). En casos extremos, en un árbol normal, una operación de búsqueda podría ocurrir en el orden O(n) en el peor caso y de O (1) en el mejor caso. Sin embargo, cuando se realizan *m* operaciones se demuestra que tarda a lo más O (m Log n), para ello utilizamos la técnica de análisis amortizado. En un árbol binario convencional se pretende mantener el equilibrio para que los accesos sean homogéneos, debiéndose pagar el costo de mantener el equilibrio; mientras que en el caso de los árboles biselados, no es necesario mantener el equilibrio, preocupándose en su lugar de asegurar que un tipo de comportamiento no puede producirse repetidamente.

2. FUNDAMENTACIÓN TEÓRICA

2.1. Árboles

Los árboles son estructuras de datos no lineales de naturaleza dinámica y abstracta, son de naturaleza abstracta porque se pueden implementar de muchas formas sobre otra estructura (vectores, cursores, apuntadores etc.). Las operaciones básicas que se realizan sobre árboles son: búsqueda, inserción y eliminación, en adelante nos referiremos a estas operaciones como "operaciones OP".

2.2. Árboles biselados

Un árbol biselado también conocido como árbol desplegado (AD) es un arbol binario de búsqueda autobalanceable con la propiedad adicional de que a los elementos recientemente accedidos se accederá más rápidamente en accesos posteriores. Realiza operaciones básicas como pueden ser la inserción, la búsqueda y el borrado en un tiempo del orden de O (Log n). Esta estructura de datos fue inventada por Robert Tarjan y Daniel Sleator. Las operaciones OP en un árbol biselado son combinadas con una operación básica, llamada biselación de allí su nombre "árboles biselados". Esta operación consiste en reorganizar el árbol para un cierto elemento, colocando éste en la raíz. Una manera de hacerlo es realizando primero una búsqueda binaria en el árbol para encontrar el elemento en cuestión y, a continuación, usar rotaciones de árboles de una manera específica para traer el elemento a la raíz.

Lo que se pretende es utilizar una estructura en la que dada una secuencia de m operaciones la complejidad temporal sea óptima. Algunos modelos de representación son los árboles AVL, los árboles enhebrados, los árboles B.

Un árbol también se concibe como un grafo acíclico, conexo y dirigido [3]. Cada operación OP requiere un tiempo que está en el orden O (Log n), en el peor caso, donde n es el número de nodos en el árbol. Las técnicas más antiguas son los árboles AVL y los árboles 2-3. Las técnicas menos antiguas incluyen a los árboles rojinegros y los árboles biselados. La idea básica de los AD es que después de accesado un nodo, este se despliegue ("desplace") a la raíz a través de rotaciones como se hace en los árboles AVL.

La complejidad de las operaciones OP en un árbol depende del equilibrio del árbol. Se dice que un árbol está equilibrado, si para todo nodo del árbol la altura entre el subárbol izquierdo y el subárbol derecho difieren a lo más en uno. Un árbol AVL es un árbol con condición de equilibrio.

En un árbol binario de búsqueda es fácil de implementar las operaciones OP. Cuando el árbol está completamente equilibrado, lo cual hace más homogénea las operaciones, la complejidad temporal es de orden logarítmico O(Log n). El problema ocurre cuando se dan los casos extremos, por ejemplo cuando tenemos un árbol degenerado o sesgado (una lista), en este caso, en promedio el número de exploraciones para encontrar o no un valor es de (N+1) /2 siendo del orden O(N).

Cuando se construye un árbol, al introducir n nodos, se formará un árbol cuya forma varía de acuerdo a la manera en que se encuentran los datos. Si se modifica el orden generará un árbol diferente, posiblemente con pérdida de equilibrio. Al desequilibrarse un árbol se pierde eficiencia en cada operación. Normalmente se trata de que el árbol se mantenga equilibrado para mejorar la eficiencia; sin embargo equilibrar un árbol conlleva un mayor tiempo de proceso, que puede ser significativo. Una solución es tener un árbol completo o cuasicompleto. Un árbol es completo si los nodos

se ingresan por nivel de izquierda a derecha, y no se puede avanzar de nivel hasta que el nivel actual este completo. [17]

En la Figura N.° 1, (a) y (b) son ejemplos de árboles completos, mientras que (c) y (d) son ejemplos de árboles incompletos.

En un árbol completo, si se efectúa una operación OP, el máximo número de exploraciones que hay que realizar para encontrar o no un valor está en función de la altura del árbol H ≅ Log N.

Tabla N.° 1. Parámetros de los árboles

N número de nodos	H altura	Número de exploraciones H+1
1	0	1
2 a 3	1	2
4 a 7	2	3
8 a 15	3	4

Si T es un árbol binario completo con I nodos internos, entonces T tiene I+1 hojas y 2I+1 nodos en total. [10]

Los nodos de T constan de los nodos que son hijos (de algún padre) y los nodos que no son hijos (de ningún padre). Existe un nodo que no es hijo de nadie: la raíz.

Como existe I nodos internos, cada uno de los cuales tiene 2 hijos, existe 2I hijos. Así, la cantidad total de vértices de T es 2I+1 y el número de hojas es (2I+1)- I = I+1.

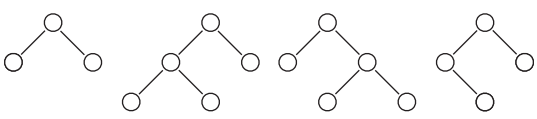
El tiempo necesario para realizar la búsqueda en el peor caso es la altura del árbol H más uno (H+1).

Log t ≤ H, donde t es el número de hojas de T

En el peor caso t = I+1

Así Log t = Log (I+1) ≤ H

Nótese que Log (2,000,000+1) = 21, lo cual significa que es posible guardar dos millones de datos en un árbol de búsqueda binario y encontrar uno de esos elementos o determinar que no lo está en a lo más 21 pasos.



(a) (b) (c) (d)

Figura N.° 1. Ejemplos de árboles completos e incompletos.

Otra solución es utilizar árboles AVL. En un árbol AVL, cada vez que se realiza una operación OP se tiene verificar la condición de equilibrio, y esto causa pérdida de eficiencia.

Los árboles biselados, también llamados árboles desplegados (*splay trees*) son árboles de búsqueda auto-organizados o autobalanceables, que emplean rotaciones similares a las que se utilizan en los árboles AVL, con el fin de mover cualquier clave accesada, en cualquier operación OP hacia la raíz. Esto es debido a que se ha comprobado que si algo ha sido accesado, es muy probable que sea nuevamente accesado. Haciendo que la posterior búsqueda sea eficiente.

Los AD son más fáciles que implementar que los árboles AVL, dado que no interesa verificar su condición de equilibrio.

En el caso de los árboles AD se lleva el elemento en cada operación a la posición de la raíz. Esto puede hacerse de dos formas. Forma bottom-up, se realiza un recorrido desde la raíz hasta encontrar el elemento buscado; o bien hasta encontrar una hoja, en caso de inserción. Luego de lo anterior se realiza una operación splay para mover el elemento a la posición de la raíz.

Esta estructura garantiza que para cualquier secuencia de M operaciones en un árbol, empezando desde un árbol vacío, toma a lo más un tiempo de O(M log N). A pesar que esto no garantiza que alguna operación en particular tome un tiempo de O(N), si asegura que no existe ninguna secuencia de operaciones que sea mala. En general, cuando una secuencia de M operaciones toma tiempo O(M f(N)), se dice que el costo amortizado en tiempo de cada operación es O(f(N)). Por lo tanto, en un AD los costos amortizados por operación son de O (Log(N)).

La estrategia de biselación es similar a la idea de las rotaciones simples. Si el nodo *k* es accesado, se realizarán rotaciones para llevarlo hasta la raíz del árbol. Sea *k* un nodo distinto a la raíz del árbol. Si el padre de *k* es la raíz del árbol, entonces se realiza una rotación simple entre estos dos nodos. En caso contrario, el nodo *k* posee un nodo padre *p* y un nodo "abuelo" *a*. Para realizar las rotaciones se deben considerar dos casos posibles (más los casos simétricos).

2.3. Tipos de algoritmos

Existen dos tipos de algoritmos, biselación ascendente (de abajo hacia arriba) o biselación descendente (de arriba hacia abajo).

2.3.1. Biselación ascendente

Las operaciones OP se realizan en forma similar a un árbol binario de búsqueda. Luego se realiza una operación biselación sobre un nodo. En una búsqueda, se devuelve el nodo que contiene el valor buscado, o el padre de la hoja si no lo encuentra. En una inserción, el nodo sobre el que se aplica la operación biselación es el de igual valor al buscado, si ya existía; o el nuevo nodo si éste no estaba en el árbol. En biselación ascendente se requiere descender de la raíz hasta el nodo al que se le aplicará la operación biselación. Luego se van efectuado las rotaciones a medida que se asciende. Es decir se recorre el árbol dos veces. A partir del nodo, al que se le aplicará la operación, se asciende hasta encontrar el abuelo, y se efectúa la rotación doble que corresponda; si no existe abuelo, pero sí padre, se efectúa rotación simple.

2.3.2. Biselación descendente

Consiste en partir el árbol en dos subárboles, uno con claves menores al buscado y otro con claves mayores al buscado, y a medida que se desciende se van efectuado las rotaciones. Cuando se encuentra el nodo en la raíz del subárbol central, se unen los subárboles, dejando como raíz al nodo. Cada vez que se desciende desde un nodo x, por un enlace izquierdo, entonces x y su subárbol derecho serán mayores que el nodo (que será insertado o que es buscado). De esta forma se puede formar un subárbol, con x y su subárbol derecho, sea este subárbol DER. El caso simétrico, que se produce cuando se sigue un enlace derecho, permite identificar el subárbol izquierdo de la nueva raíz, sea este subárbol denominado IZQ. Como se recorre sólo una vez, ocupa la mitad del tiempo que el ascendente. Se mantienen punteros a IZQ y DER, y punteros a los puntos de inserción de nuevos nodos en IZQ y DER; éstos son el hijo derecho del máximo elemento de IZQ; y el hijo izquierdo del mínimo elemento de DER. Estas variables evitan la necesidad de recorrer IZQ y DER; los nodos y subárboles que se agreguen a IZQ o DER, no cambian sus posiciones en IZQ o DER. A partir de la raíz se desciende hasta encontrar un posible nieto, se efectúa la operación pasando el abuelo y el padre a los subárboles IZQ y DER; el nieto queda en la raíz del árbol central. Si se encuentra el nodo se efectúa una unión final.

2.4. Análisis de complejidad en el peor caso, mejor caso y caso promedio

La complejidad temporal está dada por el número de operaciones que realiza el algoritmo. Durante el análisis de complejidad temporal, se dan tres casos: caso mejor, caso promedio y caso peor. Generalmente se considera el peor caso. El análisis por amortizaciones significa analizar los costos promedios para cada secuencia de operaciones.

2.5. Análisis de complejidad mediante amortizaciones

Un algoritmo realiza varias operaciones, unas más costosas que las otras. Lo que se pretende es pagar más (prepago) por una operación, considerando que posteriormente se pague menos por operaciones más costosas. El objetivo es que se adelante los pagos, para pagar menos posteriormente, cuidando que nunca se aumente la deuda.

El análisis de amortizaciones tiene sentido cuando se aplica a una secuencia de operaciones, porque lo que interesa es el costo promedio. Esto es ventajoso frente al análisis de complejidad temporal de peor caso. El método consiste en asignar un costo artificial al que denominaremos "costo amortizado", en lugar de utilizar un costo real. El costo amortizado nunca excederá al costo total amortizado de todas las operaciones. Por tanto, para analizar un algoritmo se pueden emplear los costos amortizados, en lugar de los reales. La ventaja es que existe cierta flexibilidad en la asignación de los costos amortizados.

2.6. Potencial

Una estructura de datos es cambiante en función de los datos contenidos. La estructura de datos utilizada siempre tiene un estado al que denominaremos estado actual. El estado en cualquier momento está dado por una función conocida Potencial, que no es mantenida por el programa, sino que más bien es un dispositivo de contabilidad que ayudara en el análisis [18]. Asignar un potencial consiste en asociar créditos con la estructura completa de los datos. Su principal dificultad es escoger una función potencial. Su elección no siempre es obvia, y podría ignorar detalles de estructura. A menudo se la elige por el método de ensayo y error, en otros casos, a veces, por intuición.

$$T_{amortizado} = T_{real} + \delta_{potencial}$$

δ es la diferencia entre el tiempo real y el tiempo amortizado.

El tiempo real de una operación representa la cantidad exacta de tiempo requerido para ejecutar una operacio-en particular. El tiempo amortizado es igual al tiempo real más un incremento δ potencial.

Se observa que mientras T_{real} varía de operación a operación, $T_{amortizado}$ es estable.

Dada una secuencia de m operaciones op_1, op_2 ….op_m asumiremos que la estructura de datos tiene una función potencial. La función potencial puede pensarse en forma análoga como el pago de una cuenta bancaria.

Sea D_i: estado de la estructura de datos después de la operación i-ésima.

$\partial(D_i)$: potencial de la estructura completa D_i

C_i: costo actual de la operación i-ésima, lo que corresponde a los intereses. Cada operación op_i tiene un costo proporcional a su tiempo de ejecución.

Se pagan los costos Ci, de las operaciones op_i mediante amortizaciones á

Por tanto:

á = Ci + $\partial(D_i)- \partial(D_{i-1})$ = costo amortizado de la operación i-ésima

La diferencia entre el costo real y las amortizaciones se carga al potencial de la estructura. Se aumenta (invierte o paga) el potencial si los costos amortizados representan un sobrepago de la operación i-ésima, en cuyo caso el potencial aumenta.

Lo que interesa fundamentalmente es el costo promedio en una secuencia de operaciones, por tanto para m operaciones, se tiene:

$$\sum_{i=1}^{m} \acute{a} = \sum_{i=1}^{m} (Ci + \partial(D_i) - \partial(D_{i-1}))$$

$$\sum_{i=1}^{m} \acute{a} \geq \sum_{i=1}^{m} (Ci) \quad si \quad \partial(D_m) > \partial(D_0)$$

Pero la diferencia de potencial es una serie telescópica, y se puede escribir de la siguiente forma:

$$\sum_{i=1}^{m} \acute{a} = \sum_{i=1}^{m} (Ci) + \partial(D_m)- \partial(D_0)$$

Lo cual permite establecer que: Entonces el costo amortizado total será una cota superior del costo real.

ANÁLISIS AMORTIZADO PARA ÁRBOLES DESPLEGADOS

Después de realizar un acceso a algún elemento X, un paso de despliegue mueve X a la raíz por medio de una serie de tres operaciones ZIG, ZIG-ZAG o ZIG-ZIG

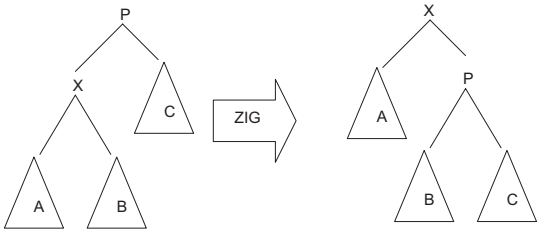


Figura 2. Rotación ZIG.

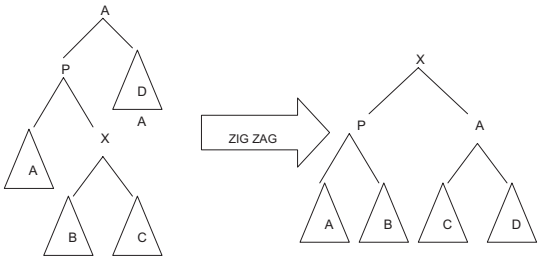


Figura 3. Rotación ZIG ZAG.

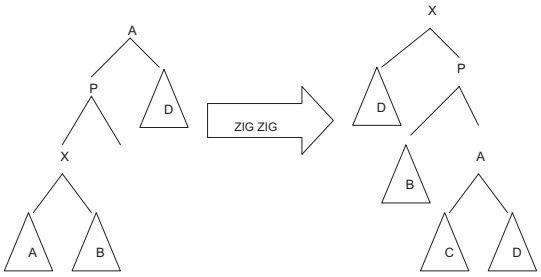


Figura 4. Rotación ZIG ZIG.

Cuando se quiere accesar a un nodo, tiene que realizarse una exploración por la rama que probablemente contenga lo que se busca. El tiempo requerido para cualquier operación de árbol sobre el nodo X es proporcional al número de nodos en el camino de la raíz a X. Cada operación ZIG significa una rotación, mientras que cada operación ZIG-ZAG o ZIG-ZIG implica dos operaciones. Por tanto el costo de cualquier acceso es igual a el número de rotaciones más uno.

R(i) representa el rango del nodo i. Consideramos que el nodo i es sucesor del nodo j, si existe camino desde j hacia i (j se dice que es antecesor de i). i es antecesor y sucesor de sí mismo.

R(i)=Log(S(i)) donde S(i) denota el número de sucesores de i

∅(A) función potencial

$\varnothing(A) = \sum_{i \in A} \text{Log } S_i$ donde representa el número de sucesores de i

Sea R(i) = $\text{Log } S_i$ donde representa el número de sucesores de i

Esto hace que

$\varnothing(A) = \sum_{i \in A} \mathbf{R(i)}$ suma de los rangos de los nodos de A

Para un árbol A con n nodos, el rango de la raíz es simplemente R(A) = Log (n)

Alternativas de función potencial

$\varnothing_1$(A): suma de los rangos de los nodos

$\varnothing_2$(A): suma de las alturas de los nodos

Una rotación puede cambiar las alturas de muchos nodos del árbol, pero solo puede cambiar los rangos de x, p y a, los demás no cambian

Se prueba que

Si a+b ≤ c entonces Log a + Log b ≤ 2 Log c - 2 (a)

Por tanto el tiempo amortizado para desplazar un árbol con raíz A en el nodo X es :

3(R(A) –R(X)) + 1 = O(log) como máximo (b)

Por tanto, el tiempo amortizado para desplazar un árbol con raíz A en el nodo X es:

3(R(A) –R(X)) + 1 = O(log) a lo mas

Consideremos los tres casos:

– **Si X es la raíz**

No existen rotaciones, por lo tanto no hay cambio de potencial. De esto se deduce que el tiempo real es 1 para tener acceso al nodo. El tiempo amortizado es 1. Por tanto se cumple la ecuación (b).

Se puede suponer que al menos hay una rotación.

– **Si X no es la raíz**

Para todo paso de despliegue X diferente de la raíz

R_i(X) rango de X antes del paso de despliegue

T_i(X) tamaño de X antes del paso de despliegue

R_f(X) rango de X inmediatamente después del paso de despliegue

T_f(X) tamaño de X inmediatamente después del paso de despliegue

– **Caso ZIG**

Cambio de potencial: costo real = 1

$R_f(x) + R_f(p) - R_i(x) + R_i(p)$

Solo los árboles de x y p cambian de tamaño.

Por tanto:

$TA_{ZIG} = 1+R_f(x)+R_f(p) - R_i(x) - R_i(p)$

Después de la rotación ZIG $T_i(p) \geq T_f(p)$ pues p desciende de nivel.

De ello se deduce que $R_i(p) \geq R_f(p)$ por lo tanto

$TA_{ZIG} \leq 1 + R_f(x) - R_i(x)$

Como $T_f(x) \geq T_i(x)$ se deduce que $R_f(x) - R_i(x) \geq 0$

Esto implica que se puede incrementar el lado izquierdo de la ecuación

$TA_{ZIG} \leq 1+3(R_f(x) - R_i(x))$

– **Caso ZIG- ZAG**

Cambio de potencial: costo real = 2

$R_f(x) + R_f(p) + R_f(a) - R_i(x) - R_i(p) - R_i(a)$

Sumadas ambas expresiones, tenemos

$TA_{ZIG-ZAG} = 2+R_f(x)+R_f(p)+R_f(a)-R_i(x)-R_i(p) - R_i(a)$

Pero vemos que $T_f(x) = T_i(a)$ por lo que sus rangos deben ser iguales

Por tanto:

$TA_{ZIG-ZAG} = 2+R_f(p) + R_f(a) - R_i(x) + R_i(p)$

Además $T_i(p) \geq T_i(x)$ por lo que $R_i(x) \leq R_i(p)$

Sustituyendo, se obtiene:

$TA_{ZIG-ZAG} = \leq 2+R_f(p)+R_f(a) -2R_i(x)$ (b)

además

$T_f(p) + T_f(a) \leq T_f(x)$

Aplicando la ecuación (a)

$\text{Log}(T_f(p)) + \text{Log}(T_f(a)) \leq 2(\text{Log } T_f(x)) - 2$

Lo cual quiere decir que

$R_f(p) + R_f(a) \leq 2 R_f(x) - 2$ (c)

Reemplazando (a) en (b) se tiene que

$TA_{ZIG-ZAG} \leq 2 R_f(x) - 2 R_i(x) = 2 (R_f(x) - R_i(x))$

Como $Rf(x) \geq R_i(x)$

Se tiene que

$TA_{ZIG\text{-}ZAG} \leq 2R_f(x) - 2R_i(x) = 2(R_f(x) - R_i(x)) \leq 3(R_f(x) - R_i(x))$

Resulta obvio que en un árbol desplegado siempre que se tenga acceso a un nodo, este debe ser movido. Si no fuera así, el nodo no cambia de posición, y en cada acceso cuesta O(n), entonces una secuencia de m accesos costará O(m.n). En un árbol desplegado se tiene O (m f(n)), decimos que el tiempo de ejecución amortizado es de O (f(n)) así un árbol desplegado tiene costo amortizado O (Log n)[17]. El caso ZIG-ZIG: se desarrolla en forma análoga.

3. DISCUSIÓN

Debido a que en un árbol biselado cada operación requiere un despliegue, el costo amortizado de cualquier operación está dentro de un factor constante del costo amortizado de un desplegado. De aquí que todos las operaciones sobre árboles desplegados toman un tiempo amortizado de O (Log n).

En un árbol normal una operación de búsqueda puede ser de orden O(n) en el peor caso y de O (1) en el mejor caso. En un árbol desplegado se garantiza que para cualquiera m operaciones tarda a los más O (m Log n) aunque podría ser que una de las m operaciones sea de orden O(n). Pero el análisis amortizado se justifica precisamente cuando se aplican m operaciones.

Si al momento de planificar un árbol binario de búsqueda, sabemos el comportamiento de los accesos y visitas futuras podemos construir un árbol binario de búsqueda óptimo con lo que conseguiremos que la media de gasto generado a la hora de buscar un elemento sea minimizado. En estos casos es conveniente usar una solución basada en la programación dinámica. En cambio cuando no se da este caso, como ocurre en un ABB de palabras usado en un corrector ortográfico, deberíamos balancear el árbol basado en la frecuencia que tiene una palabra en el Cuerpo lingüístico desplazando palabras como "antes" cerca de la raíz y palabras como "versículo" cerca de las hojas. Un árbol como tal podría ser comparado con los árboles de Huffman que tratan de encontrar elementos que son accedidos frecuentemente cerca de la raíz para producir una densa información; de todas maneras, los árboles Huffman sólo pueden guardar elementos que contienen datos en las hojas y estos elementos no necesitan ser ordenados. En cambio, si no sabemos la secuencia en la que los elementos del árbol van a ser accedidos, podemos usar árboles biselados que son tan buenos como cualquier árbol de búsqueda que podemos construir para cualquier secuencia en particular de operaciones de búsqueda.

4. CONCLUSIONES

Un árbol biselado es un árbol binario de búsqueda autobalanceable con la propiedad de que los elementos accedidos recientemente se accederán más rápidamente en accesos posteriores, esto debido a que en cada acceso a un elemento se desplaza hacia la raíz. Es decir se optimiza automáticamente. Esto es una ventaja para casi todas las aplicaciones, y es particularmente útil para implementar cachés y algoritmos de recolección.

Un árbol biselado puede ser de profundidad arbitraria, pero después de cada acceso el árbol se reajusta. El efecto es que cualquier secuencia de m operaciones tarda un tiempo O (m Log n) que es el mismo que tarda en un árbol equilibrado.

Aun cuando los árboles biselados se comportan igual que los AVL, en estos interesa más mantener el equilibrio, mientras que en los primeros no nos preocupamos tanto del costo de un acceso individual, que puede ser tan malo como O(n), en su lugar queremos asegurar que este tipo de comportamiento no puede producirse repetidamente.

5. REFERENCIAS

[1] [AHO 1988] Aho A, Hopcroft J,Ullman J. (1988). *Estructuras de datos y algoritmos.* Addison-Wesley, Wilmington-Delaware EUA.

[2] [HERNANDEZ 2001] Hernández R, Lázaro JC, Dormido R, Ros S. (2001) *Estructura de Datos y Algoritmos*. Prentice Hall. Madrid.

[3] [BRASSARD 1998] Brassard G, Bratley P. (1998). *Fundamentos de algoritmia*, Prentice Hall. Madrid.

[4] [CORTEZ 2002] Cortez Vásquez A. Estructura *de datos y Algoritmos, estructuras no lineales*. (2002). URP Lima.

[5] [CORTEZ 1999] Cortez Vásquez A. (1999). *Matematicas discretas*. Lima.

[6] [GRASSMAN 1996] Grassmann W, Tremblay J.(1996). *Matemática discreta y lógica*. Prentice Hall.

[7] [GRIMALDI 1994] Grimaldi R. (1994). *Matemáticas discreta y combinatoria*"; Addison-Wesley.

[8] [GUTIERREZ 1993] Gutiérrez XF. (1993). *Estructuras de datos, Especificacion, diseño e implementación.* Edición UPC Barcelona.

[9] [JAIME 2002] Jaime, A. (2002). *Estructuras de datos y algoritmos*". Prentice Hall, Bogotá D.C.

[10][JOHNSONBAUGH 1999] Johnsonbaugh R. (1999). *Matemáticas discretas*". Prentice Hall.

[11] [JOYANES 1999] Joyanes Aguilar L. (1999) *Estructura de datos, algoritmos, abstraccion y objetos*"; Mc Graw Hill.

[12] [KOLMAN 1997] Kolman, Busby, Ross "Estructuras de matematicas discretas"; Prentice Hall Mexico 1997.

[13] [LIPSCHUTZ 1987] Lipschutz Seymour "Estructura de datos", Mc Graw-Hill,1987

[14] [PEÑA 1998] Peña Mori Ricardo "Diseño de programas- Formalismo y Abstracción", Prentice Hall 1998.

[15] [CARMONA 1999] Carmona, Poyato Angel y Otros "Estructuta de Datos", Caja Sur Universidad de Cordova España 1999

[16] [SEDGEWICK 1993] Stroustrup Bjarne (1993). *El C++ Lenguaje de programación*. Addison-Wesley, Wilmington-Delaware EUA.

[17] [WEISS 2000a] Weiss, MA (2000). *Estructuras de datos en JAVA*; Addison-Wesley.

[18] [WEISS 2000b] Weiss, Mark Allen. "Estructuras de datos en JAVA"; Addison-Wesley, 2000 .

[19] http://it.ciidit.uanl.mx/"elisa/teching/aa/pdf/clase0210.pdf

[20] http://www.gedlc.ulpgc.es/docencia/ed_ii/temario.html