2.1.7. Árbol Binario de Búsqueda (ABB).

1. Definición. Se trata de árboles de orden 2 en los que se cumple que, para cada nodo, el valor de la clave de la raíz del subárbol izquierdo es menor que el valor de la clave del nodo y que el valor de la clave raíz del subárbol derecho es mayor que el valor de la clave del nodo (Figura 25).

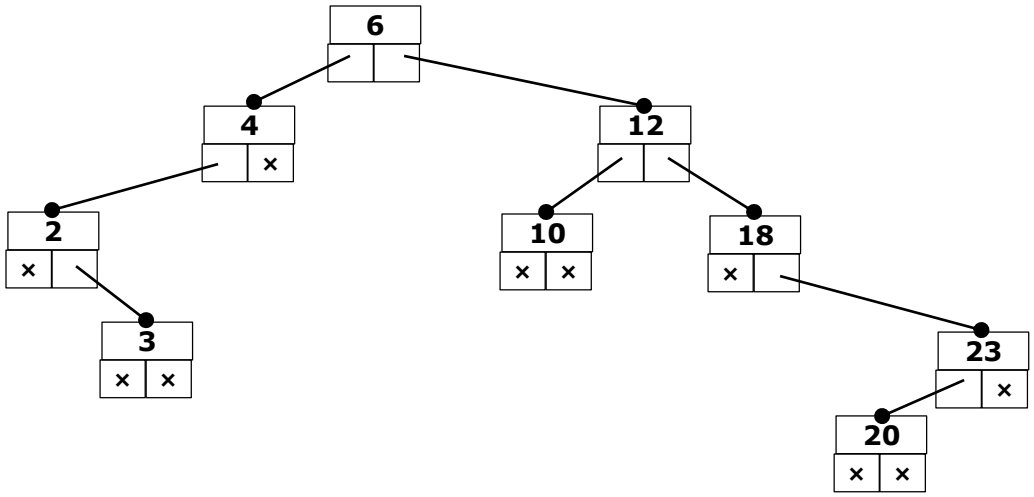

Figura 25

son estructuras de datos que presentan un gran rendimiento cuando las funciones a realizar implican búsquedas, inserciones y eliminación de nodos. De hecho, con un árbol de búsqueda, dichas operaciones se pueden realizar tanto a partir de un valor clave,

Definición: Un árbol binario de búsqueda es un árbol binario, que puede estar vacío, y que si es no vacío cumple las siguientes propiedades:
(1) Todos los nodos están identificados por una clave y no existen dos elementos con la misma clave.
(2) Las claves de los nodos del subárbol izquierdo son menores que la clave del nodo raíz.
(3) Las claves de los nodos del subárbol derecho son mayores que la clave del nodo raíz.
(4) Los subárboles izquierdo y derecho son también árboles binarios de búsqueda.

La primera propiedad resultaría redundante, ya que de las propiedades (2), (3) y (4) se puede deducir que la clave de un nodo es única. Sin embargo, la aparición explícita de esta propiedad hace que la definición sea más clara.

2. Operaciones.

2.1. Búsqueda: La definición de árbol binario de búsqueda especifica un criterio en la estructuración del árbol en función de las claves de los nodos. En este caso, existe un criterio de ordenación de los nodos. Por lo tanto, es factible describir un método eficiente de búsqueda que explote esa ordenación.

- Suponer que se busca un elemento que posea una clave x.
- La búsqueda comenzará por el nodo raíz del árbol. La clave de ese nodo informará por dónde debe continuar la búsqueda, ya no es necesario recorrer exhaustivamente todos los nodos del árbol.
- Si la clave del nodo es igual a x, la búsqueda finaliza con éxito.
- Si la clave es menor que x, se sabe que si existe un nodo en el árbol que posea como clave el valor x deberá estar en el subárbol derecho, por lo tanto, la búsqueda deberá continuar por esa parte del árbol.
- Si, por el contrario, la clave del nodo es mayor que x, entonces la búsqueda deberá continuar por el subárbol izquierdo.

El proceso continuará hasta que se encuentre un nodo con clave igual a x o un subárbol vacío, en cuyo caso se puede asegurar que no existe ningún nodo con clave x en el árbol. Este método de búsqueda sugiere seguir un esquema recursivo.

Suponiendo que la clave que identifica cada nodo es un campo contenido en la información general del nodo, el algoritmo de búsqueda quedaría como sigue:

```
Algoritmo Buscar_Recurrente_ABB
{* Buscar en el árbol arb la clave x                                    *}
{* Devolver la posición del nodo donde se encuentre, NULO si no está*}

Entradas
        Valor(clave) x
        ArbolBinario arb

Salida
        ArbolBinario

Inicio
        Si(1) (arb es un árbol vacío) entonces
                Devolver (Árbol vacío)
        Sino(1)
                Si(2) (x = Informacion(clave) de arb) entonces
                        Devolver (arb)
                Sino(2)
                        Si(3) (x < Informacion(claver) de arb) entonces
                                Devolver (Buscar_Recurrente_ABB (x, Hijo izquierdo de arb) )
                        Sino(3)
                                Devolver (Buscar_Recurrente_ABB (x, Hijo derecho de arb) )
                        Fin_si(3)
                Fin_si(2)
        Fin_si(1)
Fin
```

En este ejemplo, la recursividad puede ser fácilmente sustituida por un esquema iterativo mediante la utilización de un bucle de repetición mientras. En ese caso, el algoritmo de búsqueda iterativo sería:

```
Algoritmo Buscar_Iterativo_ABB
{ Buscar en el árbol arb la clave x                                    }
{ Devolver la posición del nodo donde se encuentre, NULO si no está }

Entradas
        Valor(clave) x
        ArbolBinario arb

Salida
        ArbolBinario

Variables
        ArbolBinario aux
        Boolean encontrado: (CIERTO, FALSO)

Inicio
        aux ← arb
        encontrado ← falso
        Mientras (aux no sea árbol vacío) y (NO encontrado) hacer


                Si(1) (x = informacion (clave) de aux) entonces
                        encontrado ← CIERTO
                Sino(1)
                        Si(2) (x < Informacion (clave) de donde) entonces
                                aux ← Hijo izquierdo de aux
                        Sino(2)
                                aux ← Hijo derecho de aux
                        Fin_si(2)
                Fin_si(1)
        Fin_mientras

        Si(3) (encontrado) entonces
                Devolver (aux)
        Sino(3)
                Devolver (Árbol vacío)
        Fin_si(3)
Fin
```

El proceso de búsqueda en un árbol binario de búsqueda resulta muy eficaz. El coste asociado sería O(h), donde h es la altura del árbol. Hay que hacer notar que la dependencia es lineal con la altura del árbol y no con el número de elementos del mismo. En función del número de nodos en el árbol, n, el coste sería O(log n) (Esto es cierto si el árbol está suficientemente equilibrado. En el peor de los casos el árbol binario de búsqueda degeneraría en una lista y el coste llegaría a ser lineal.)

El método de búsqueda se asemeja mucho al método de búsqueda binaria sobre arrays ordenados, tras la comparación con un elemento se puede decidir en qué región de la estructura se puede encontrar la información buscada, descartándose el resto de los elementos de la estructura. De esta forma, se reduce considerablemente el número de comparaciones necesarias para localizar el elemento.

2.2. Insertar un elemento. Para insertar un elemento nos basamos en el algoritmo de búsqueda. Si el elemento está en el árbol no lo insertaremos. Si no lo está, lo insertaremos a continuación del último nodo visitado.

Necesitamos un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL.

- Padre = NULL
- nodo = Raiz
- Bucle: mientras actual no sea un árbol vacío o hasta que se encuentre el elemento.
    - Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo: Padre=nodo, nodo=nodo->izquierdo.
    - Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho: Padre=nodo, nodo=nodo->derecho.
- Si nodo no es NULL, el elemento está en el árbol, por lo tanto, salimos.
- Si Padre es NULL, el árbol estaba vacío, por lo tanto, el nuevo árbol sólo contendrá el nuevo elemento, que será la raíz del árbol.
- Si el elemento es menor que el Padre, entonces insertamos el nuevo elemento como un nuevo árbol izquierdo de Padre.
- Si el elemento es mayor que el Padre, entonces insertamos el nuevo elemento como un nuevo árbol derecho de Padre.

Este modo de actuar asegura que el árbol sigue siendo ABB. Como se muestra en la Figura 26.

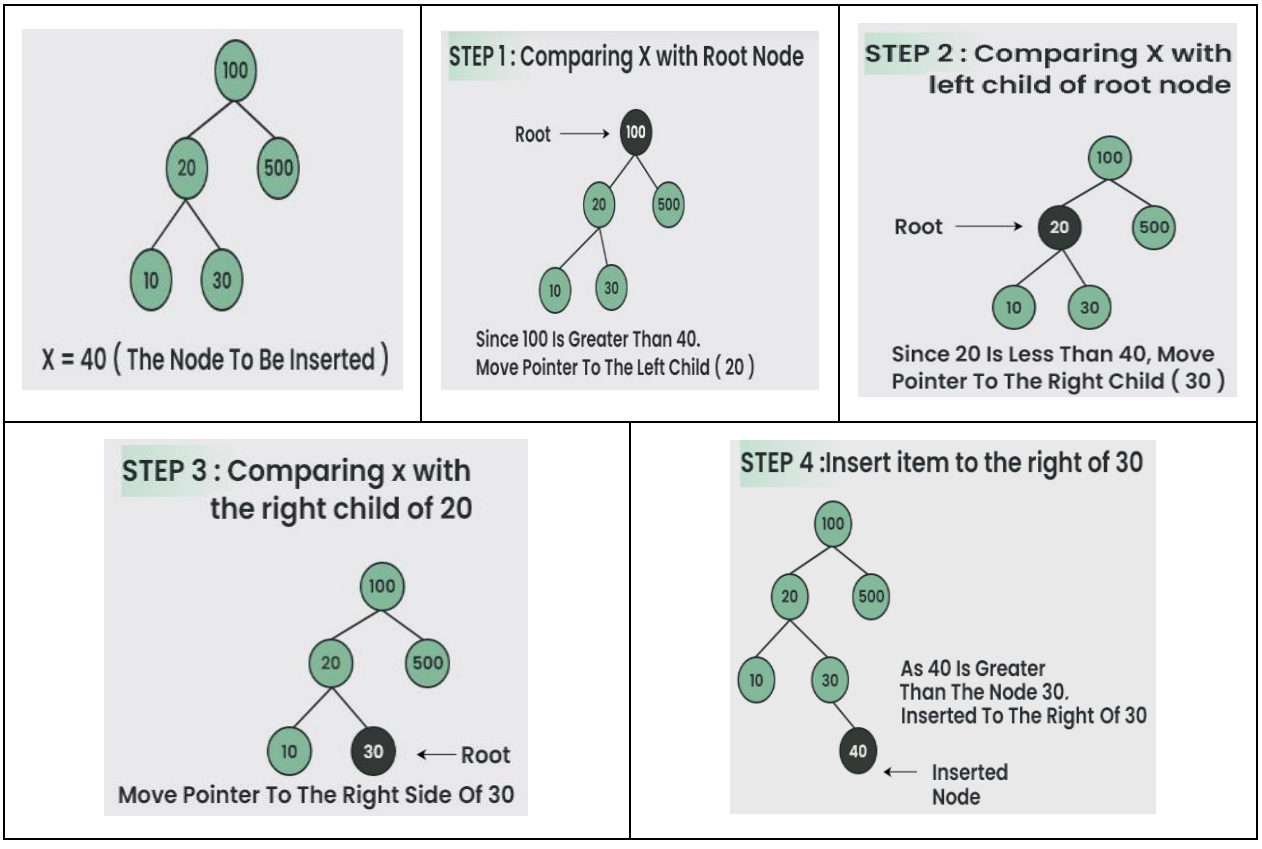

Figura 26

2.3. Borrar un elemento. Para borrar un elemento también nos basamos en el algoritmo de búsqueda. Si el elemento no está en el árbol no lo podremos borrar. Si está, hay dos casos posibles:

a. Se trata de un nodo hoja: en ese caso lo borraremos directamente.
b. Se trata de un nodo rama: en ese caso no podemos eliminarlo, puesto que perderíamos todos los elementos del árbol de que el nodo actual es padre. En su lugar buscamos el nodo más a la izquierda del subárbol derecho, o el más a la derecha del subárbol izquierdo e intercambiamos sus valores. A continuación eliminamos el nodo hoja.

Necesitamos un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL.

- Padre = NULL
- Si el árbol está vacío: el elemento no está en el árbol, por lo tanto salimos sin eliminar ningún elemento.
- **(1)** Si el valor del nodo raíz es igual que el del elemento que buscamos, estamos ante uno de los siguientes casos:
    - El nodo raíz es un nodo hoja:
        - Si 'Padre' es NULL, el nodo raíz es el único del árbol, por lo tanto el puntero al árbol debe ser NULL.
        - Si raíz es la rama derecha de 'Padre', hacemos que esa rama apunte a NULL.
        - Si raíz es la rama izquierda de 'Padre', hacemos que esa rama apunte a NULL.
        - Eliminamos el nodo, y salimos.
    - El nodo no es un nodo hoja:
        - Buscamos el 'nodo' más a la izquierda del árbol derecho de raíz o el más a la derecha del árbol izquierdo. Hay que tener en cuenta que puede que sólo exista uno de esos árboles. Al mismo tiempo, actualizamos 'Padre' para que apunte al padre de 'nodo'.
        - Intercambiamos los elementos de los nodos raíz y 'nodo'.
        - Borramos el nodo 'nodo'. Esto significa volver a **(1)**, ya que puede suceder que 'nodo' no sea un nodo hoja.
- Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo.
- Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho.

Ejemplo caso 1: Borrar un nodo hoja. En el árbol de ejemplo (Figura 25), borrar el **nodo 3**.

Paso 1: Localizamos el nodo a borrar, al tiempo que mantenemos un puntero a 'Padre'.

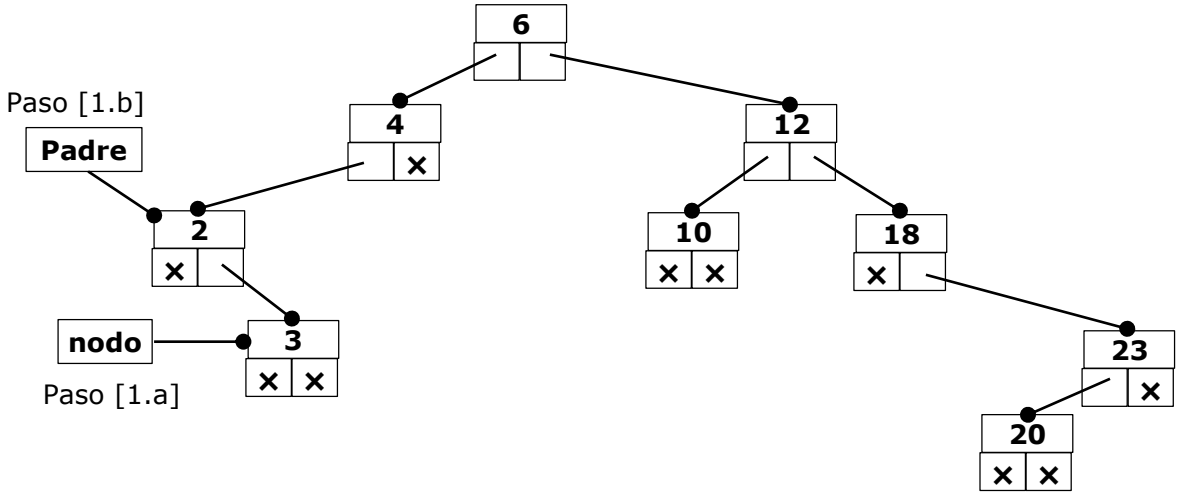

Figura 27-A

Paso 2: Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.

Paso 3: Borramos el 'nodo'.

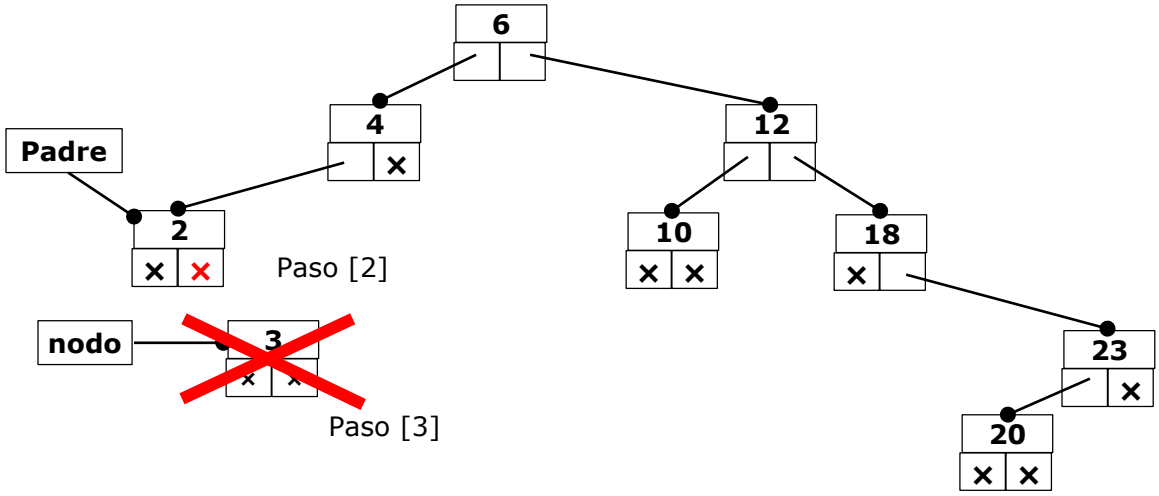

Figura 27-B

Ejemplo caso 2: Borrar un nodo rama con intercambio de un nodo hoja. En el árbol de ejemplo, tal como aparece en su estado original (Figura 25), borrar el **nodo 4**

Paso 1: Localizamos el nodo a borrar ('raíz').

Paso 2: Buscamos el nodo más a la derecha del árbol izquierdo de 'raíz', en este caso el 3, al tiempo que mantenemos un puntero a 'Padre' a 'nodo'.

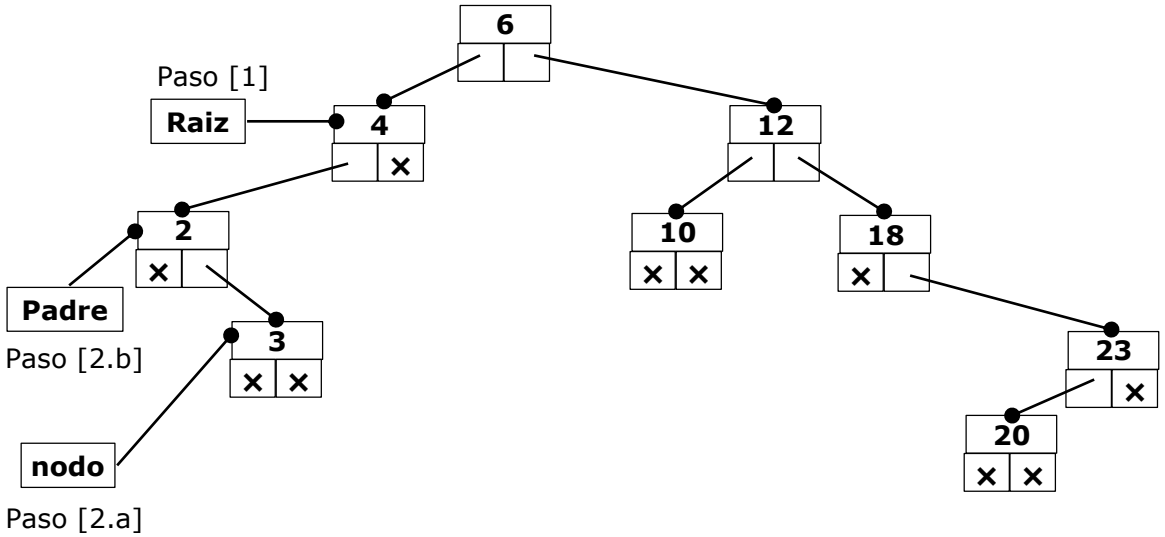

Figura 28-A

Paso 3: Intercambiamos los elementos 3 y 4.

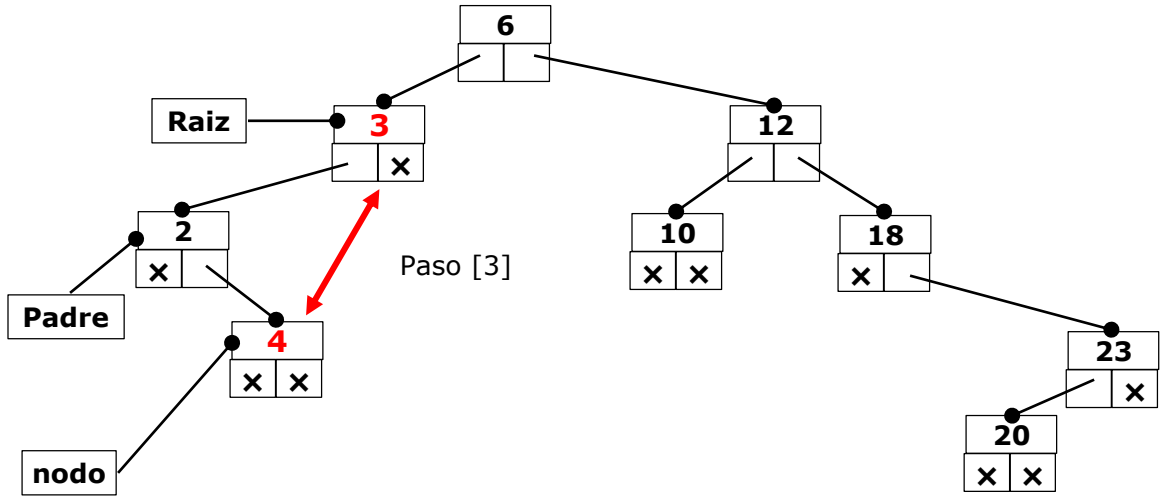

Figura 28-B

Paso 4: Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.

Paso 5: Borramos el 'nodo'.

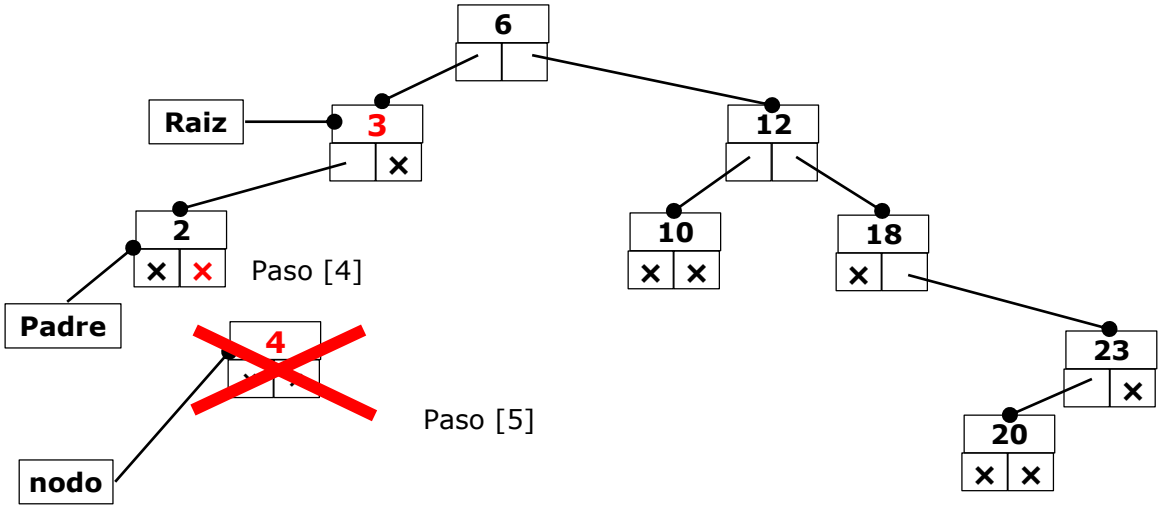

Figura 28-C

Ejemplo caso 3: Borrar un nodo rama con intercambio de un nodo rama. Para este ejemplo usaremos otro árbol. En éste borraremos el **elemento 6**.

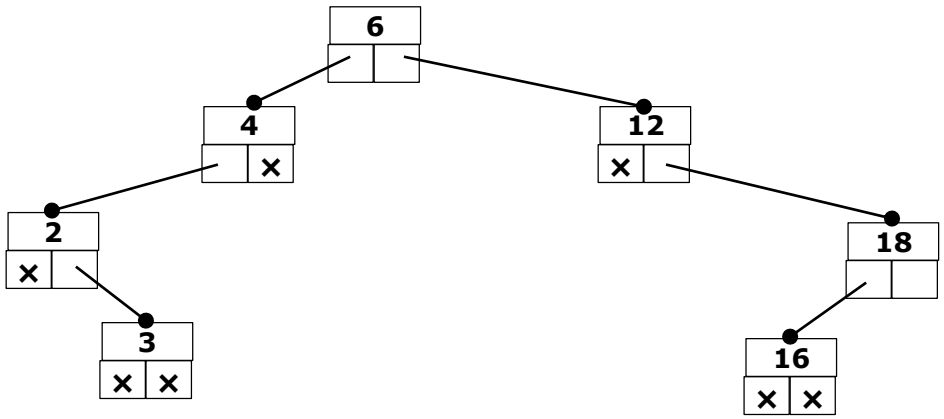

Figura 29

Paso 1: Localizamos el nodo a borrar ('raíz').

Paso 2: Buscamos el nodo más a la izquierda del árbol derecho de 'raíz', en este caso el 12, ya que el árbol derecho no tiene nodos a su izquierda, si optamos por la rama izquierda, estaremos en un caso análogo. Al mismo tiempo que mantenemos un puntero a 'Padre' a 'nodo'.

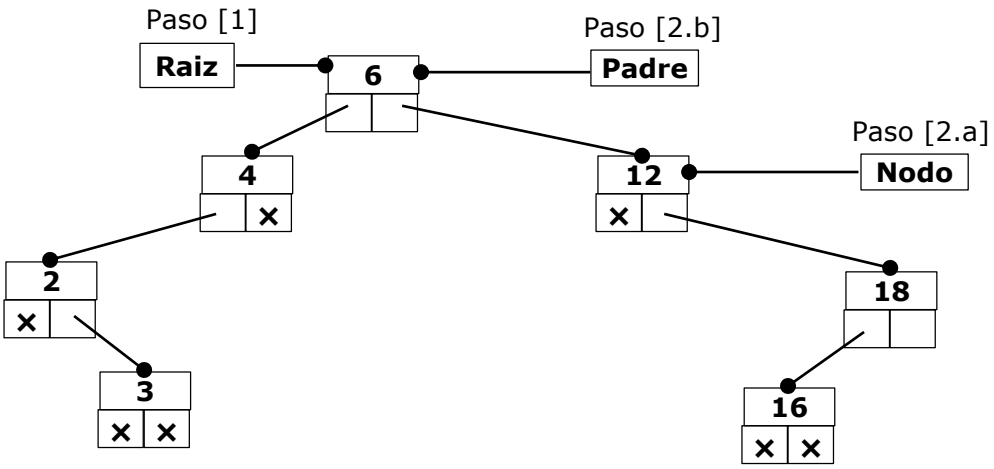

Figura 30-A

Paso 3: Intercambiamos los elementos 6 y 12.

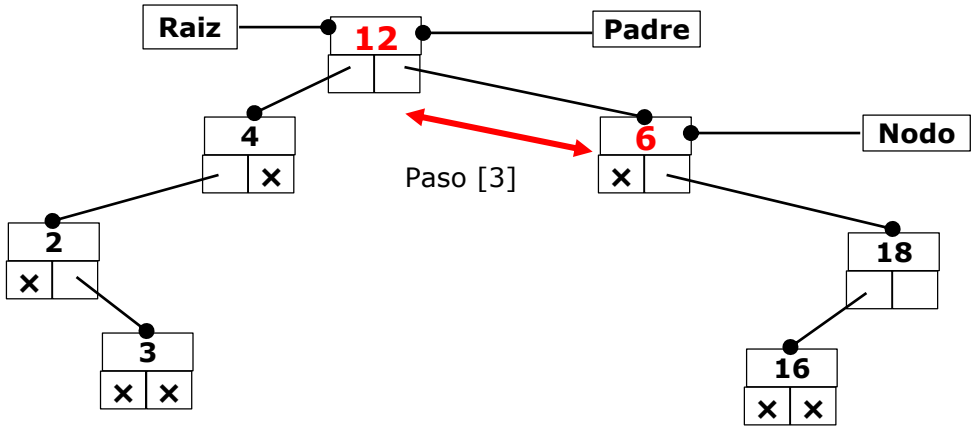

Figura 30-B

Paso 4: Ahora tenemos que repetir el bucle para el nodo 6 de nuevo, ya que no podemos eliminarlo.

Paso 5: Localizamos de nuevo el nodo a borrar ('raíz').

Paso 6: Buscamos el nodo más a la izquierda del árbol derecho de 'raíz', en este caso el 16, al mismo tiempo que mantenemos un puntero a 'Padre' a 'nodo'.

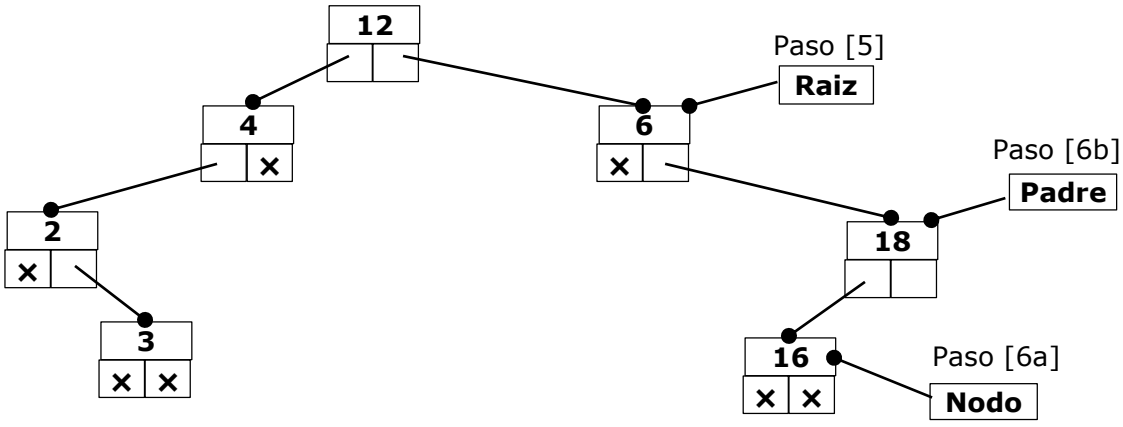

Figura 30-C

Paso 7: Intercambiamos los elementos 6 y 16.

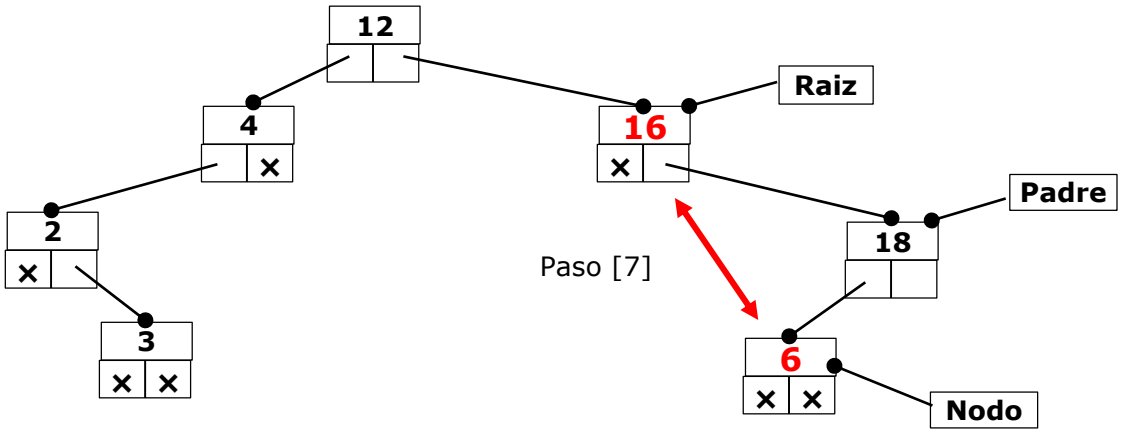

Figura 30-D

Paso 8: Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.

Paso 9: Borramos el 'nodo'.

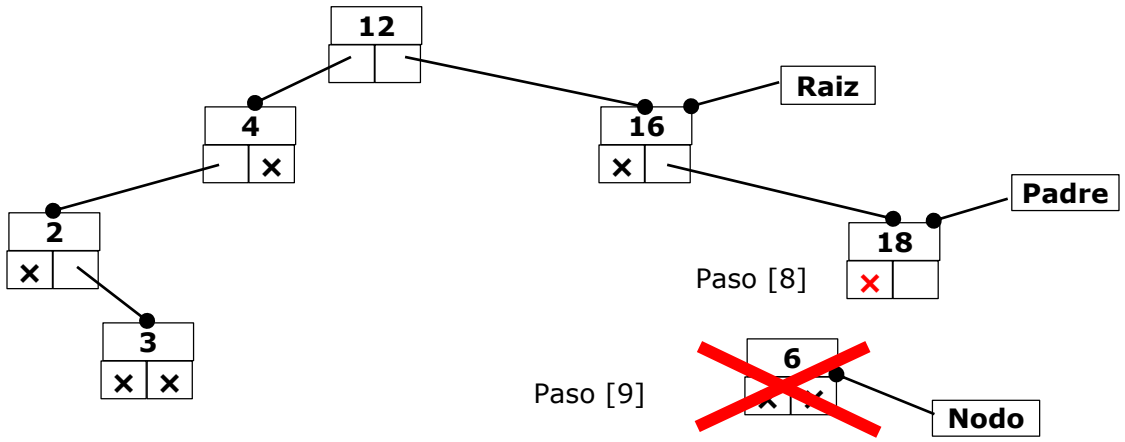

Figura 30-E

Este modo de actuar asegura que el árbol sigue siendo ABB.

6. Movimiento e Información. Con respecto al movimiento o navegación a través del árbol, esta estructura se referenciará siempre mediante un puntero al nodo Raíz, este puntero no debe perderse nunca.

Para moverse a través del árbol se usan punteros auxiliares, de modo que desde cualquier puntero los movimientos posibles serán: moverse al nodo raíz de la rama izquierda, moverse al nodo raíz de la rama derecha o moverse al nodo Raíz del árbol. En cuanto a la información, hay varios parámetros que se pueden calcular dentro de un árbol. Algunos de ellos darán idea de lo eficientemente que está organizado o el modo en que funciona.

INFORMACIÓN REQUERIDA	CÓMO OBTENERLA
Comprobar si un árbol está vacío.	Un árbol está vacío si su raíz es NULL.
Calcular el número de nodos.	Tenemos dos opciones para hacer esto, una es llevar siempre la cuenta de nodos en el árbol al mismo tiempo que se añaden o eliminan elementos. La otra es, sencillamente, contarlos. Para ello se recurre a cualquiera de los tres modos de recorrer el árbol: InOrden, PreOrden o PostOrden, como acción sencillamente incrementamos el contador.
Comprobar si el nodo es hoja.	Basta con comprobar si tanto el árbol izquierdo como el derecho están vacíos. Si ambos lo están, se trata de un nodo hoja.
Calcular la altura de un nodo.	No hay un modo directo de hacer esto, ya que no es posible recorrer el árbol en la dirección de la raíz. De modo que tendremos que recurrir a otra técnica para calcular la altura. Lo que haremos es buscar el elemento del nodo de que queremos averiguar la altura. Cada vez que avancemos un nodo incrementamos la variable que contendrá la altura del nodo. • Empezamos con el nodo raíz apuntando a Raíz, y la 'Altura' igual a cero. • Si el valor del nodo raíz es igual al elemento que buscamos, terminamos la búsqueda y el valor de la altura es 'Altura'. • En caso contrario, incrementamos 'Altura'. • Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo. • Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho.
Calcular la altura de un árbol.	La altura del árbol es la altura del nodo de mayor altura. Será necesario recorrer todo el árbol; de nuevo es indiferente el tipo de recorrido. Cada vez que cambiemos de nivel incrementamos la variable que contiene la altura del nodo actual, y cuando lleguemos a un nodo hoja compararemos su altura con la variable que contiene la altura del árbol. Si es mayor, actualizamos la altura del árbol. • Iniciamos un recorrido del árbol en PostOrden, con la variable de altura igual a cero. • Cada vez que empecemos a recorrer una nueva rama, incrementar la altura para ese nodo. • Después de procesar las dos ramas, verificamos si la altura del nodo es mayor que la variable que almacena la altura actual del árbol, si es así, actualizamos esa variable.
Recorrer un árbol: • PreOrden • InOrden • PostOrden	Pueden hacerse recursivamente reflejando el principio de cada recorrido: *PreOrden:* Acceder al Nodo Raíz Subárbol Izquierdo (y se aplica PreOrden con recursividad) Subárbol Derecho (y se aplica PreOrden con recursividad) *InOrden:* Subárbol Izquierdo (y se aplica InOrden con recursividad) Acceder al Nodo Raíz Subárbol Derecho (y se aplica InOrden con recursividad) *PostOrden:* Subárbol Izquierdo (y se aplica PostOrden con recursividad) Subárbol Derecho (y se aplica PostOrden con recursividad) Acceder al Nodo Raíz

---------- FIN DE ESTA SECCIÓN