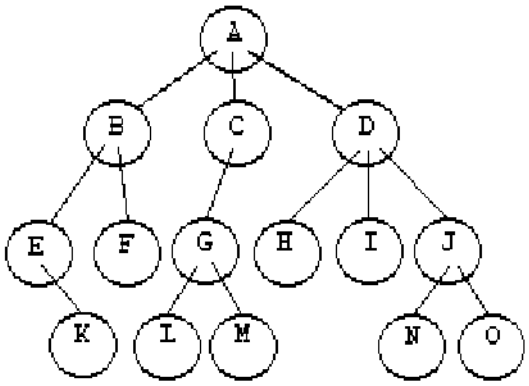
Capítulo 6 Árboles:

6.1 Definición

Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos.

También se suele dar una definición recursiva: un árbol es una estructura en compuesta por un dato y varios árboles.

Esto son definiciones simples. Pero las características que implican no lo son tanto.



Definiremos varios conceptos. En relación con otros nodos:

- **Nodo hijo:** cualquiera de los nodos apuntados por uno de los nodos del árbol. En el ejemplo, 'L' y 'M' son hijos de 'G'.
- **Nodo padre:** nodo que contiene un puntero al nodo actual. En el ejemplo, el nodo 'A' es padre de 'B', 'C' y 'D'.

Los árboles con los que trabajaremos tienen otra característica importante: cada nodo sólo puede ser apuntado por otro nodo, es decir, cada nodo sólo tendrá un padre. Esto hace que estos árboles estén fuertemente jerarquizados, y es lo que en realidad les da la apariencia de árboles.

En cuanto a la posición dentro del árbol:

- **Nodo raíz:** nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol. En el ejemplo, ese nodo es el 'A'.
- **Nodo hoja:** nodo que no tiene hijos. En el ejemplo hay varios: 'F', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.
- **Nodo rama:** aunque esta definición apenas la usaremos, estos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'G' y 'J'.

Otra característica que normalmente tendrán nuestros árboles es que todos los nodos contengan el mismo número de punteros, es decir, usaremos la misma

estructura para todos los nodos del árbol. Esto hace que la estructura sea más sencilla, y por lo tanto también los programas para trabajar con ellos.

Tampoco es necesario que todos los nodos hijos de un nodo concreto existan. Es decir, que pueden usarse todos, algunos o ninguno de los punteros de cada nodo.

Un árbol en el que en cada nodo o bien todos o ninguno de los hijos existe, se llama **árbol completo**.

En una cosa, los árboles se parecen al resto de las estructuras que hemos visto: dado un nodo cualquiera de la estructura, podemos considerarlo como una estructura independiente. Es decir, un nodo cualquiera puede ser considerado como la raíz de un árbol completo.

Existen otros conceptos que definen las características del árbol, con relación a su tamaño:

- **Orden**: es el numero potencial de hijos que puede tener cada elemento de árbol. De este modo, diremos que un árbol en el que cada nodo puede apuntar a otros dos es de *orden dos*, si puede apuntar a tres será de *orden tres*, etc.
- **Grado**: el número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es tres, ya que tanto 'A' como 'D' tienen tres hijos, y no existen elementos con más de tres hijos.
- **Nivel**: se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raiz es cero y el de sus hijos uno. Así sucesivamente. En el ejemplo, el nodo 'D' tiene nivel 1, el nodo 'G' tiene nivel 2, y el nodo 'N', nivel 3.
- **Altura**: la altura de un árbol se define como el nivel del nodo de mayor nivel. Como cada nodo de un árbol puede considerarse a su vez como la raíz de un árbol, también podemos hablar de altura de ramas. El árbol del ejemplo tiene altura 3, la rama 'B' tiene altura 2, la rama 'G' tiene altura 1, la 'H' cero, etc.

Los árboles de orden dos son bastante especiales, de hecho les dedicaremos varios capítulos. Estos árboles se conocen también como **árboles binarios**.

Frecuentemente, aunque tampoco es estrictamente necesario, para hacer más fácil moverse a través del árbol, añadiremos un puntero a cada nodo que apunte al nodo padre. De este modo podremos avanzar en dirección a la raíz, y no sólo hacia las hojas.

Es importante conservar siempre el nodo *raíz* ya que es el nodo a partir del cual se desarrolla el árbol, si perdemos este nodo, perderemos el acceso a todo el árbol.

El nodo típico de un árbol difiere de los nodos que hemos visto hasta ahora para listas, aunque sólo en el número de nodos. Veamos un ejemplo de nodo para crear árboles de orden tres:

```
struct nodo {
```

```
    int dato;
    struct nodo *rama1;
    struct nodo *rama2;
    struct nodo *rama3;
};
```

O generalizando más:

```
#define ORDEN 5

struct nodo {
    int dato;
    struct nodo *rama[ORDEN];
};
```

6.2 Declaraciones de tipos para manejar árboles en C:

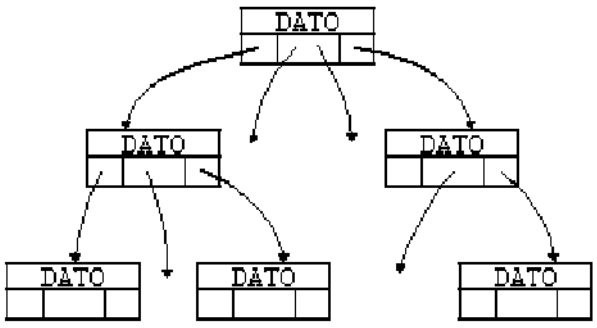
Para C, y basándonos en la declaración de nodo que hemos visto más arriba, trabajaremos con los siguientes tipos:

```
typedef struct _nodo {
    int dato;
    struct _nodo *rama[ORDEN];
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Arbol;
```

Al igual que hicimos con las listas que hemos visto hasta ahora, declaramos un tipo *tipoNodo* para declarar nodos, y un tipo *pNodo* para es el tipo para declarar punteros a un nodo.

Arbol es el tipo para declarar árboles de orden *ORDEN*.



El movimiento a través de árboles, salvo que implementemos punteros al nodo padre, será siempre partiendo del nodo raíz hacia un nodo hoja. Cada vez que lleguemos a un nuevo nodo podremos optar por cualquiera de los nodos a los que apunta para avanzar al siguiente nodo.

En general, intentaremos que exista algún significado asociado a cada uno de los punteros dentro de cada nodo, los árboles que estamos viendo son abstractos, pero las aplicaciones no tienen por qué serlo. Un ejemplo de estructura en árbol es el sistema de directorios y ficheros de un sistema operativo. Aunque en este caso se trata de árboles con nodos de dos tipos, nodos directorio y nodos fichero, podríamos considerar que los nodos hoja son ficheros y los nodos rama son directorios.

Otro ejemplo podría ser la tabla de contenido de un libro, por ejemplo de este mismo curso, dividido en capítulos, y cada uno de ellos en subcapítulos. Aunque el libro sea algo lineal, como una lista, en el que cada capítulo sigue al anterior, también es posible acceder a cualquier punto de él a través de la tabla de contenido.

También se suelen organizar en forma de árbol los organigramas de mando en empresas o en el ejército, y los árboles genealógicos.

6.3 Operaciones básicas con árboles:

Salvo que trabajemos con árboles especiales, como los que veremos más adelante, las inserciones serán siempre en punteros de nodos hoja o en punteros libres de nodos rama. Con estas estructuras no es tan fácil generalizar, ya que existen muchas variedades de árboles.

De nuevo tenemos casi el mismo repertorio de operaciones de las que disponíamos con las listas:

- Añadir o insertar elementos.
- Buscar o localizar elementos.
- Borrar elementos.
- Moverse a través del árbol.
- Recorrer el árbol completo.

Los algoritmos de inserción y borrado dependen en gran medida del tipo de árbol que estemos implementando, de modo que por ahora los pasaremos por alto y nos centraremos más en el modo de recorrer árboles.

6.4 Recorridos por árboles:

El modo evidente de moverse a través de las ramas de un árbol es siguiendo los punteros, del mismo modo en que nos movíamos a través de las listas.

Esos recorridos dependen en gran medida del tipo y propósito del árbol, pero hay ciertos recorridos que usaremos frecuentemente. Se trata de aquellos recorridos que incluyen todo el árbol.

Hay tres formas de recorrer un árbol completo, y las tres se suelen implementar mediante recursividad. En los tres casos se sigue siempre a partir de cada nodo todas las ramas una por una.

Supongamos que tenemos un árbol de orden tres, y queremos recorrerlo por completo.

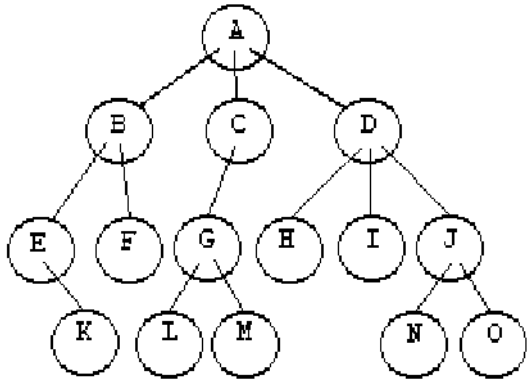
Partiremos del nodo raíz:

```
RecorrerArbol(raiz);
```

La función *RecorrerArbol*, aplicando recursividad, será tan sencilla como invocar de nuevo a la función *RecorrerArbol* para cada una de las ramas:

```
void RecorrerArbol(arbol a)
{
   if(a == NULL) return;
   RecorrerArbol(a->rama[0]);
   RecorrerArbol(a->rama[1]);
   RecorrerArbol(a->rama[2]);
}
```

Lo que diferencia los distintos métodos de recorrer el árbol no es el sistema de hacerlo, sino el momento que elegimos para procesar el valor de cada nodo con relación a los recorridos de cada una de las ramas.



Los tres tipos son:

Pre-orden:

En este tipo de recorrido, el valor del nodo se procesa antes de recorrer las ramas:

```
void PreOrden(arbol a)
{
   if(a == NULL) return;
   Procesar(dato);
   RecorrerArbol(a->rama[0]);
   RecorrerArbol(a->rama[1]);
   RecorrerArbol(a->rama[2]);
}
```

Si seguimos el árbol del ejemplo en pre-orden, y el proceso de los datos es sencillamente mostrarlos por pantalla, obtendremos algo así:

```
A B E K F C G L M D H I J N O
```

In-orden:

En este tipo de recorrido, el valor del nodo se procesa después de recorrer la primera rama y antes de recorrer la última. Esto tiene más sentido en el caso de árboles binarios, y también cuando existen ORDEN-1 datos, en cuyo caso procesaremos cada dato entre el recorrido de cada dos ramas (este es el caso de los árboles-b):

```
void InOrden(arbol a)
{
   if(a == NULL) return;
   RecorrerArbol(a->rama[0]);
   Procesar(dato);
   RecorrerArbol(a->rama[1]);
   RecorrerArbol(a->rama[2]);
}
```

Si seguimos el árbol del ejemplo en in-orden, y el proceso de los datos es sencillamente mostrarlos por pantalla, obtendremos algo así:

```
K E B F A L G M C H D I N J O
```

Post-orden:

En este tipo de recorrido, el valor del nodo se procesa después de recorrer todas las ramas:

```
void PostOrden(arbol a)
{
   if(a == NULL) return;
   RecorrerArbol(a->rama[0]);
   RecorrerArbol(a->rama[1]);
   RecorrerArbol(a->rama[2]);
   Procesar(dato);
}
```

Si seguimos el árbol del ejemplo en post-orden, y el proceso de los datos es sencillamente mostrarlos por pantalla, obtendremos algo así:

```
K E F B L M G C H I N O J D A
```

6.5 Eliminar nodos en un árbol:

El proceso general es muy sencillo en este caso, pero con una importante limitación, sólo podemos borrar nodos hoja:

El proceso sería el siguiente:

1. Buscar el nodo padre del que queremos eliminar.
2. Buscar el puntero del nodo padre que apunta al nodo que queremos borrar.
3. Liberar el nodo.
4. padre->nodo[i] = NULL;.

Cuando el nodo a borrar no sea un nodo hoja, diremos que hacemos una "poda", y en ese caso eliminaremos el árbol cuya raíz es el nodo a borrar. Se trata de un procedimiento recursivo, aplicamos el recorrido PostOrden, y el proceso será borrar el nodo.

El procedimiento es similar al de borrado de un nodo:

1. Buscar el nodo padre del que queremos eliminar.
2. Buscar el puntero del nodo padre que apunta al nodo que queremos borrar.
3. Podar el árbol cuyo padre es nodo.
4. padre->nodo[i] = NULL;.

En el árbol del ejemplo, para podar la rama 'B', recorreremos el subárbol 'B' en postorden, eliminando cada nodo cuando se procese, de este modo no perdemos los punteros a las ramas apuntadas por cada nodo, ya que esas ramas se borrarán antes de eliminar el nodo.

De modo que el orden en que se borrarán los nodos será:

```
K E F y B
```

6.6 Árboles ordenados:

A partir del siguiente capítulo sólo hablaremos de árboles ordenados, ya que son los que tienen más interés desde el punto de vista de TAD, y los que tienen más aplicaciones genéricas.

Un árbol ordenado, en general, es aquel a partir del cual se puede obtener una secuencia ordenada siguiendo uno de los recorridos posibles del árbol: inorden, preorden o postorden.

En estos árboles es importante que la secuencia se mantenga ordenada aunque se añadan o se eliminen nodos.

Existen varios tipos de árboles ordenados, que veremos a continuación:

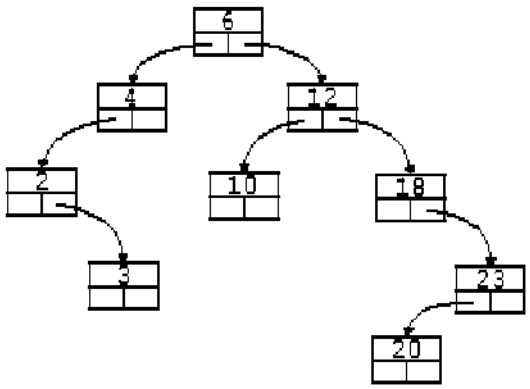
- Árboles binarios de búsqueda (ABB): son árboles de orden 2 que mantienen una secuencia ordenada si se recorren en inorden.
- Árboles AVL: son árboles binarios de búsqueda equilibrados, es decir, los niveles de cada rama para cualquier nodo no difieren en más de 1.
- Árboles perfectamente equilibrados: son árboles binarios de búsqueda en los que el número de nodos de cada rama para cualquier nodo no defieren en más de 1. Son por lo tanto árboles AVL también.
- Árboles 2-3: son árboles de orden 3, que contienen dos claves en cada nodo y que están también equilibrados. También generan secuencias ordenadas al recorrerlos en inorden.

Árboles-B: caso general de árboles 2-3, que para un orden M, contienen M-1 claves.

Capítulo 7 Árboles binarios de búsqueda (ABB):

7.1 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.



7.2 Operaciones en ABB

El repertorio de operaciones que se pueden realizar sobre un ABB es parecido al que realizábamos sobre otras estructuras de datos, más alguna otra propia de árboles:

- Buscar un elemento.
- Insertar un elemento.
- Borrar un elemento.
- Movimientos a través del árbol:
  - Izquierda.
  - Derecha.
  - Raíz.
- Información:
  - Comprobar si un árbol está vacío.
  - Calcular el número de nodos.
  - Comprobar si el nodo es hoja.
  - Calcular la altura de un nodo.
  - Calcular la altura de un árbol.

7.3 Buscar un elemento

Partiendo siempre del nodo raíz, el modo de buscar un elemento se define de forma recursiva.

- Si el árbol está vacío, terminamos la búsqueda: el elemento no está en el árbol.
- Si el valor del nodo raíz es igual que el del elemento que buscamos, terminamos la búsqueda con éxito.
- 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.

El valor de retorno de una función de búsqueda en un ABB puede ser un puntero al nodo encontrado, o NULL, si no se ha encontrado.

7.4 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 = Raíz
- 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.

7.5 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:

1. Se trata de un nodo hoja: en ese caso lo borraremos directamente.
2. 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.

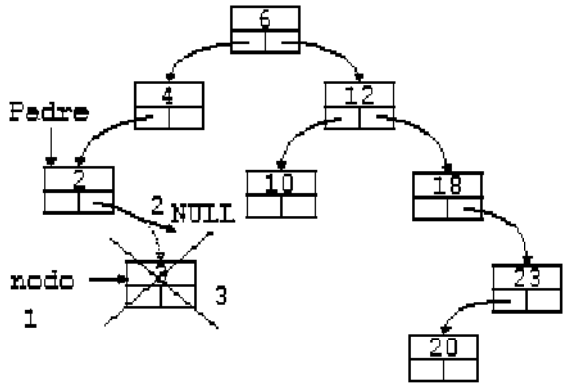
Necesitamos un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL.

- 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. (Ver ejemplo 3)
- 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 1: Borrar un nodo hoja

En el árbol de ejemplo, borrar el nodo 3.

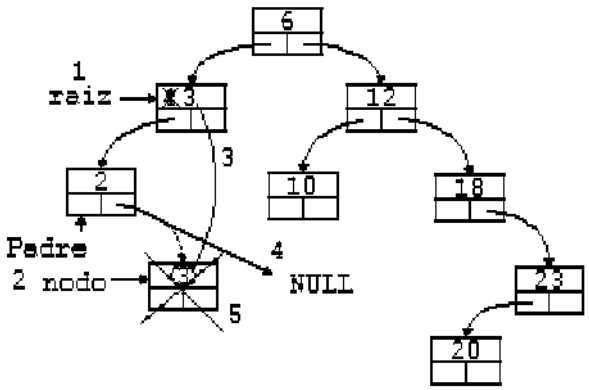
1. Localizamos el nodo a borrar, al tiempo que mantenemos un puntero a 'Padre'.
2. Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.
3. Borramos el 'nodo'.



Ejemplo 2: Borrar un nodo rama con intercambio de un nodo hoja.

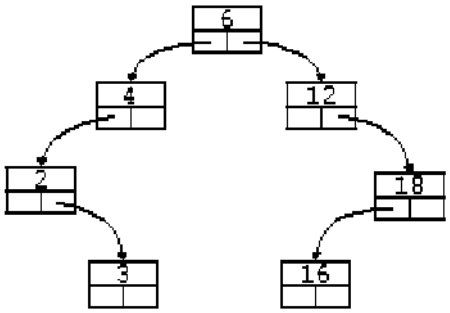
En el árbol de ejemplo, borrar el nodo 4.

1. Localizamos el nodo a borrar ('raíz').
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'.
3. Intercambiamos los elementos 3 y 4.
4. Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.
5. Borramos el 'nodo'.

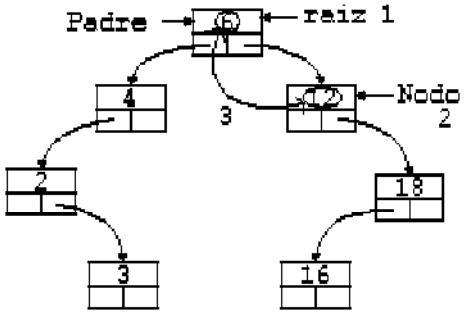


Ejemplo 3: Borrar un nodo rama con intercambio de un nodo rama.

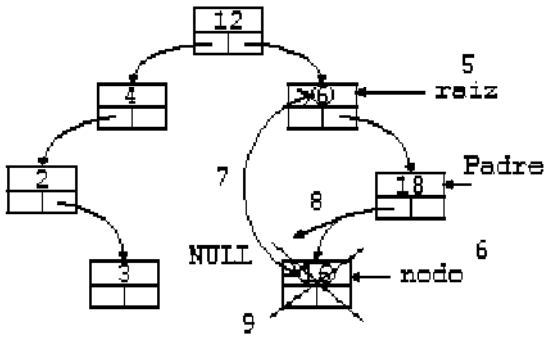
Para este ejemplo usaremos otro árbol. En éste borraremos el elemento 6.



1. Localizamos el nodo a borrar ('raíz').
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'.
3. Intercambiamos los elementos 6 y 12.
4. Ahora tenemos que repetir el bucle para el nodo 6 de nuevo, ya que no podemos eliminarlo.



5. Localizamos de nuevo el nodo a borrar ('raiz').
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'.
7. Intercambiamos los elementos 6 y 16.
8. Hacemos que el puntero de 'Padre' que apuntaba a 'nodo', ahora apunte a NULL.
9. Borramos el 'nodo'.



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

7.6 Movimientos a través del árbol

No hay mucho que contar. Nuestra estructura se referenciará siempre mediante un puntero al nodo Raíz, este puntero no debe perderse nunca.

Para movernos a través del árbol usaremos 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.

7.7 Información

Hay varios parámetros que podemos calcular o medir dentro de un árbol. Algunos de ellos nos darán idea de lo eficientemente que está organizado o el modo en que funciona.

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 contar los nodos podemos recurrir 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.

Esto es muy sencillo, 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 nos 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 que el del elemento que buscamos, terminamos la búsqueda y el valor de la altura es 'Altura'.
- 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. Para buscar este valor tendremos que recorrer todo el árbol, de nuevo es indiferente el tipo de recorrido que hagamos, cada vez que cambiemos de nivel incrementamos la variable que contiene la altura del nodo actual, 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.

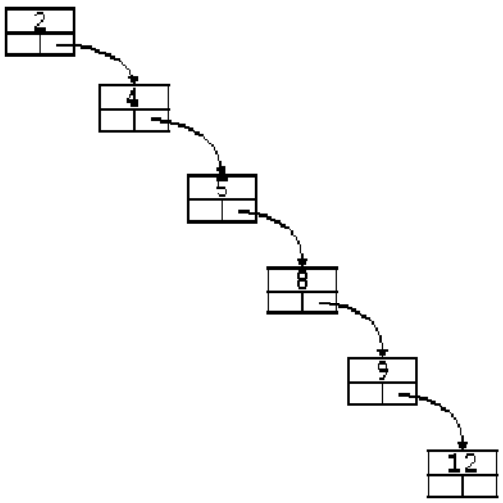
- Iniciamos un recorrido del árbol en postorden, con la variable de altura igual a cero.
- Cada vez que empecemos a recorrer una nueva rama, incrementamos la altura para ese nodo.
- Después de procesar las dos ramas, verificamos si la altura del nodo es mayor que la variable que almacena la altura actual del árbol, si es así, actualizamos esa variable.

7.8 Árboles degenerados

Los árboles binarios de búsqueda tienen un gran inconveniente. Por ejemplo, supongamos que creamos un ABB a partir de una lista de valores ordenada:

```
2, 4, 5, 8, 9, 12
```

Difícilmente podremos llamar a la estructura resultante un árbol:



Esto es lo que llamamos un árbol binario de búsqueda degenerado, y en el siguiente capítulo veremos una nueva estructura, el árbol AVL, que resuelve este problema, generando árboles de búsqueda equilibrados.