C++

ARBOLES BINARIOS

Vamos a ver...

Arboles:

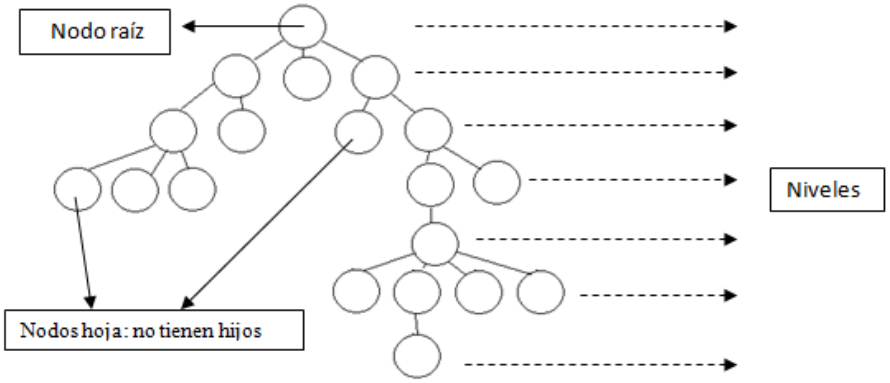
- Definición
- Vocabulario

Arboles binarios:

- Definición
- Implementación
- Recorrido en profundidad
- Recorrido por niveles
- ABB

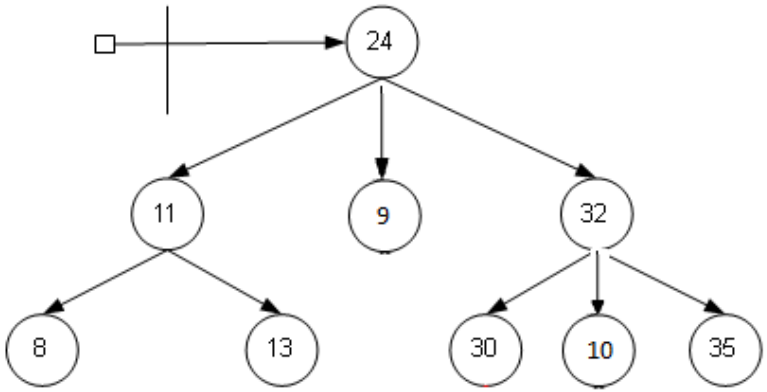
Arboles: Definición 1

- Un **árbol** es una colección de elementos, llamados nodos, uno de los cuales se distingue con el nombre de *raíz*.
- Son estructuras de datos que constan de un conjunto de nodos organizados *jerárquicamente*. Cada nodo tiene un padre y sólo uno, excepto la *raíz*.
- Cada nodo puede tener cero o más hijos. Según las características del árbol, la cantidad de hijos puede estar limitada o no.



Arboles: Definición 2

- *Recursivamente* un **árbol** puede verse como una estructura formada por la *raíz* de dicho árbol y una *lista de arboles* (los hijos).
- Este nodo raíz es el *padre de las raíces* de los arboles que componen la lista, a partir del cual, se establece la relación de paternidad entre ellos.
- El *conjunto vacío* de nodos es un árbol llamado nulo o vacío.



Arboles: Vocabulario

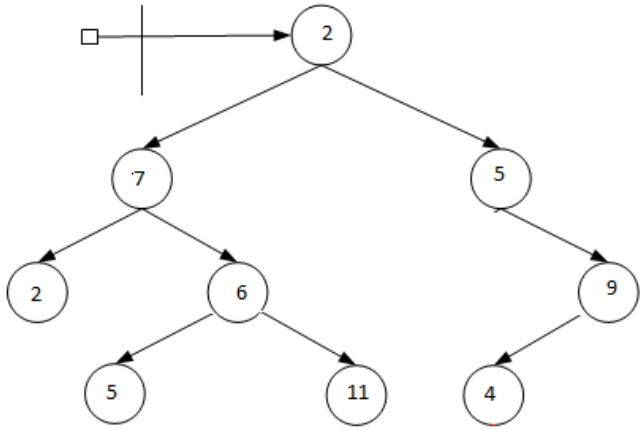
- El *camino* de un nodo *n* a un nodo *m*: es una secuencia de nodos n, n1, n2, ..., m.
- La *longitud* de un camino: número de nodos en el camino menos uno.
- Si existe camino de un nodo *a* a un nodo *b*, entonces se dice que *a* es un *ancestro* de *b* y que *b* es un *descendiente* de *a*.
- Una *hoja* es un nodo sin descendientes propios.
- Un *subárbol* de un árbol es un nodo del árbol junto con todos sus descendientes.
- La *profundidad* de un nodo es la longitud del camino único de la raíz al nodo.

Arboles: Vocabulario

- El *grado* de un nodo es el número de hijos que tiene.
- La *altura* de un árbol es la distancia desde la raíz hasta la hoja más lejana.
- Dado un árbol de altura h se definen como *niveles* 0...h, de modo que el nivel i esta compuesto por todos los nodos de profundidad i.
- Un árbol es *completo* si todas las hojas están a igual distancia de la raíz.
- Un árbol es *balanceado* si en cada nodo, las alturas de los subárboles no dieren en mas de un nivel.

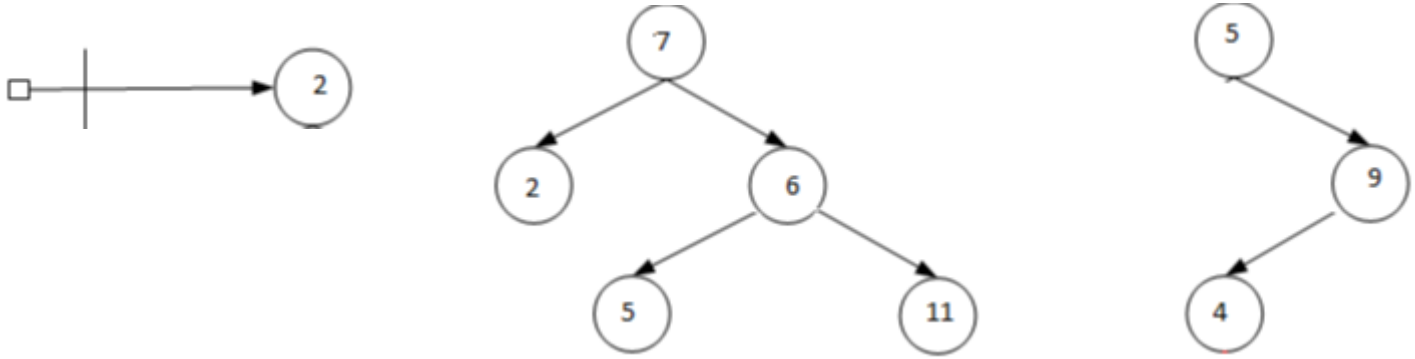
Arboles binarios: Definición 1

- Un **árbol binario** es un árbol cuyos nodos tienen a lo sumo dos hijos o bien es un árbol nulo.
- Los hijos de un árbol binario se indican como *hijo izquierdo* e *hijo derecho.*
- Un árbol binario puede definirse como un árbol que en cada nodo puede tener como máximo *grado 2* (máximo dos hijos).



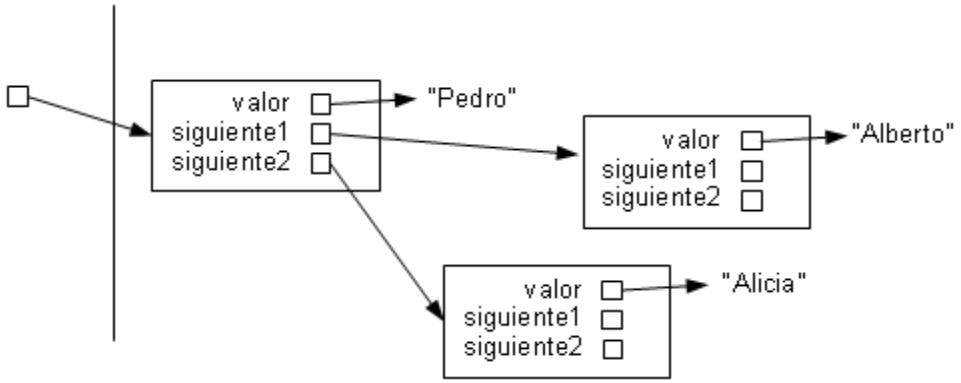
Arboles binarios: Definición 2

- Un **árbol binario** es un conjunto finito de elementos, el cual esta vacío o dividido en tres subconjuntos separados:
  - El primer subconjunto contiene un elemento único llamado *raíz.*
  - El segundo subconjunto es en si mismo un árbol binario y se le conoce como *subárbol izquierdo* del árbol original.
  - El tercer subconjunto es también un árbol binario y se le conoce como *subárbol derecho* del árbol original.



Arboles binarios: Implementación

- Una opción es usar nodos que no tienen un único siguiente sino *dos.*



Arboles binarios: Implementación

```
class Nodo
{
   private:
      int valor ;
      Nodo* hijoIzquierda;
      Nodo* hijoDerecha;
   …
}
```

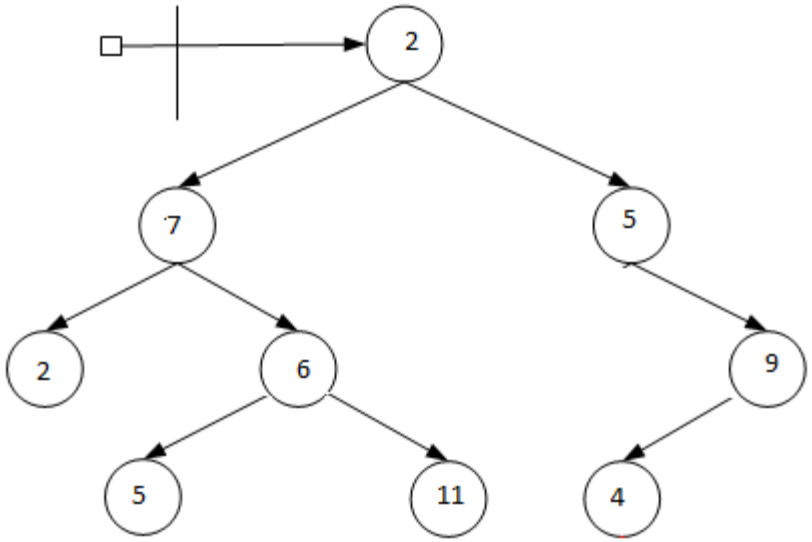
Arboles binarios: Implementación

```
class Arbol
{
   private:
      Nodo* raiz ;
   public:
      Arbol (int valor)   // constructor
      {
         this->raiz = new Nodo (valor) ;
      }
   …
}
```

Recorridos en profundidad

Recorrido en preorden

- Se realiza cierta acción sobre el nodo actual, luego se trata el subárbol izquierdo y cuando se haya concluido, el subárbol derecho.

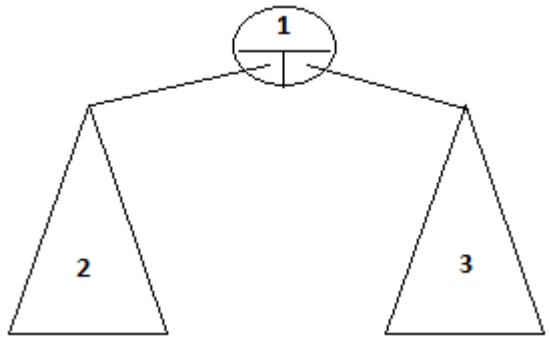


- El recorrido en preorden será: 2, 7, 2, 6, 5, 11, 5, 9 y 4.

Recorridos en profundidad

Recorrido en preorden (recursivo)

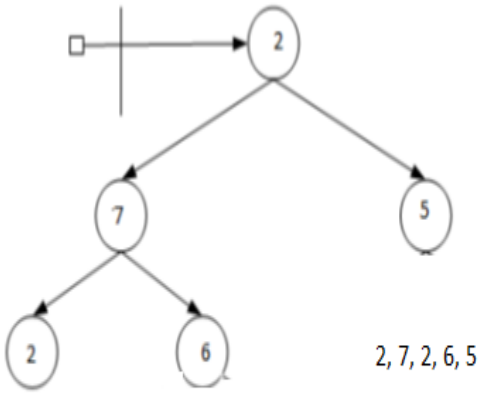
```
public void preorden (Nodo* raiz)
{
   if (raiz != nullptr)
   {
      tratar (raiz);
      preorden (raiz->getHijoIzquierda()) ;
      preorden (raiz->getHijoDerecha()) ;
   }
}
```



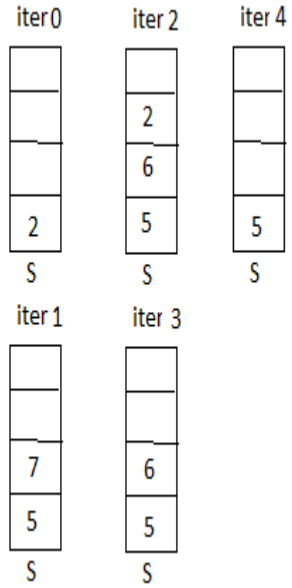
Recorridos en profundidad

Recorrido en preorden (iterativo):

```
push (s,raiz);
MIENTRAS (s <> NULL)
   p = pop (s);
   tratar (p);
   SI (DER(p) <> NULL)
      ENTONCES push (s ,DER(p));
   FIN-SI
   SI (IZQ(p) <> NULL)
      ENTONCES push (s , IZQ(p));
   FIN-SI
FIN-MIENTRAS
```



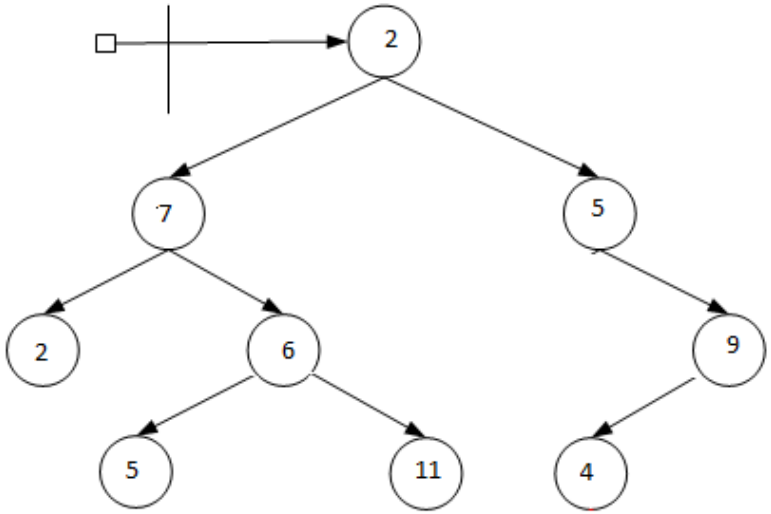
2, 7, 2, 6, 5



Recorridos en profundidad

Recorrido en postorden

- Se trata primero el subárbol izquierdo, después el derecho y por ultimo el nodo actual.

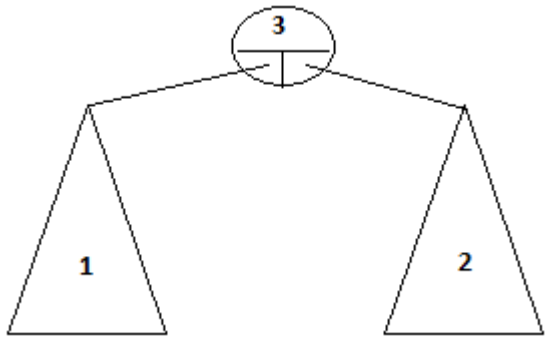


- El recorrido en inorden será: 2, 5, 11, 6, 7, 4, 9, 5 y 2.

Recorridos en profundidad

Recorrido en postorden (recursivo)

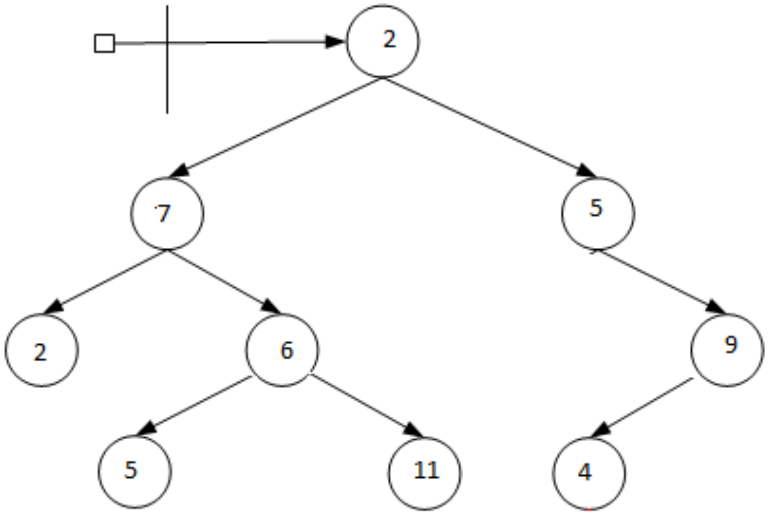
```
public void postorden (Nodo* raiz)
{
   if (raíz != nullptr)
   {   postorden (raiz->getHijoIzquierda());
       postorden (raiz->getHijoDerecha());
       tratar(raiz);
   }
}
```



Recorridos en profundidad

Recorrido en inorden

- Se trata primero el subárbol izquierdo, después el nodo actual y por ultimo el subárbol derecho.

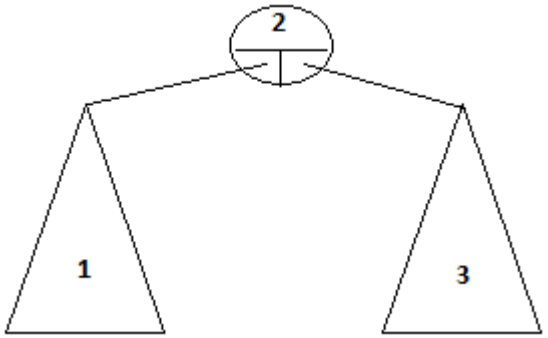


- El recorrido en inorden será: 2, 7, 5, 6, 11, 2, 5, 4, 9.

Recorridos en profundidad

Recorrido en inorden (recursivo)

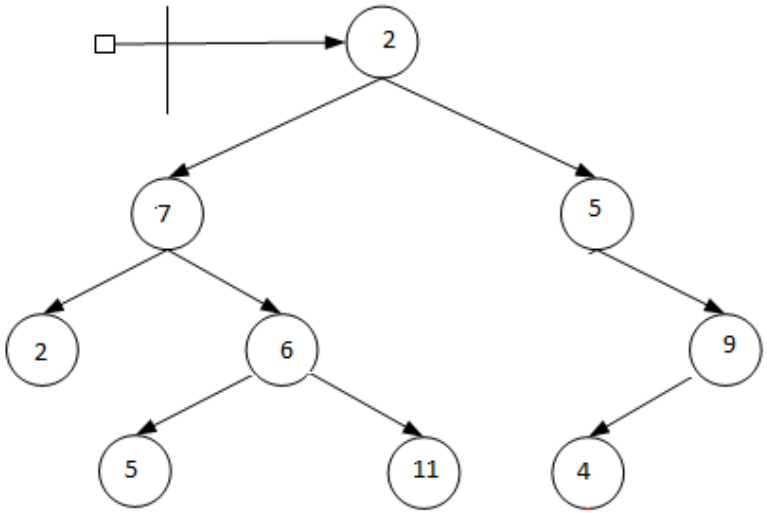
```
public void inorden (Nodo* raiz)
{   if (raiz != nullptr)
    {   inorden (raiz->getHijoIzquierda());
        tratar(raiz);
        inorden (raiz->getHijoDerecha());
    }
}
```



Recorridos en amplitud

Recorrido por niveles

- El recorrido se realiza en orden por los distintos *niveles* del árbol.
- El recorrido por niveles *no es de naturaleza recursiva*.

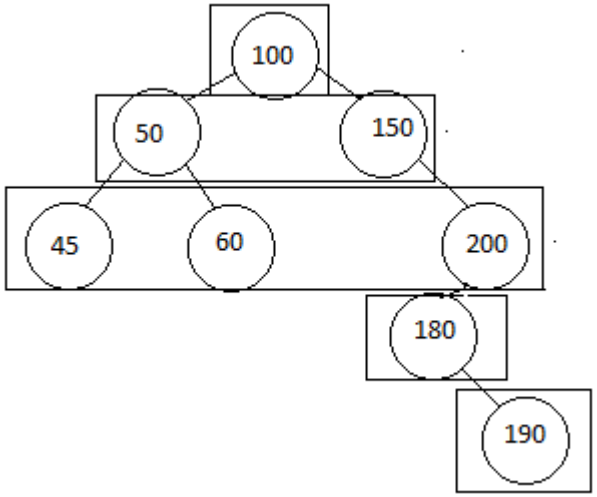


- El recorrido en amplitud será: 2, 7, 5, 2, 6, 9, 5, 11 y 4.

Recorridos en amplitud

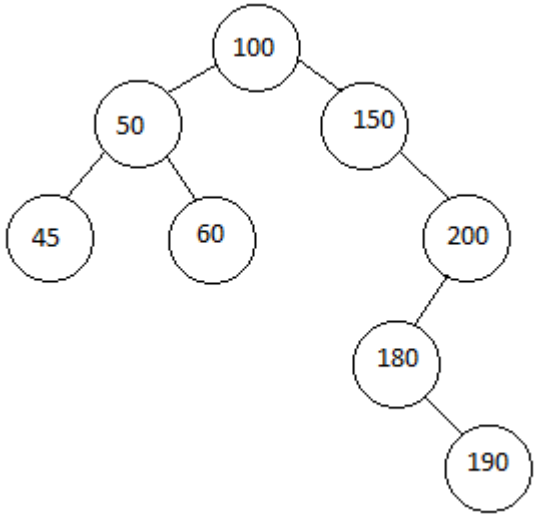
Recorrido por niveles (pseudocódigo)

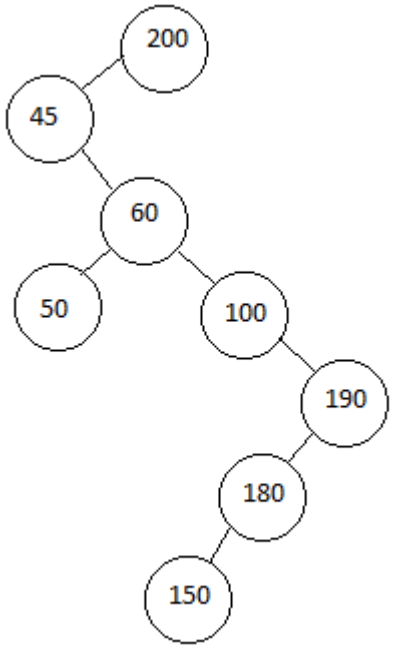
```
enqueue (q , raiz) ;
MIENTRAS (q<>NULL)
   p=dequeue (q) ;
   tratar (p) ;
   SI (IZQ(p) <> NULL)
      ENTONCES enqueue (q ,IZQ(p)) ;
   SI (DER(p) <> NULL)
      ENTONCES enqueue (q ,DER(p)) ;
FIN-MIENTRAS
```



Arboles binarios: ABB

- Un ABB es un *árbol binario de búsqueda*.





- Los descendientes a la izquierda de todo nodo x son *menores que x*, y los descendientes a la derecha son *mayores que x*.
- El recorrido *inorden* dará los valores de clave ordenados de menor a mayor.

Arboles binarios: ABB

Búsqueda recursiva (pseudocódigo):

```
TDato buscarABB (k,t)
p=raiz (t) ;
SI (p==NULL O clave(p)==k)  //caso base
   RETORNAR dato(p);
SINO                        //caso recursivo
   SI (k < clave(p))
      bucarABB=buscarABB(k,IZQ(p));
   SINO
      bucarABB=buscarABB(k,DER(p));
   FIN-SI
FIN-SI
```

Arboles binarios: ABB

Búsqueda iterativa (pseudocódigo):

```
TDato buscarABB (k,t)
p=raiz (t);
SI (p<>NULL)
MIENTRAS ((p<>NULL) && (clave(p)<>k))
   SI (k < clave(p))
      p=IZQ(p);
   SINO
      p=DER(p);
FIN-MIENTRAS
SI (clave(p) == k)
   RETORNAR dato ( p );
```

Arboles binarios: ABB

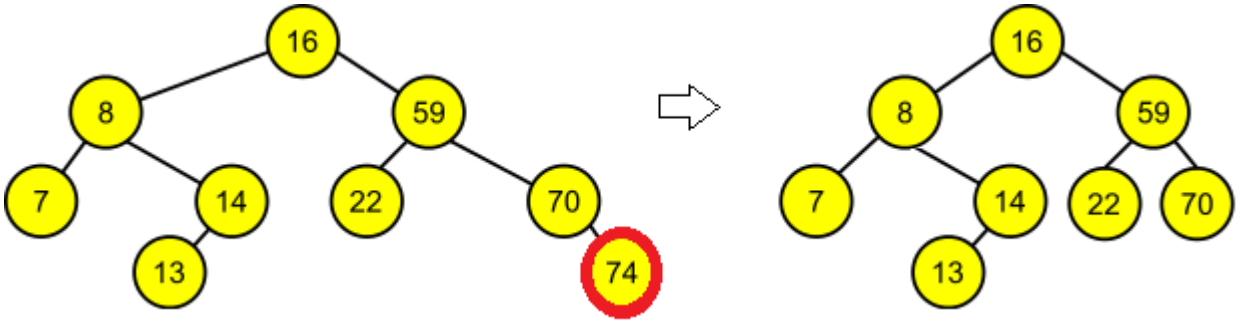
Inserción recursiva (pseudocódigo):

```
void insertarABB (t,k)
p=raiz (t);
SI (p==NULL)
   p=new Nodo (k);
SINO
   SI (k < clave (p))
      insertarABB (IZQ(p),k);
   SINO
      insertarABB (DER(p),k);
   FIN-SI
FIN-SI
```

Arboles binarios: ABB

Borrado: Caso 1

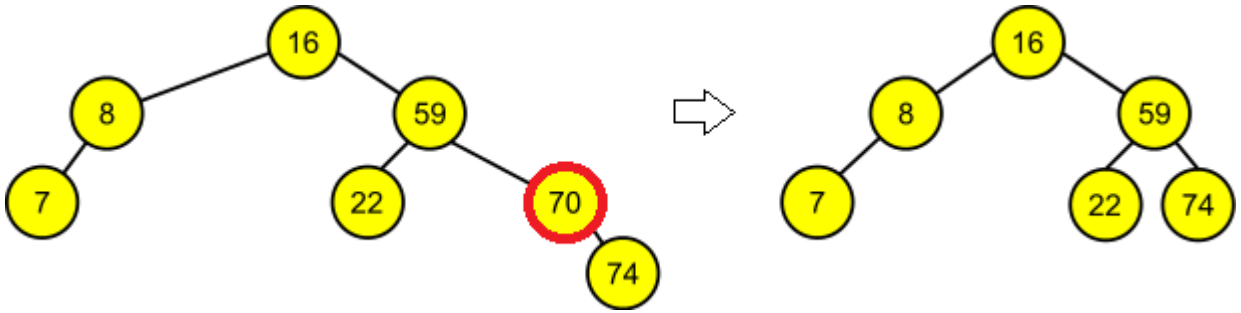
- Para borrar un nodo sin hijos o nodo hoja simplemente se borra y se establece a nulo el apuntador de su padre.



Arboles binarios: ABB

Borrado: Caso 2

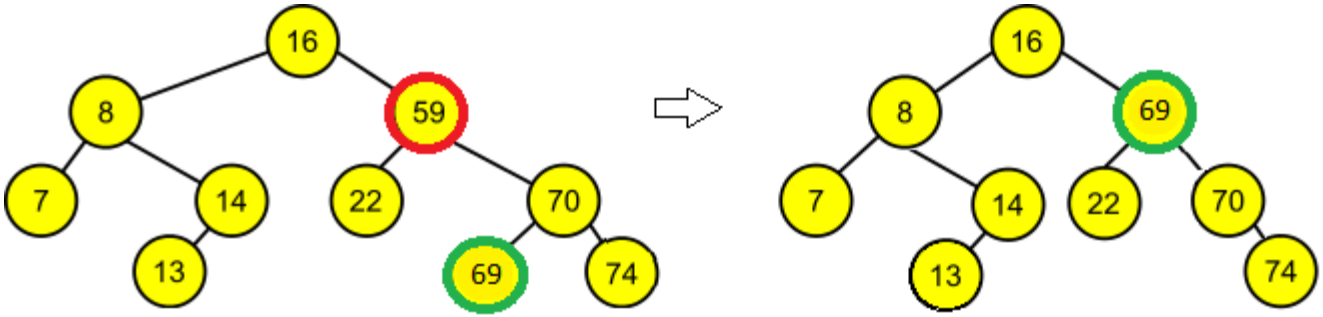
- Para borrar un nodo con un subárbol hijo se borra el nodo y el subárbol pasa a ocupar su lugar.



Arboles binarios: ABB

Borrado: Caso 3

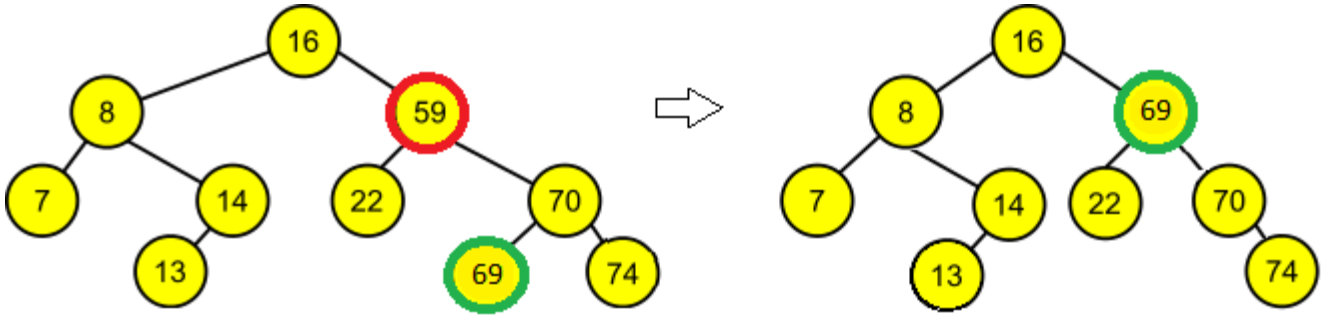
- Para borrar un nodo con dos subárboles hijo, hay que tomar el hijo derecho del Nodo que queremos eliminar y recorrerlo hasta el hijo mas a la izquierda.
- Reemplazamos el valor del nodo que queremos eliminar por el nodo que encontramos.



Arboles binarios: ABB

Borrado: Caso 3

- El nodo que encontramos por ser el mas a la izquierda es imposible que tenga nodos a su izquierda pero si es posible que tenga un subárbol a la derecha.
- Eliminamos este nodo de acuerdo a caso 1 o caso 2.



Arboles binarios: ABB

Borrado (pseudocódigo):

```
pa= EncontrarPadre (); //retorna el puntero al padre del nodo que contiene x
si (pa no es nulo)     //si pa es nulo, x no está en el ABB
{
   Si (nodo con x no tiene hijos)
      {ElimCaso1(pa);}
   Si no
      Si (nodo con x tiene un solo hijo)
         {ElimCaso2(pa);}
      Si no
         {ElimCaso3(pa);}
}
```

Arboles binarios: ABB

Borrado: Caso 1

```
ElimCaso1 (var pa: puntero a nodo)
{
   //aux: puntero a nodo
   Asignar a aux la dirección del nodo que contiene x;
   Poner a nulo el puntero del nodo apuntado por pa que apuntaba a nodo que
   contiene x;
   Eliminar nodo que contiene x utilizando aux;
}
```

Arboles binarios: ABB

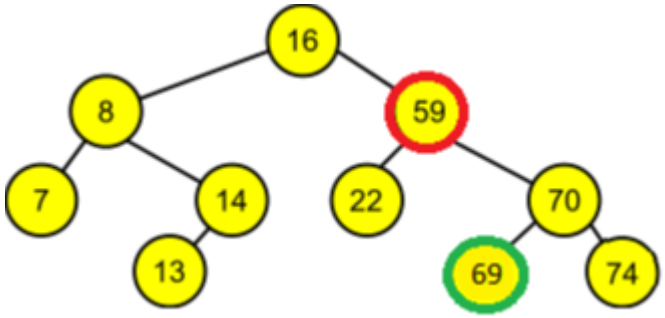
Borrado: Caso 2

```
ElimCaso2 (var pa: puntero a nodo)
{
   Asignar a aux la dirección del nodo que contiene x;
   Asignar al puntero del nodo apuntado por pa que apuntaba a nodo que contiene x
   la dirección del hijo no nulo del nodo que contiene x;
   Eliminar nodo que contiene x utilizando aux;
}
```

Arboles binarios: ABB

Borrado: Caso 3

```
ElimCaso3 (var pa: puntero a nodo)
{
   Asignar a aux la dirección del nodo que contiene x;
   aux2= pPadreMayorMenores(aux);
   Asignar al atributo clave del nodo que contiene x el mayor de los menores,
   apuntado por un hijo de aux2
   Si (nodo apuntado por aux2 tiene un hijo)
      {ElimCaso1(aux2);}
   Si no
      {ElimCaso2(aux2);}
   Eliminar nodo que contiene x utilizando aux;
}
```



Fin