Act 3.3 - Árbol Desplegado: Conociendo un Splay Tree.

Instituto Tecnológico de Estudios Superiores de Monterrey

Programación de estructura de datos

Valery Moreno Vega

Cristian Ricardo Luque Arámbula - A01741850

Campus Sinaloa

24 de Octubre del 2024

Árboles desplegados

Entre las múltiples presentaciones que tienen las estructuras de datos, donde el enfoque es reducir la complejidad para aplicaciones particulares, un ejemplo destacable son los árboles desplegados. Estos son árboles cuyos métodos básicos siguen una lógica distinta para favorecer implementaciones que requieren utilizar un mismo dato, o datos cercanos en el árbol repetidas veces, ahorrando el proceso de recorrer los nodos para acceder a estos.

¿Qué es un 'Splay tree'?

Un árbol desplegado es una estructura de datos que tiene la misma estructura que un árbol binario de búsqueda (sin adición de atributos), donde los valores del subárbol de la izquierda son menores al nodo padre, y los valores del subárbol de la derecha son mayores a este (Weiss, 2011 p. 134). Este árbol binario no utiliza ninguna regla explícita para imponer su equilibrio, sino que aplica una determinada operación de movimiento a la raíz (denominada despliegue) después de cada acceso, para mantener el árbol de búsqueda 'equilibrado' (en un sentido amortizado) (Goodrich & Tamassia, 2010, p. 450), mas no balanceado (como se realiza con los árboles AVL). Así se garantiza que cualquier operación consecutiva de M árboles que comience desde un árbol vacío tome como máximo O(M logN) tiempo, aunque un solo acceso pueda tomar O(N) (Weiss, 2011, p. 134). La lógica detrás de este es que no hay secuencias malas, y así como hay algunos movimientos costosos, al considerar todas las operaciones se obtiene un promedio amortizado de O (log N).

¿Cómo funciona un árbol desplegado?

Se realizan una serie de rotaciones como las que se llevan a cabo con los árboles AVL, con la diferencia de que en su implementación, no hay tantos casos a considerar antes de realizarla. Además, el criterio de parada de las rotaciones es distinto, puesto que en el árbol desplegado, estas se realizan hasta cambiar la disposición del árbol al punto de que el nodo al que se accede esté posicionado como nodo raíz. Todo esto mientras se mantiene el orden de los elementos en su estructura. Esto implica que cada operación básica realiza un reacomodo automático llamado 'splay' después de acceder al elemento, permitiendo operar con el elemento que se acaba de insertar o buscar en el árbol.

¿Cuáles son las principales operaciones que se implementan?

Splay: Aplicar rotaciones como las que se aplican en los árboles AVL hasta que el nodo al que se accede (o que se acaba de insertar), sea el nodo raíz del árbol. Estas son de tres tipos, zig, zig-zig y zig-zag. Tiene una complejidad O(log N) en todo el proceso de despliegue, haciéndolo distinto a una rotación simple que sube de nivel el nodo uno por uno.

Búsqueda: Cuando se busca una llave k, se sigue la misma lógica que al buscar en un Binary Sorted Tree (se compara con los nodos, si el nodo actual es mayor al que se busca, se traslada al nodo izquierdo, si no, se traslada al nodo derecho). Si en algún punto de la búsqueda se es equivalente el nodo a la llave k proporcionada, se despliega x. Si no se encuentra en ninguno de los nodos, se despliega el padre del nodo externo, y la búsqueda acaba sin éxito.

Inserción: El proceso de inserción es similar al de un árbol binario de búsqueda. Se compara con los nodos, si el nodo actual es mayor al que se busca insertar, se traslada al nodo izquierdo, si no, se traslada al nodo derecho, y si son equivalentes, se descarta la inserción. Esto hasta llegar al nodo hoja, donde se agrega. Después de la inserción de la llave k, se despliega el nodo creado. La figura debajo muestra un ejemplo de inserciones múltiples, en las que primero se inserta acorde a las reglas de inserción del BST, y luego se despliega.

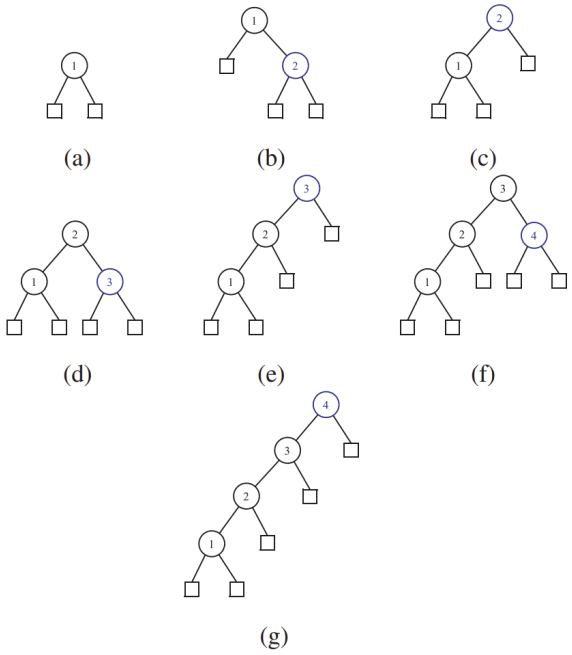

Figure 10.18: A sequence of insertions in a splay tree: (a) initial tree; (b) after inserting 2; (c) after splaying; (d) after inserting 3; (e) after splaying; (f) after inserting 4; (g) after splaying.

Eliminación: Se puede eliminar un elemento al acceder al nodo que se quiere borrar, realizando un despliegue para colocarlo en la raíz. Si se elimina este nodo, se obtienen dos subárboles T_L y T_R (izquierda y derecha, respectivamente), y si se obtiene el elemento más grande del subárbol izquierdo, este nodo es desplegado, posicionándolo como la raíz de este subárbol, dejándolo a su vez sin hijo derecho. La eliminación termina

al hacer que el subárbol derecho sea el hijo derecho del nodo raíz del subárbol izquierdo (Weiss, 2011, p. 144)

¿Cuáles son los tipos de rotación que se utilizan en un árbol desplegado?

Estas operaciones son, como se explicó anteriormente, iguales en esencia a las del árbol AVL (difiriendo en la implementación en el código), y se dividen en tres dependiendo de la posición respecto a la raíz del nodo a desplegar y de su relación con sus nodos padres (si es mayor o menor que ellos).

Zig-zig: El nodo X y su padre Y son ambos hijos izquierdos o derechos de otro nodo Z. En este caso se reemplaza este último por el nodo X al hacer que Y sea su hijo, y el nodo Z sea hijo del nodo Y. Esto se visualiza en el ejemplo debajo (Goodrich & Tamassia, 2010, p. 450)

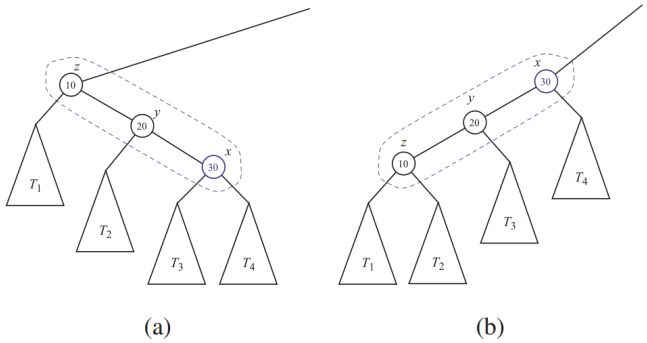

Figure 10.13: Zig-zig: (a) before; (b) after. There is another symmetric configuration where *x* and *y* are left children.

Zig-Zag: Se realiza cuando uno de los nodos X o Y es un hijo izquierdo y el otro es hijo derecho de un nodo Z. Se reemplaza el nodo Z por X, y hacer que tanto Y como Z sean nodos de X mientras se mantiene la relación en los nodos.

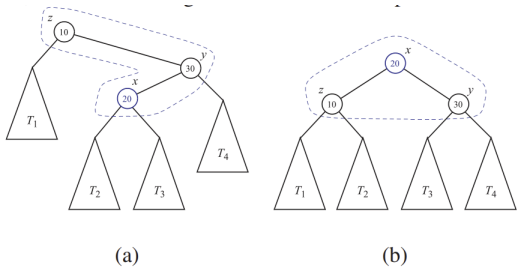

Figure 10.14: Zig-zag: (a) before; (b) after. There is another symmetric configuration where *x* is a right child and *y* is a left child.

Zig: Se realiza cuando el nodo que se busca desplegar se encuentra a un nivel del nodo raíz, o bien, que el nodo X con padre Y, no tiene un nodo abuelo. En este caso, se incorpora el nodo Y como hijo izquierdo del nodo X, colocándolo como raíz del árbol.

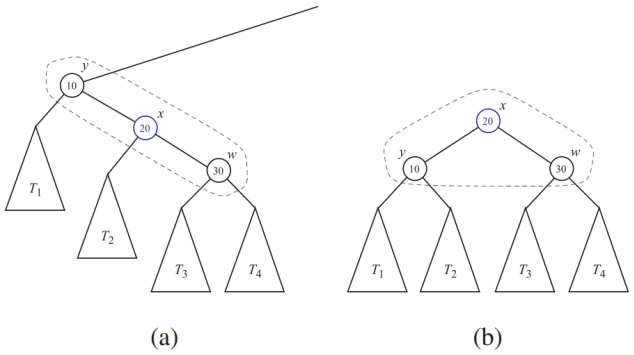

Figure 10.15: Zig: (a) before; (b) after. There is another symmetric configuration where *x* and *w* are left children.

¿Cuáles son sus ventajas y desventajas?

Ventajas: A diferencia de otros tipos de árboles, al acceder a un nodo que está bastante profundo, hay nodos que también se encuentran relativamente profundos y gracias a la reestructuración, se puede lograr que para estos nodos los accesos sean más económicos. Así, y tomando en cuenta que los elementos más populares se mantienen hasta arriba en el árbol, se facilita el acceso a los nodos más recurrentes y accesos secuenciales

de los datos del árbol. Además, no requieren el mantenimiento de información de altura o equilibrio, ahorrando código y espacio.

Desventajas: Muestran inconsistencia en las operaciones individuales, haciendo que operaciones básicas puedan ser O(n). Además, se mantiene el comportamiento base de un BST y aparte se agrega la complejidad del despliegue, lo cual puede ser peligroso si se tienen casos aleatorios uniformes donde no hay garantía de que se acceda a los nodos cercanos, haciendo que la complejidad adicional del despliegue en cada operación resulte en una complejidad promedio peor a un árbol BST normal. Finalmente, no se mantiene balanceado como los árboles AVL, así que en operaciones donde no se sigan patrones en general, es mejor no optar por este tipo de árboles.

¿Para qué tipo de aplicaciones es útil?

Este árbol sigue la filosofía de que si se requiere un dato una vez, es probable que las siguientes operaciones también requieran de este nodo para realizarse. Por ende, es excelente para aplicaciones como sistemas de caché, donde ciertos elementos se acceden con frecuencias mucho más altas que otros. Los elementos más utilizados se reubican cerca de la raíz y se reduce el tiempo de acceso. También en sistemas operativos, donde la administración de memoria es más eficiente puesto que los bloques accedidos de manera reciente suelen ser reutilizados, así que al reorganizarse en la parte superior del árbol, se optimiza el tiempo de acceso y mejorando el rendimiento del sistema. Otra aplicación es en el contexto de redes donde es posible optimizar el proceso del enrutamiento para mantener cercano a la raíz las rutas que se acceden recientemente. Esto es particularmente útil en situaciones donde el tráfico es predecible o concentrado en ciertos destinos. Su

contribución es alta al minimizar los tiempos de búsqueda y al reducir una posible sobrecarga al procesar direcciones frecuentes.

Referencias

Goodrich, M. T., & Tamassia, R. (2010). *Data structures and algorithms in C++* (2.ª ed.). Wiley.

Weiss, M. A. (2011). *Data structures and algorithm analysis in Java* (3.ª ed.). Addison-Wesley.