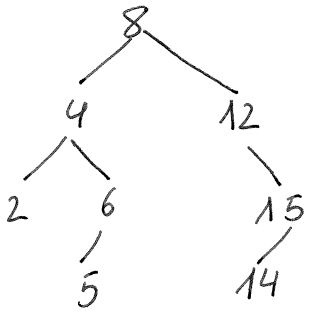
(ABB)

LECCIÓN 21: Arboles Binarios Búsqueda

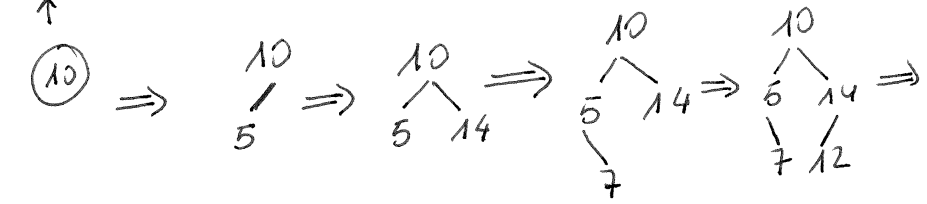
Arboles Binarios de Búsqueda (ABB)

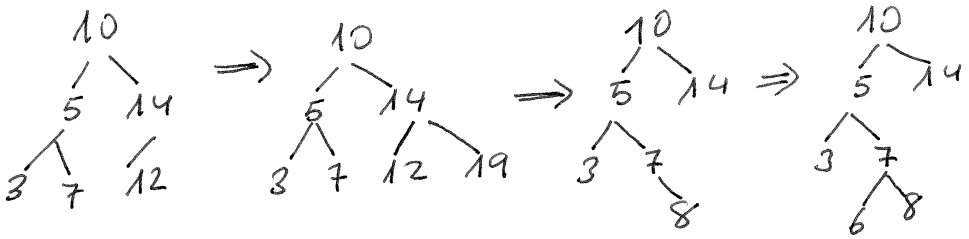
Es un árbol binario, con las etiquetas de los nodos ordenados de forma que el elemento situado en un nodo es mayor que todos los que se encuentran en el subarbol izquierdo y menor que los que se situan en el subarbol derecho.



Cómo se construye

10, 5, 14, 7, 12, 3, 19, 8, 6





—

- Se suponen que no hay elementos repetidos. Las búsquedas se realizan en $O(\log_2(n))$ donde n es el nº de nodos o etiquetas
- Los procesos de insercion y borrado son más complicados.
- El recorrido en inorden de un ABB da la ordenacion de todos los elementos.

```
template <class T>
struct info-nodo{
   T et;
   infonodo*padre*hizq*hder;
}
```

operacion ==, <, > sobre T tienen que estar definidas.

```
template <class T>
info-nodo<T>* Buscar(infonodo<T> * n, T x)
{
  if (n!=0){
    if (n->et==x)
       return n;
    else
       if (n->et<x)
          return Buscar(n->hizq,x)
       else return Buscar(n->hder,x)
  }
  else return n;
}
```

2ABB

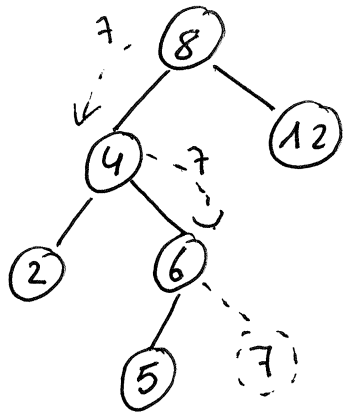
LECCION 21: ABB

Búsqueda sin usar recursividad

```
template <class T>
info_nodo <T> * Buscar(info_nodo <T> * n, T x){
        if (n==0)
            return n;
        else{
            info_nodo <T> *p=n;
            while (p!=0){
                if (p->et==x)
                    return p;
                else
                    if (p->et <x)
                        p= p->hder;
                    else
                        p=p->hizq;
            }
            return p;
        }
}
```

LECCION 21: A BB

- Buscar donde insertar el elemento x=7 en el siguiente árbol:



El proceso va comparando con las etiquetas del árbol empezando por la raiz hasta alcanzar un nodo que no tiene hder y x es mayor que el nodo o alcanzar un nodo que no tiene hizq y x es menor que el nodo.

```
template <class T>
bool Insertar (info_nodo <T> *&n, T x) {
    bool res=false;
    if (n==0){
        n= new info_nodo (x);
        return true;
    }else{
        if (n->et <x){
            res= Insertar (n->hder, x);
            if (res) {
                n->hder->padre = n;
            }
            return res;
        }else
            if (n->et >x){
                res= Insertar (n->hizq ,x);
                if (res){
                    n->hizq->padre=n;
                }return res;
            }else return false;
    }
}
```

4ABB LECCION 22: ABB SIN RECURSIVIDAD

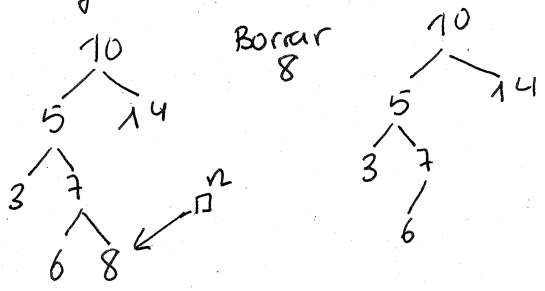
```
template <class T>
bool Insertar (info_nodo <T> * &n, T x) {

        if (n==0) {
            n = new info_nodo (x);
            return true;
        }
        else {
            char where_put=1; info_nodo <T> * aux=n;
            info_nodo <T> * padre = n->padre;
            bool find=false;
            while (! find && n!=0){
                if (n->et <x) {
                    padre=n;
                    where_put=2;
                    n=n->hder;
                }
                else
                    if (n->et>x){
                        padre=n;
                        where_put=1;
                        n=n->hizq;
                    }
                    else find=true;
            }
            if (find)
                return false;
            else {
                if (where_put==1) {
                    padre->hizq = new info_nodo (x);
                    padre->hizq->padre=padre;
                    n=aux;
                }
                else {
                    padre->hder= new info_nodo (x);
                    padre->hder->padre = padre;
                    n=aux;
                }
            return true;
        }
}
```

LECCION 22: ABB

BORRADO

1er POSIBILIDAD: El elemento es una hoja

Ej:



CODIGO: Queremos borrar el info-nodo que apunta n.

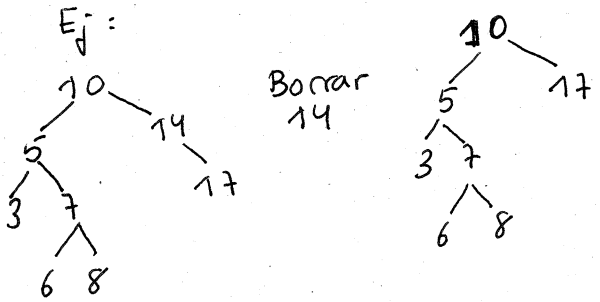
```
info_nodo <T> * aux=n;
if (aux->padre != 0){
    if (aux->padre->hder==n)
        aux->padre->hder=0;
    else
        aux->padre->hizq=0;
}
delete aux;
```

2ª POSIBILIDAD: El elemento está en un nodo que no es hoja.

ESTRATEGIA: Buscar el siguiente en orden y reemplazarlo

CASO 1: Solamente tiene un hijo a la drch

Ej:

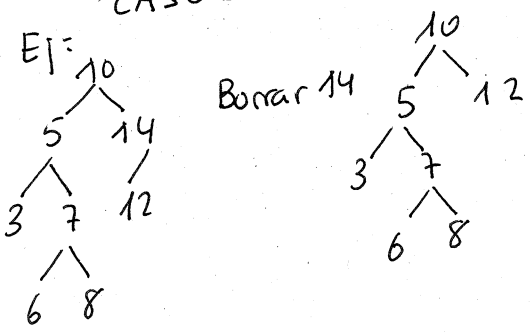


CODIGO

```
info_nodo <T> * padre=n->padre;
if (padre!=0){
    if (padre->hder==n){
        padre->hder=n->hder;
        padre->hder->padre=padre;
    }
    else
        if (padre->hizq==n){
            padre->hizq=n->hder;
            padre->hizq->padre=padre;
        }
}
info_nodo <T> * aux=n; n=n->hder; delete aux;
```

CASO 2: Solamente tiene un hijo a la hizq

Ej:



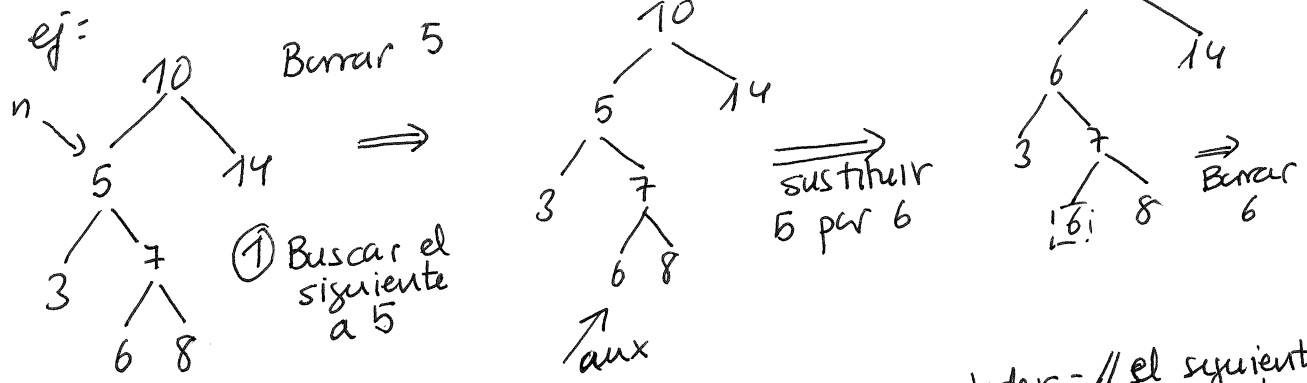
CODIGO

```
info_nodo <T> * padre=n->padre;
if (padre!=0){
    if (padre->hizq==n){
        padre->hizq=n->hizq;
        padre->hizq->padre=padre;
    }
    else if (padre->hder==n){
        padre->hder=n->hizq;
        padre->hder->padre=padre;
    }
}
info_nodo <T> * aux=n; n=n->hizq; delete aux;
```

LECCION 23: ABB

CASO 3: Tiene hizq y hder

ej:




```
info_nodo <T> * aux = n->hder; // el siguiente tiene
                               // que estar en esta
                               // rama

while (aux->hizq != 0){
      aux = aux->hizq;

}
n->et = aux->et
Borrar(aux)
```

```
void Borrar (info_nodo <T> *&n, T x){
    if (n!=0){
      if (n->et == x)
        EliminarRaiz(n);
      else if (n<et<x)
             Borrar (n->hder);
           else Borrar(n->hizq);

    }
}
void PutHijo_Padre (info_nodo <T> * n, info_nodo <T>* nuevo)
{   if (n->padre != 0)
      if (n->padre->hder == n)
            n->padre->hder = nuevo;
      else  n->padre->hizq = nuevo;

}
```

7ABB

LECCION 24: ABB

```
void EliminarRaiz( info_nodo <T> *&n){

    if (n->hizq ==0 && n->hder==0){
            PutHijo_Padre(n,0);
            delete n;

    }
    else
      if (n->hizq==0){
            PutHijo_Padre(n, n->hder);
            info_nodo<T> * aux=n;
            n=n->hder;
            delete aux;

      }
      else
        if (n->hder==0){
            PutHijo_Padre(n, n->hizq);
            info_nodo<T> * aux=n;
            n=n->hizq;
            delete aux;
        }
        else{
            info_nodo<T> *aux= n->hder;
            while(aux->hizq!=0)
                    aux=aux->hizq;

            n->et=aux->et;

            Borrar(aux, aux->et);

        }

}
```