Apuntes

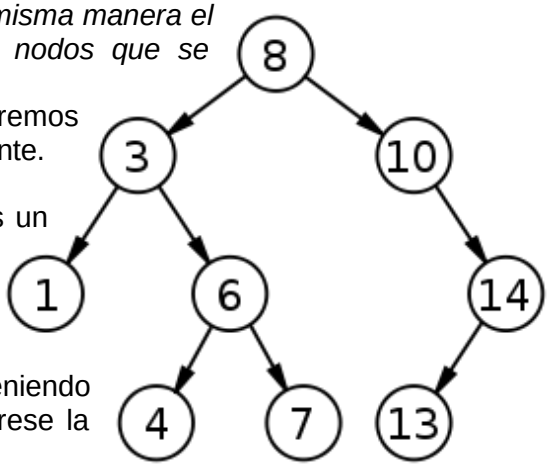
Tema ***Arboles Binarios de Búsqueda***

"La experiencia no es lo que te sucede, sino lo que haces con lo que te sucede."
Aldous Huxley (1894-1963) Novelista, ensayista y poeta inglés.

Arboles binarios de búsqueda (ABB o BST)

¿Qué son?

Un árbol binario de búsqueda también llamados BST (*acrónimo del inglés* ***Binary Search Tree***) es un tipo particular de árbol binario en el que los elementos están organizados de tal manera que facilitan las búsquedas
El árbol depende de un campo **clave** que es distinto para cada nodo y está ordenado respecto a ese campo, tal que si T es un árbol binario, es de búsqueda si cada nodo N del árbol tiene la siguiente propiedad: "*El campo* ***clave de n*** *es mayor que cualquiera de los campos clave de los nodos del subárbol izquierdo de n y de la misma manera el campo clave de n es menor el campo clave de todos los nodos que se encuentran en el subárbol derecho de n"*
Esto supone que si hago un recorrido **inorden** del árbol obtendremos sus elementos ordenados por el campo clave en orden ascendente.



NOTA: Para una fácil comprensión queda resumido en que es un árbol binario que cumple que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores.
Dos árboles binarios de búsqueda *pueden no ser iguales* aun teniendo la misma información, va a depender del orden en el que ingrese la información.

Búsqueda de un elemento

El algoritmo localiza (PTR o dato referencial) el nodo que contiene el valor y asigna PAD al padre del nodo. La filosofía del algoritmo consiste en comenzar comparando el elemento con la raíz del árbol y luego ir recorriendo dicho árbol por la derecha o por la izquierda según si el elemento es mayor o menor que el nodo que estamos examinando y así hasta encontrar el elemento o llegar a una hoja del árbol (indicando que el elemento no se encuentra).

Inserción de un elemento

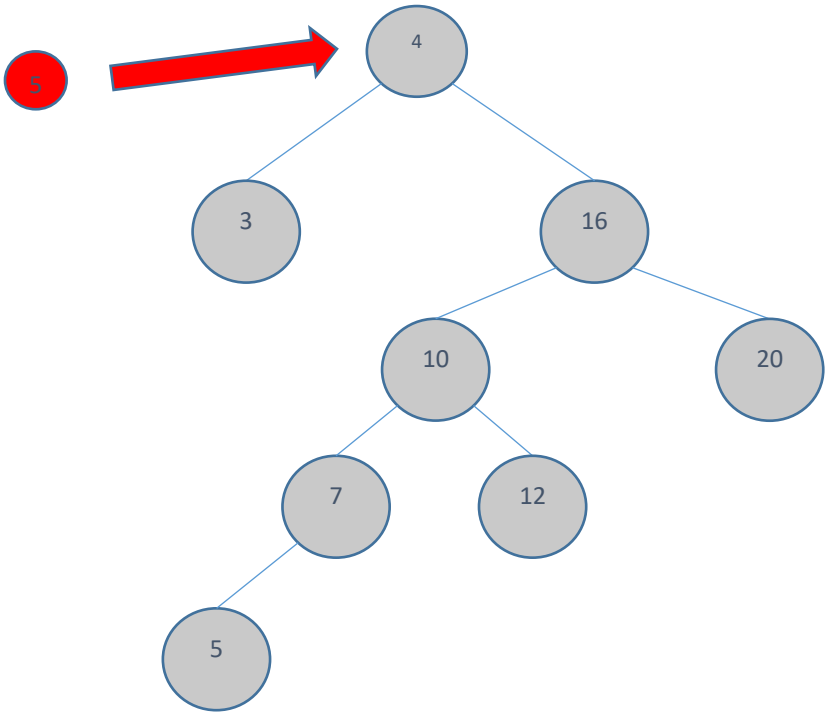
1º Verificar que hay recursos (espacio) disponibles.

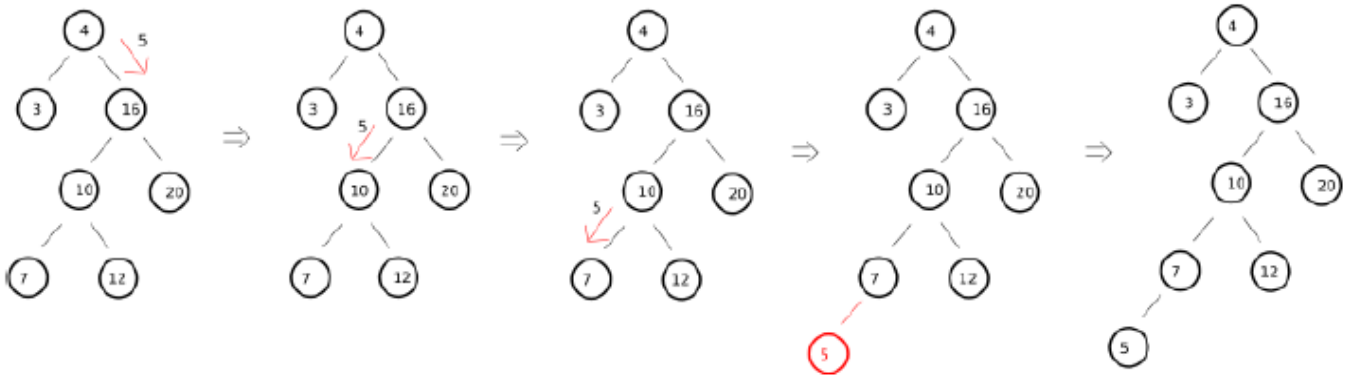
2º Verificar si el elemento está en el árbol y si no está, localizar la posición dónde debe ser insertado, para lo que utilizaré el algoritmo de búsqueda tal que PAD será el nodo al que hay que enganchar el nodo que insertamos.
3º Averiguar si el debe insertarse como el hijo derecho o izquierdo de PAD.
4º Finalmente, el nodo tiene que quedar como una hoja.

Eliminación de un elemento

Con el elemento lo primero que hay que hacer es controlar la situación de ***UNDERFLOW***, después buscar el nodo y una vez encontrado el nodo se pueden dar 3 casos:
1. - que el nodo no tenga hijos: simplemente se borra y se establece a nulo el apuntador de su padre.
2. - que el nodo tenga un hijo: se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.
3. - que tenga dos hijos: la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en **inorden** y posteriormente borrar este nodo. Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subarbol izquierdo), y su sucesor el nodo más a la izquierda de su subárbol derecho (menor nodo del subarbol derecho).

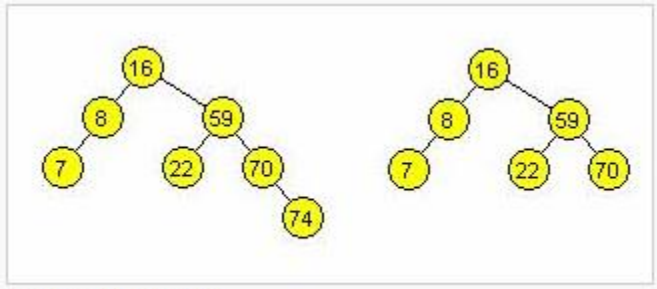
Ejemplo de inserción de un elemento





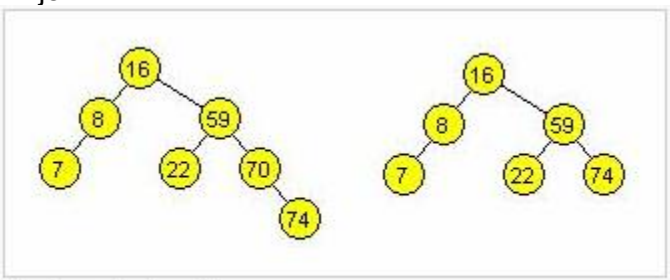
Ejemplo de eliminación de un elemento

- Borrado sin hijos:



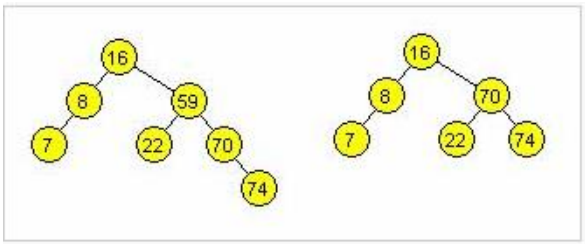
Nodo a eliminar 74

- Borrado con un hijo:

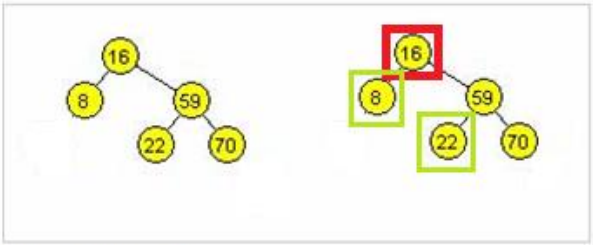


Nodo a eliminar 70

- Borrado con 2 hijos:

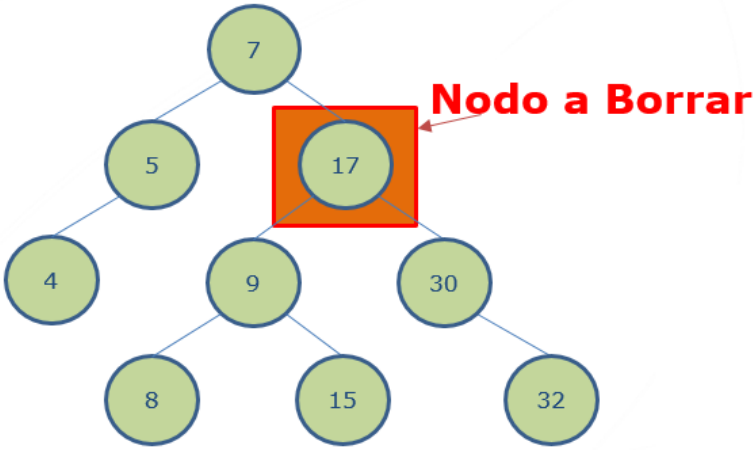


Nodo a eliminar 59

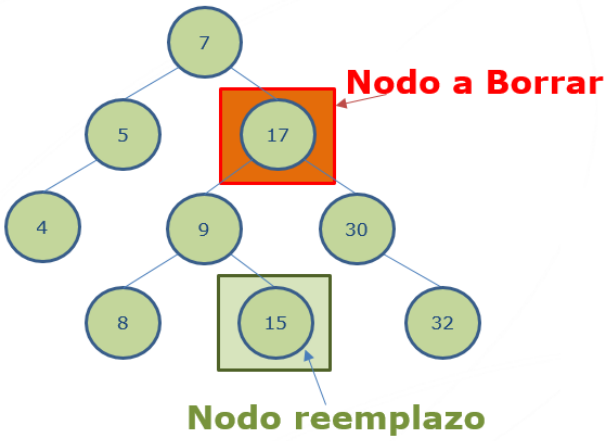


Nodo a eliminar **16**

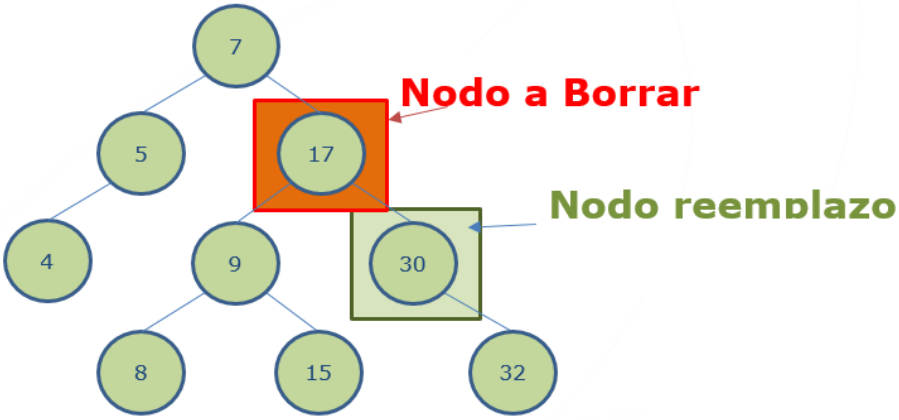
Simulemos la eliminación de un nodo con 2 hijos



Opción 1: reemplazar el valor del nodo por el de su **predecesor** en **inorden** y posteriormente borrar este nodo. Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subarbol izquierdo.

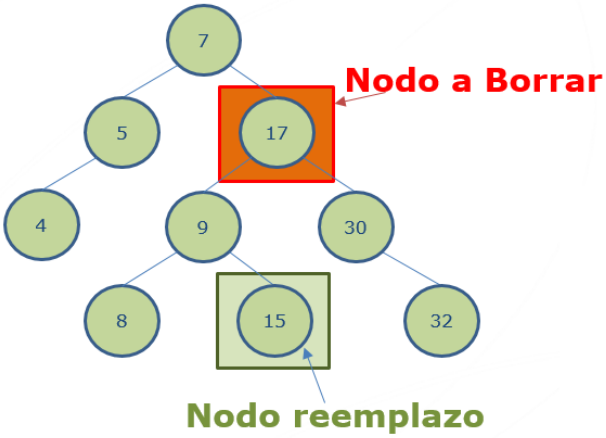


Opción 2: reemplazar el valor del nodo por el de su **sucesor** en **inorden** y posteriormente borrar este nodo. Su sucesor es el nodo más a la izquierda de su subárbol derecho (menor nodo del subarbol derecho).

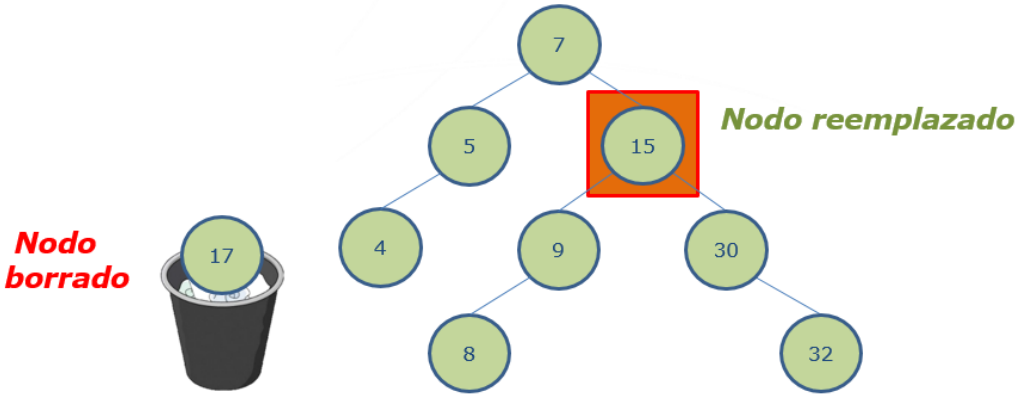


Nota: Tomando cualquiera de las opciones, pueden darse 2 casos. **Que No tenga hijos** o **que tenga 1 hijo**.

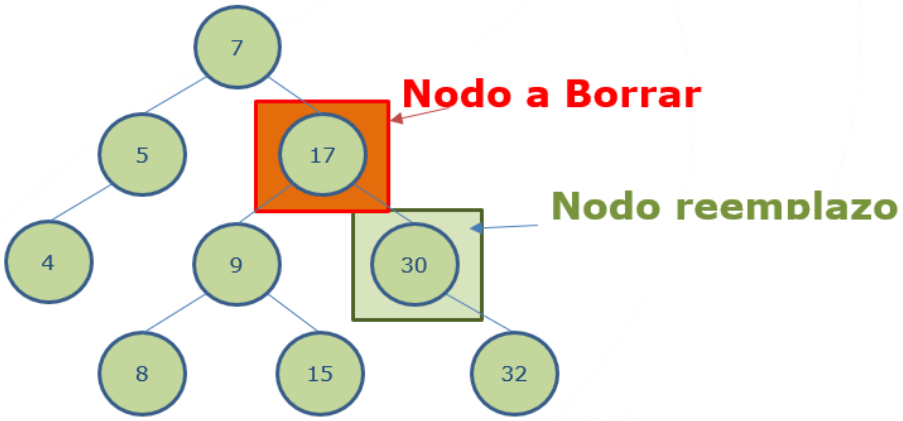
Tomando la opción 1, en este caso sin hijos



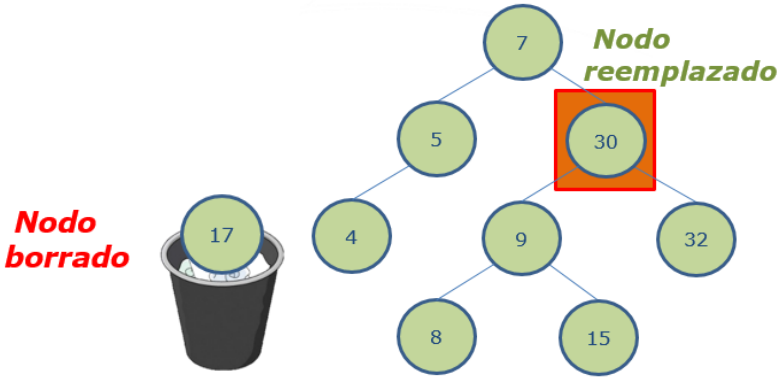
1) Al nodo padre del nodo reemplazo se le elimina la referencia (izquierda o derecha), en este caso la derecha.
2) Al nodo padre del nodo a borrar se lo apunta al nodo reemplazo.
3) Al nodo reemplazo se ponen las referencias de hijos del nodo a borrar.
4) Se borra el nodo.



Tomando la opción 2, en este caso con hijo.



1) Al nodo padre del nodo reemplazo se le elimina la referencia (izquierda o derecha), en este caso no se hace porque el padre es el nodo a borrar.
2) Al nodo padre del nodo a borrar se lo apunta al nodo reemplazo.
3) 3) Al nodo reemplazo se ponen las referencias del sub árbol contrario como hijo. En este caso el sub árbol izquierdo se pone como hijo izquierdo.
4) 4) Al nodo reemplazo se ponen las referencias del sub árbol al que pertenece en su el extremo de su sub subárbol. En este caso por ser subárbol derecho se debería agregar al nodo mas a la derecha, pero como como el nodo reemplazo era hijo del nodo borrado, no se hace nada.
5) 5) Se borra el nodo.



Aplicaciones de los árboles binarios

- Reconocimiento de patrones: Una de las aplicaciones más importantes de los árboles es utilizarla en el reconocimiento de patrones que es una de las ramas de la inteligencia artificial y en general en la mayoría de las aplicaciones en que se necesita hacer búsquedas.

  La filosofía de reconocimiento de patrones es llegar a una conclusión al ir moviéndonos en el árbol según unas determinadas pistas.
  Se pueden representar muchos patrones en forma de árboles binarios. El truco está en asignar los significados adecuados a los movimientos de ir hacia la izquierda o la derecha.
  El inconveniente es que como lo que usamos son árboles binarios, estas relaciones de búsqueda o patrones sólo pueden ser binarias.

- Transformación de una notación en otra: La transformación de una notación a otra se puede realizar estructurando los exponentes como un árbol binario tal que un recorrido sea notación infija, un recorrido preorden la notación prefija y uno postorden la postfija.

  La primera tarea es construir el árbol teniendo en cuenta que cada subárbol tendrá como raíz un operador y los hijos o son operados u otros subárboles.
  Para equilibrar 0 + a (b + c * d)

Árbol binario auto balanceado

En ciencias de la computación, un ***árbol binario de búsqueda auto-balanceable*** o autoequilibrado es un árbol binario de búsqueda que *intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento, automáticamente*. Esto es importante, ya que *muchas operaciones en un árbol de búsqueda binaria tardan un tiempo proporcional a la altura del árbol*, y los árboles binarios de búsqueda

ordinarios pueden tomar alturas muy grandes en situaciones normales, *como cuando las claves son insertadas en orden*. Mantener baja la altura se consigue habitualmente realizando transformaciones en el árbol, como la rotación de árboles, en momentos clave.
Tiempos para varias operaciones en términos del número de nodos en el árbol n:

Operación	Tiempo en cota superior asintótica
Búsqueda	O(log *n*)
Inserción	O(log *n*)
Eliminación	O(log *n*)
Iteración en orden	O(*n*)

Criterios de equilibrio (o de balanceo)