Árboles Binarios de Búsqueda

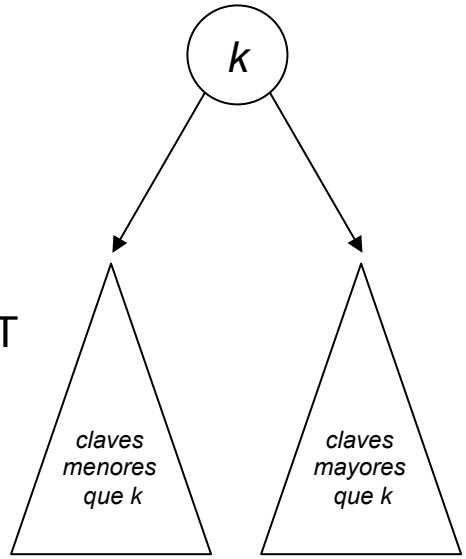
Programación II

Facultade de Informática

UNIVERSIDADE DA CORUÑA

Definición

- Un árbol binario de búsqueda (ABB)
  - Es un árbol binario
  - Tiene asociada una clave de ordenación
  - Cumple que para cualquier nodo T del árbol,
    - los valores de los nodos del subárbol izquierdo de T son menores que el valor de T
    - los valores de los nodos del subárbol derecho de T son mayores que el valor de T.



- Mayor eficiencia frente a…
  - estructuras estáticas en operaciones de inserción y eliminación
  - estructuras dinámicas en la operación de búsqueda

Ventajas e Inconvenientes

- Eficiencia del proceso de búsqueda en árboles equilibrados
  - Árbol equilibrado: las ramas izquierda y derecha de cada nodo tienen aproximadamente la misma altura
    - El árbol lleno sería el árbol equilibrado perfecto con todos los nodos con subárboles izquierdo y derecho de la misma altura
- Si los nodos a insertar en el árbol aparecen en orden aleatorio el árbol tenderá a ser equilibrado
- Si los nodos a insertar aparecen con un orden determinado el árbol tenderá a ser degenerado y se pierde eficiencia en las búsquedas

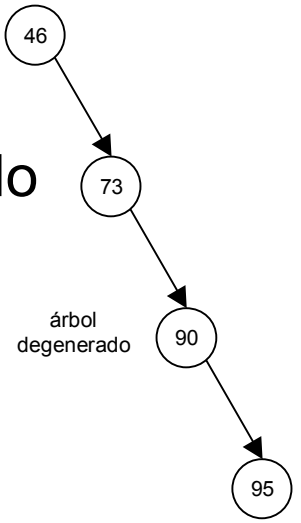


ABB: Búsqueda de una clave

- Pseudocódigo
  - Se compara la clave a buscar con la raíz del árbol
    - Si el árbol es vacío la búsqueda acaba sin éxito
    - Si clave = valor de la raíz, la búsqueda acaba con éxito
    - Si clave < valor de la raíz, la búsqueda continúa por el subárbol izquierdo
    - Si clave > valor de la raíz, la búsqueda continúa por el subárbol derecho

```
function BuscarClave (A:tArbolBin; clave:tClave): tArbolBin;
begin
  if EsArbolVacio(A) then
     BuscarClave:= nulo
  else if clave = A^.clave then
          BuscarClave:= A
       else if clave < A^.clave then
               BuscarClave:= BuscarClave (A^.izdo, clave)
            else
               BuscarClave:= BuscarClave (A^.dcho, clave)
end;
```

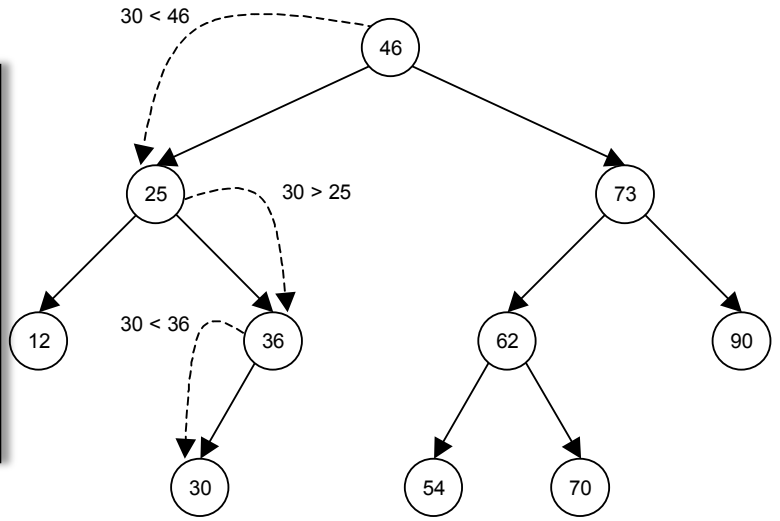


ABB: Inserción de una clave

- Pseudocódigo
  - Se compara la clave a insertar con la raíz del árbol
    - Si el árbol está vacio insertamos una hoja con la clave en esa posición
    - Si clave < valor de la raíz, la inserción continúa por el subárbol izquierdo
    - Si clave > valor de la raíz, la inserción continúa por el subárbol derecho
    - Si clave = valor (claves repetidas) no se hace nada

```
Function InsertarClave (var A: tArbolBin; clave: tClave):
boolean;
begin
  if EsArbolVacio(A) then
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

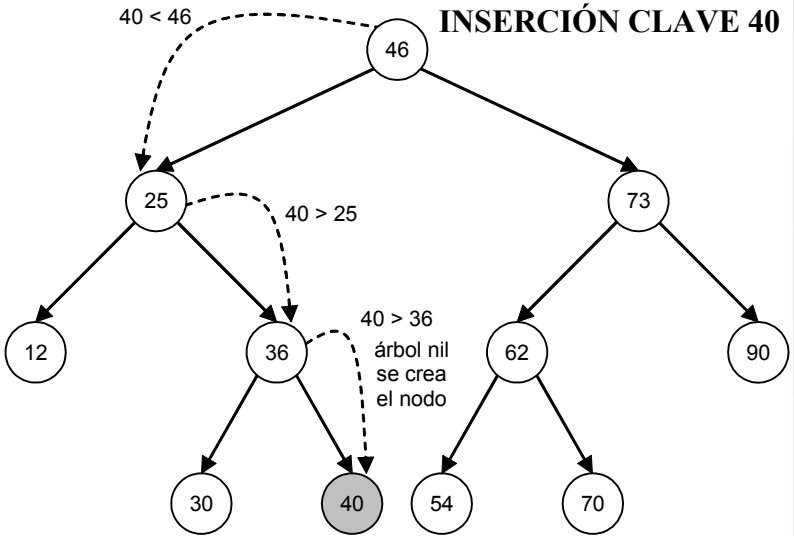
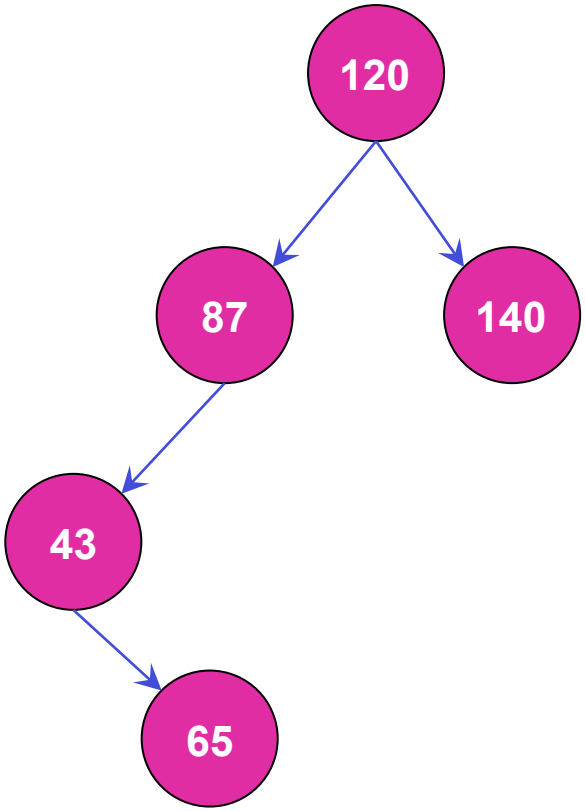
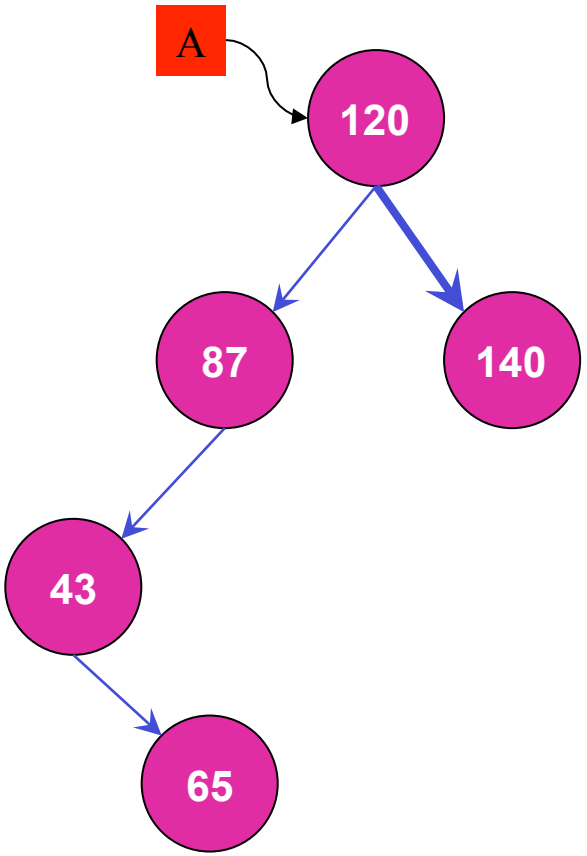


ABB: Insertar la clave 130



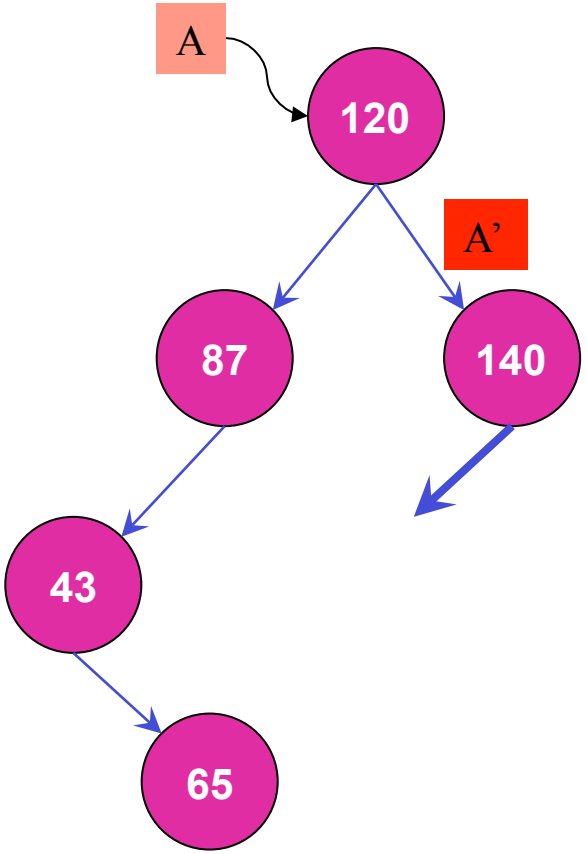

```
Function InsertarClave (var A:tArbolBin; clave:tClave): boolean;
begin
  if EsArbolVacio(A)
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

ABB: Insertar la clave 130



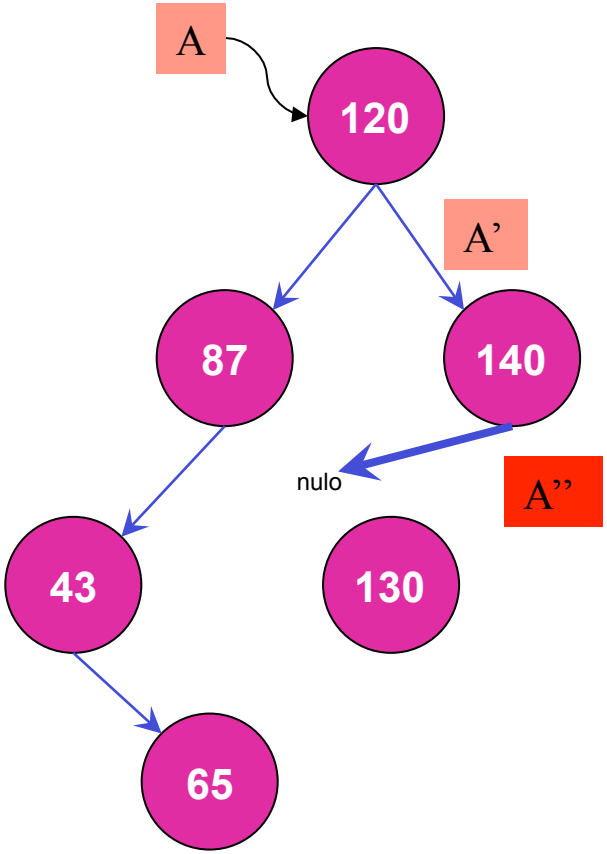

```
Function InsertarClave (var A:tArbolBin; clave:tClave): boolean;
begin
  if EsArbolVacio(A)
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

ABB: Insertar la clave 130



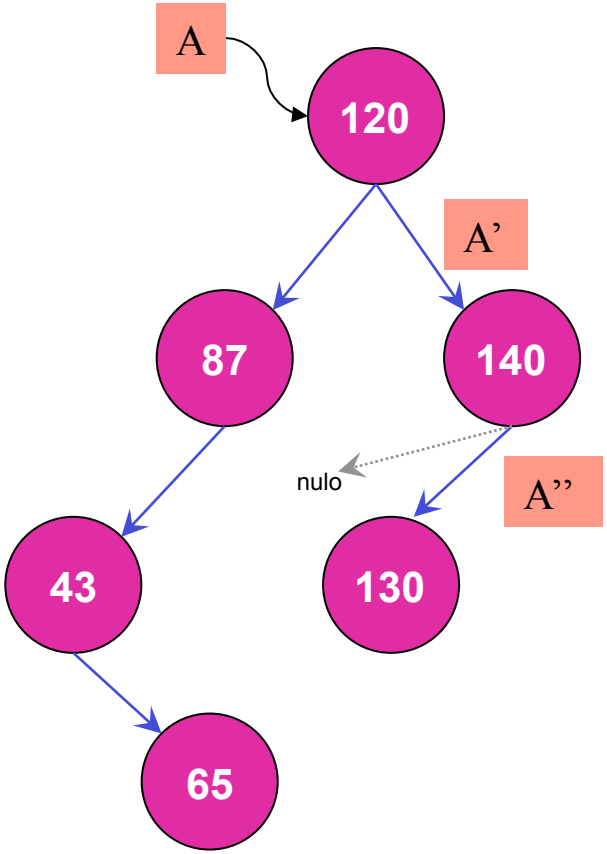

```
Function InsertarClave (var A:tArbolBin; clave:tClave): boolean;
begin
  if EsArbolVacio(A)
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

ABB: Insertar la clave 130




```
Function InsertarClave (var A:tArbolBin; clave:tClave): boolean;
begin
  if EsArbolVacio(A) (* fin recursividad *)
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

ABB: Insertar la clave 130




```
Function InsertarClave (var A:tArbolBin; clave:tClave): boolean;
begin
  if EsArbolVacio(A) (* fin recursividad *)
     InsertarClave:= CrearNodo (clave, A)
  else if clave = A^.clave then
     InsertarClave:= true
  else if clave < A^.clave then
     InsertarClave:= InsertarClave (A^.izdo, clave)
  else
     InsertarClave:= InsertarClave (A^.dcho, clave);
end;
```

ABB: Borrado (Pseudocódigo)

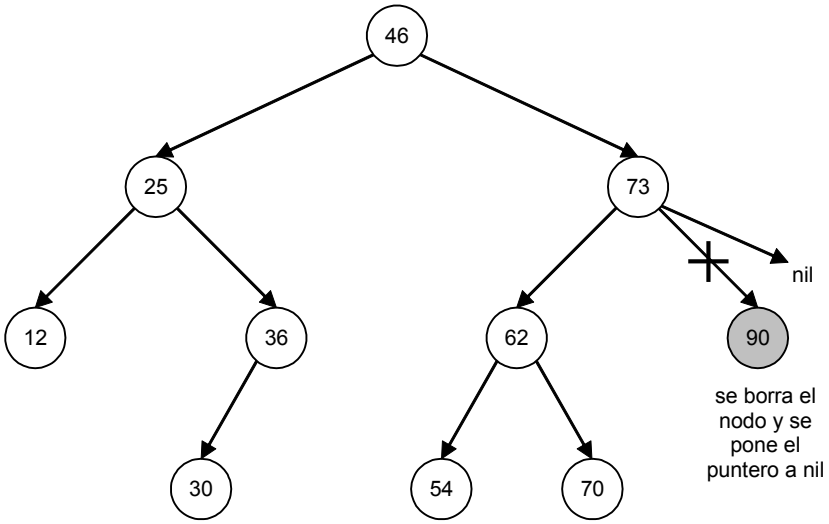
- Se busca en el árbol la posición del nodo a eliminar
  - Si es un nodo hoja ⇒ se actualiza el puntero del padre a NULO y se borra el nodo
  - Si es un nodo con un solo hijo ⇒ se actualiza el puntero del padre para que apunte al hijo y se borra el nodo

ABB: Borrado (Pseudocódigo II)

- …
  - Si es un nodo con dos hijos ⇒ se sustituye su clave por:
    - la mayor contenida en su subárbol izquierdo (nodo anterior en el orden) o
    - la menor contenida en su subárbol derecho (nodo posterior en el orden)
  - …y se borra el nodo que la contenga
  - El nodo del subárbol izquierdo con mayor clave se caracteriza por ser un nodo sin hijos o con un hijo a la izquierda (si tuviera un hijo a la derecha ya no sería el mayor). Por lo tanto es un nodo sencillo de borrar
  - El nodo del subárbol derecho con menor clave se caracteriza por ser un nodo sin hijos o con un hijo a la derecha (si tuviera un hijo a la izquierda ya no sería el menor). Por lo tanto es un nodo sencillo de borrar
  - Al sustituir la clave de un nodo por la de su anterior (o posterior) la propiedad de árbol binario de búsqueda se sigue cumpliendo.

ABB: Borrado. Ejemplo

- Nodo sin hijos (90) o con un hijo (36)



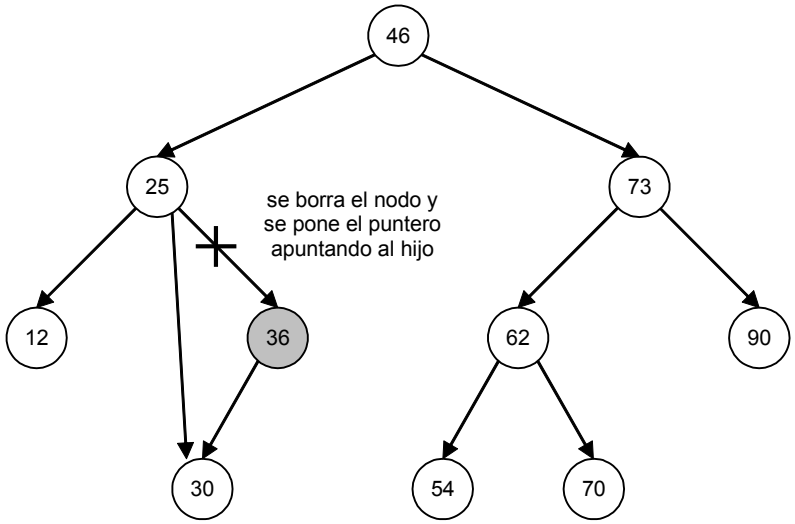


ABB: Borrado. Ejemplo

- Nodo con dos hijos (46)

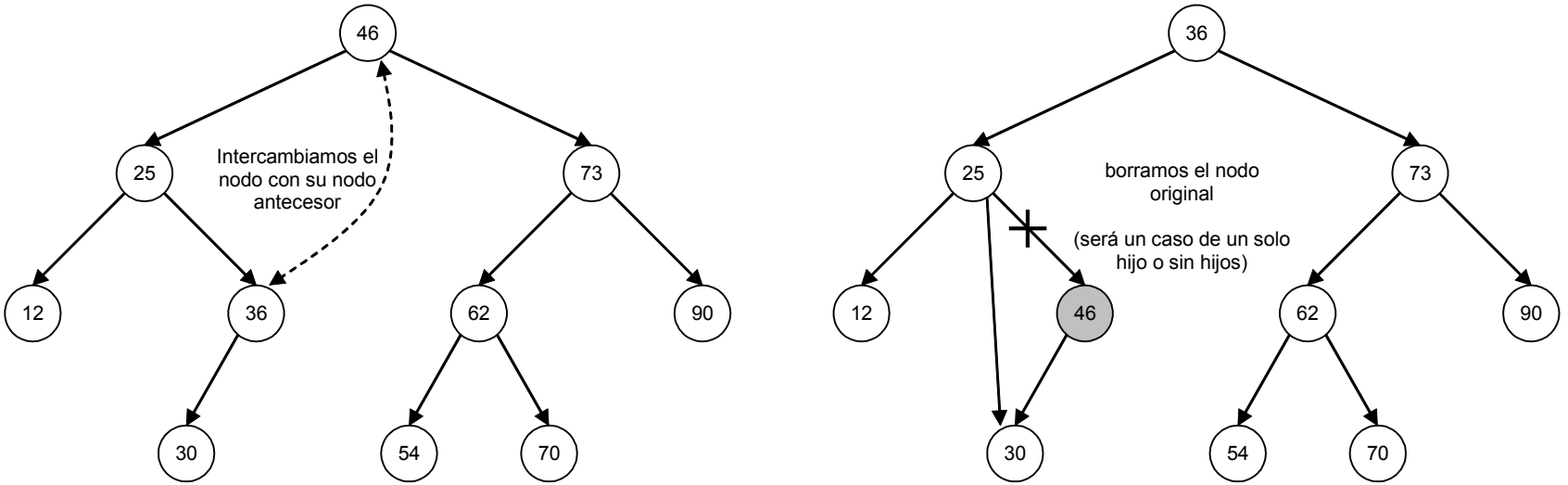


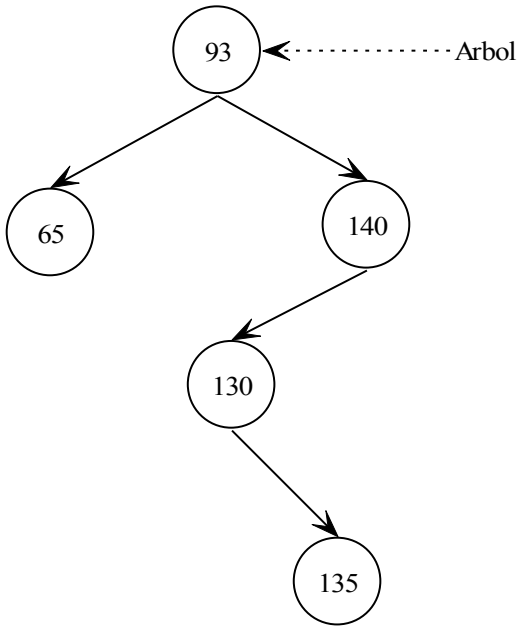
ABB: Borrado (Código)

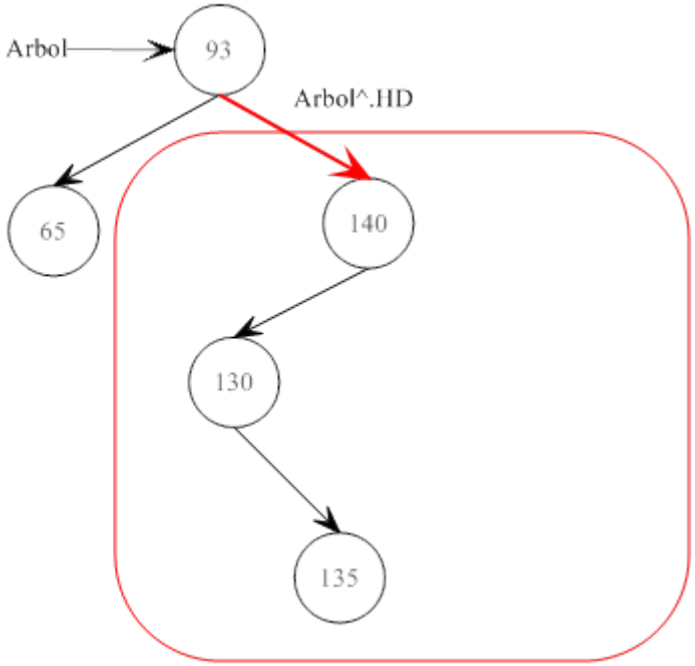
```
Procedure EliminarClave(var Arbol: tArbolBin; clave: tClave);
{Precondición: La clave existe en el árbol}
var
   aux: tArbolBin;

procedure reemplazar (var ArbolAux: tArbolBin);   // Reemplaza el nodo por su anterior
begin
   if not EsArbolVacio (ArbolAux^.dcho) then
      reemplazar (ArbolAux^.dcho)                 // Bajamos por la rama derecha
   else begin
      Aux^.clave:= ArbolAux^.clave;;              // Reemplazamos los campos de información
      aux:= ArbolAux;                             // Marcamos el nodo sobre el que hacer dispose
      ArbolAux:= ArbolAux^.izdo                   //    y reenlazamos la estructura "saltando"
   end;                                           //    al nodo que se va a eliminar
end;

begin
   if clave < Arbol^.clave then                   // Si la clave es menor borrar en subárbol izqdo
      EliminarClave (Arbol^.izdo, clave)
   else if clave > Arbol^.clave then              // Si la clave es mayor borrar en subárbol dcho
      EliminarClave (Arbol^.dcho, clave)
   else begin                                     // Si la clave es igual borramos el nodo
      aux:= Arbol;
      if EsArbolVacio (Arbol^.dcho) then          // Si no tiene hijo dcho sustituir por el izqdo
         Arbol:= Arbol^.izdo                      //    (incluye el caso de los dos hijos vacios)
      else if EsArbolVacio (Arbol^.izdo) then     // Si no tiene hijo izqdo sustituir por el dcho
         Arbol:= Arbol^.dcho                      // Si tiene dos hijos llamamos a reemplazar
      else reemplazar (Arbol^.izdo);              //    pasando como parámetro su subarbol izqdo
      dispose(aux);
   end;
end;
```

ABB. Borrado con un descendiente (140)

EliminarClave (Arbol, “140”)

```
..
else if clave > Arbol^.clave then {140>93}
   EliminarClave (Arbol^.dcho, clave);
else...
```

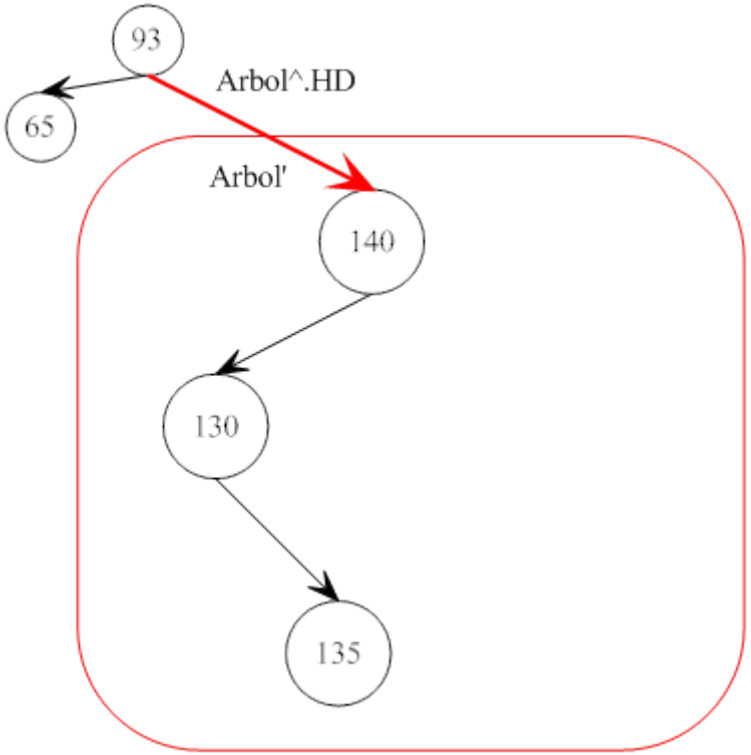




NOTA: Donde dice Arbol^.HD debería decir Arbol^.dcho

ABB. Borrado con un descendiente (140)

EliminarClave (Arbol', "140")

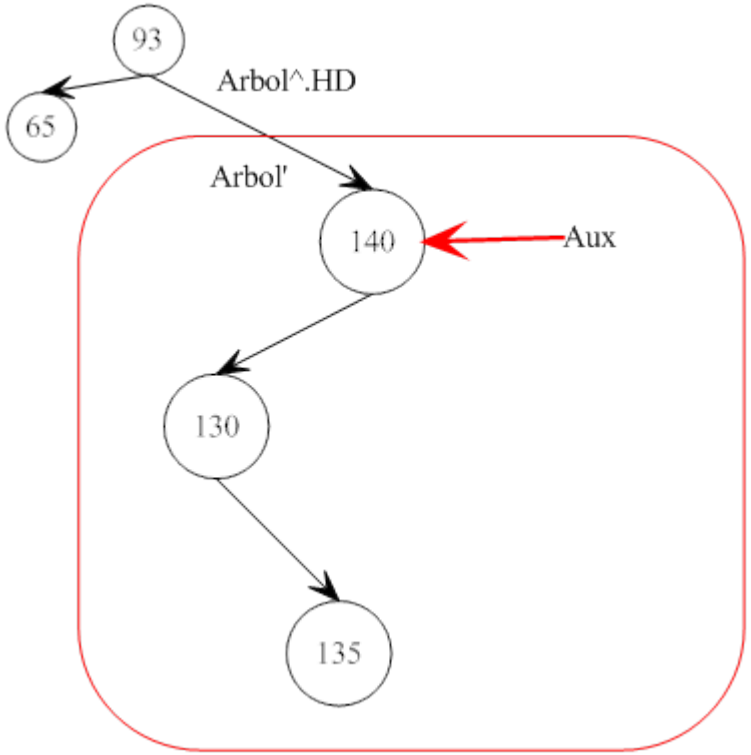


NOTA: Donde dice Arbol^.HD debería decir Arbol^.dcho

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con un descendiente (140)

EliminarClave (Arbol', "140")

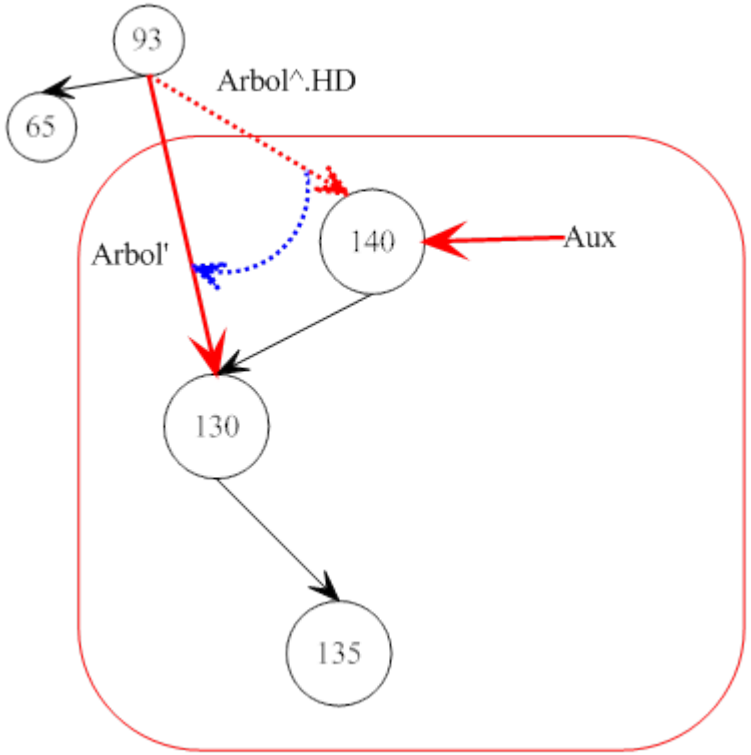


NOTA: Donde dice Arbol^.HD debería decir Arbol^.dcho

```
...
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con un descendiente (140)

EliminarClave (Arbol', "140")

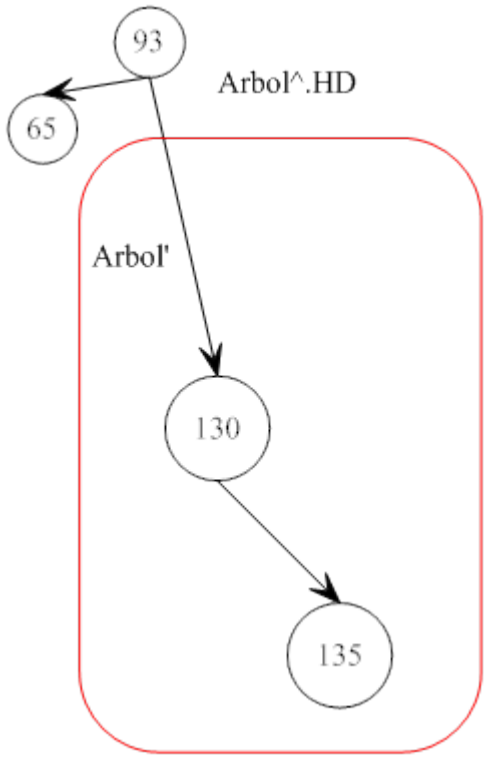


NOTA: Donde dice Arbol^.HD debería decir Arbol^.dcho

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con un descendiente (140)

EliminarClave (Arbol', "140")



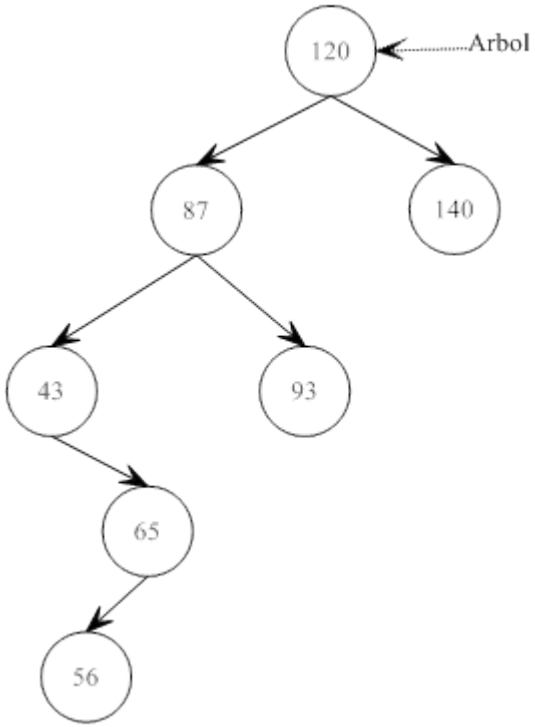

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

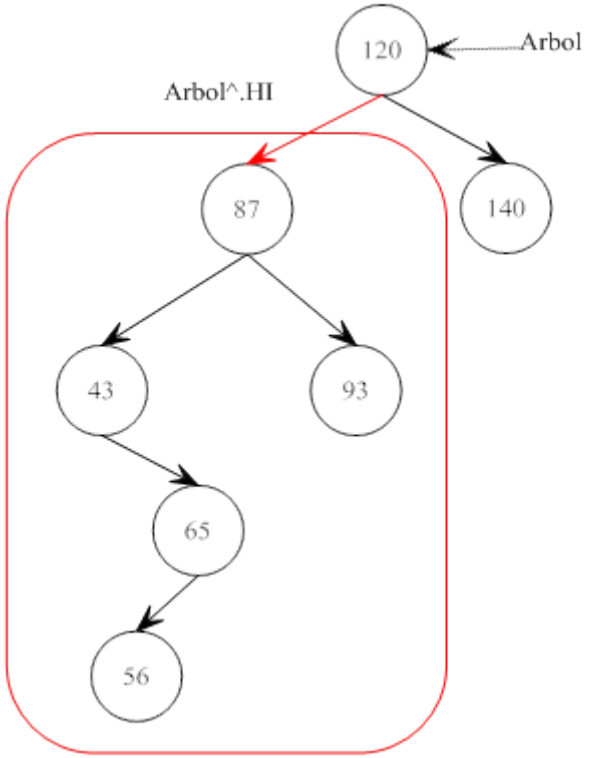
NOTA: Donde dice Arbol^.HD debería decir Arbol^.dcho

ABB. Borrado con dos descendientes (87)

EliminarClave (Arbol, “87”)

```
if clave < Arbol^.clave {87<120} then
   Eliminar (Arbol^.izdo, clave)
else...
```

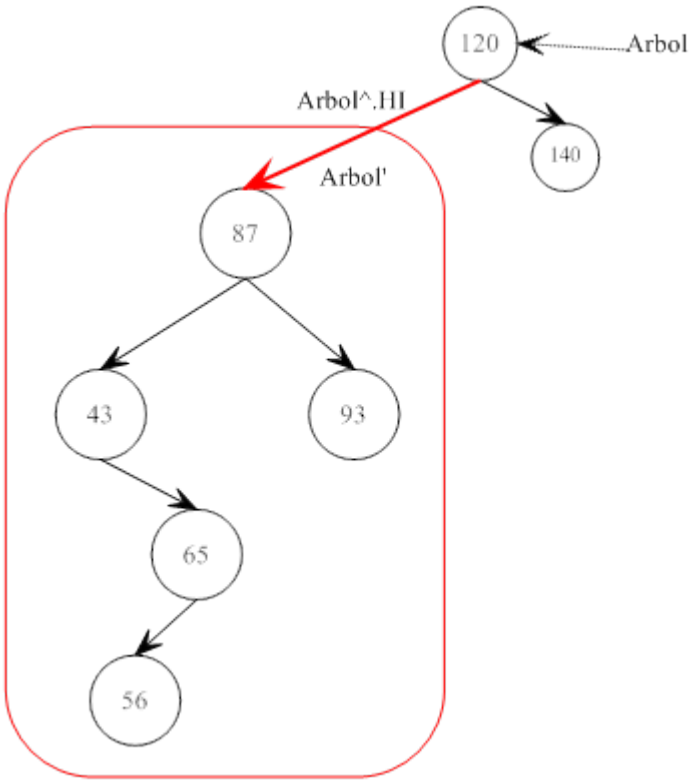




NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo

ABB. Borrado con dos descendientes (87)

EliminarClave (Arbol', "87")

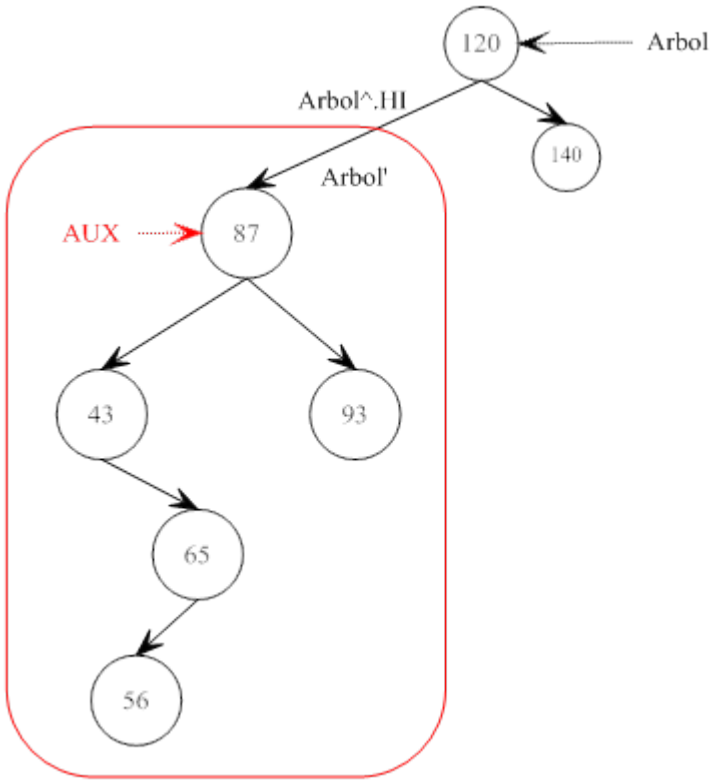


NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con dos descendientes (87)

EliminarClave (Arbol', "87")

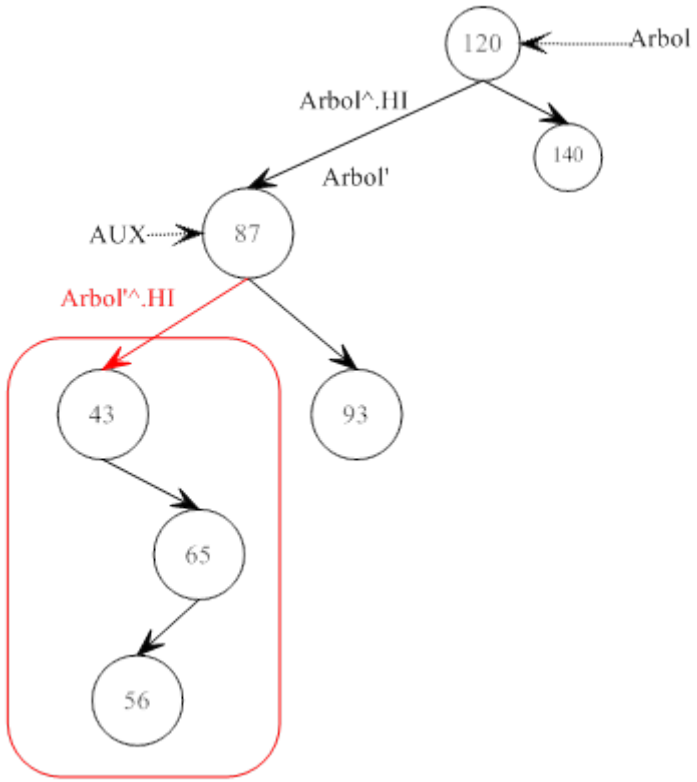


NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con dos descendientes (87)

EliminarClave(Arbol', "87")

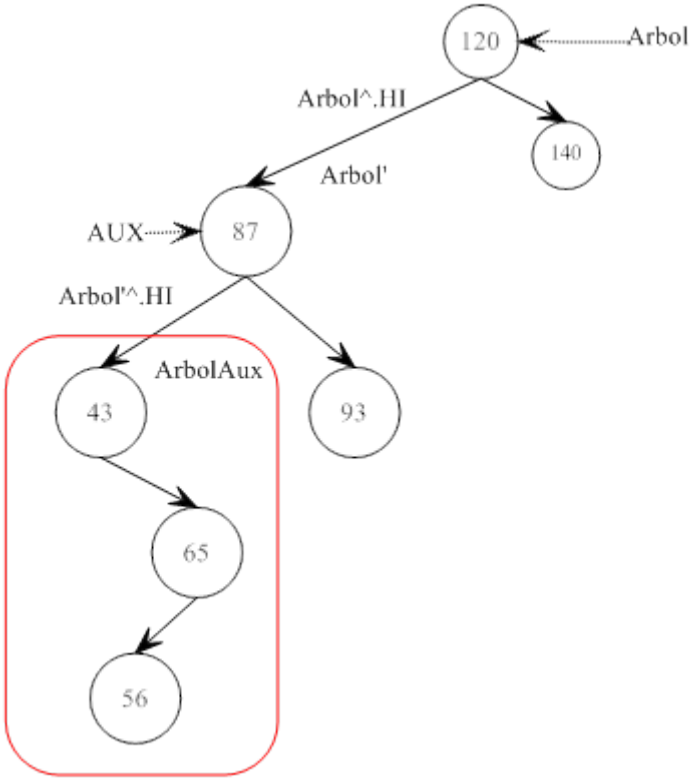


NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo

```
…
{Localizado el elemento a borrar}
Aux:= Arbol;

{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

ABB. Borrado con dos descendientes (87)

Reemplazar (Arbol'')

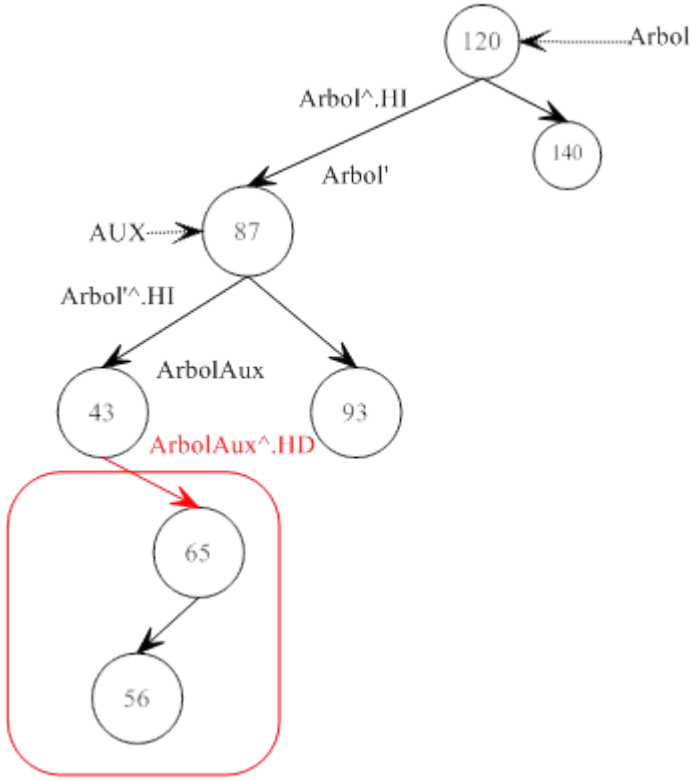


NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo

```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.info:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

ABB. Borrado con dos descendientes (87)

Reemplazar (ArbolAux)

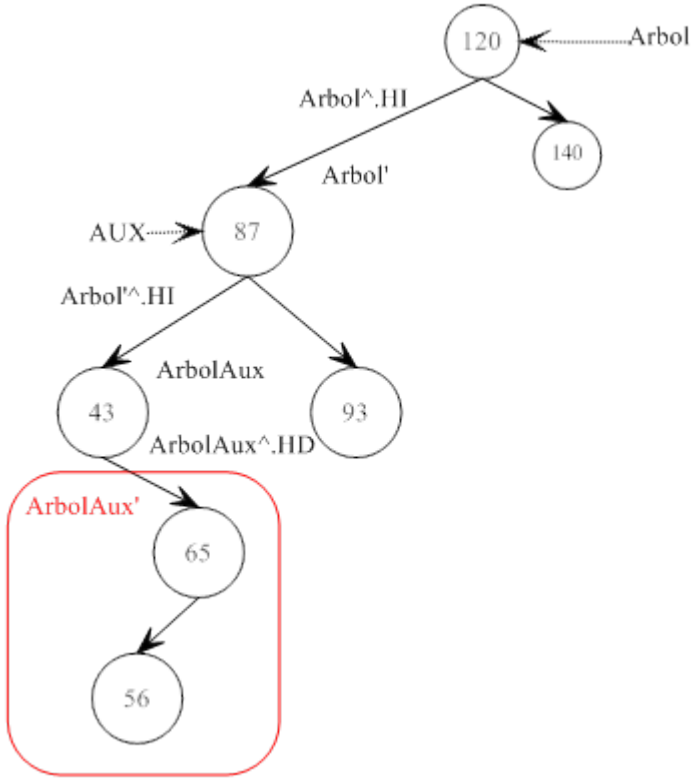



```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.clave:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

Reemplazar (ArbolAux')

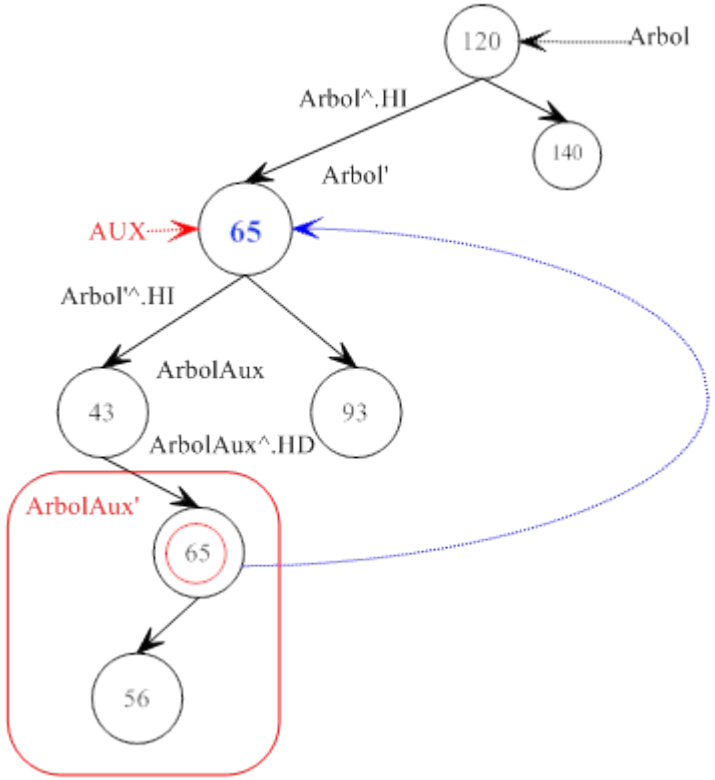



```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.clave:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

Reemplazar (ArbolAux')

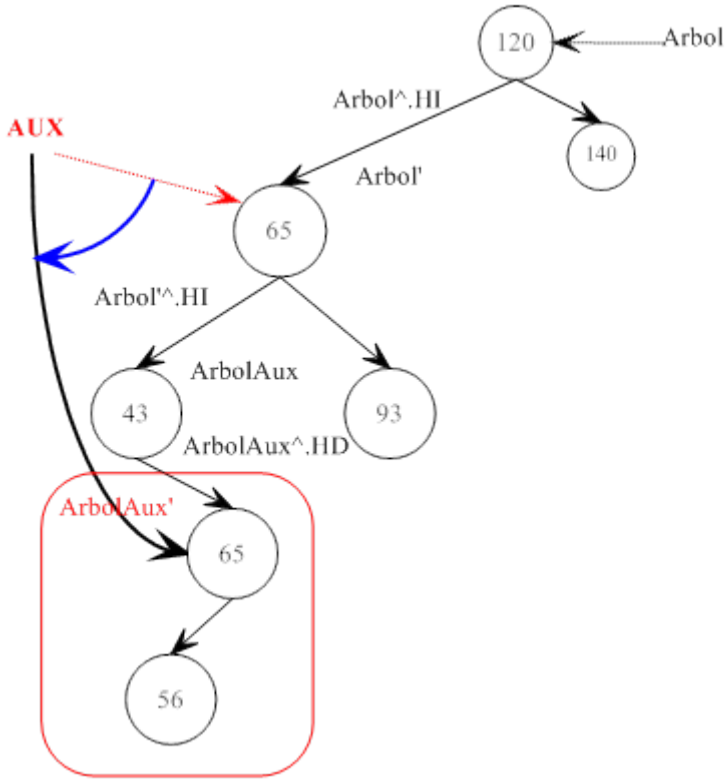



```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.clave:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

Reemplazar (ArbolAux')

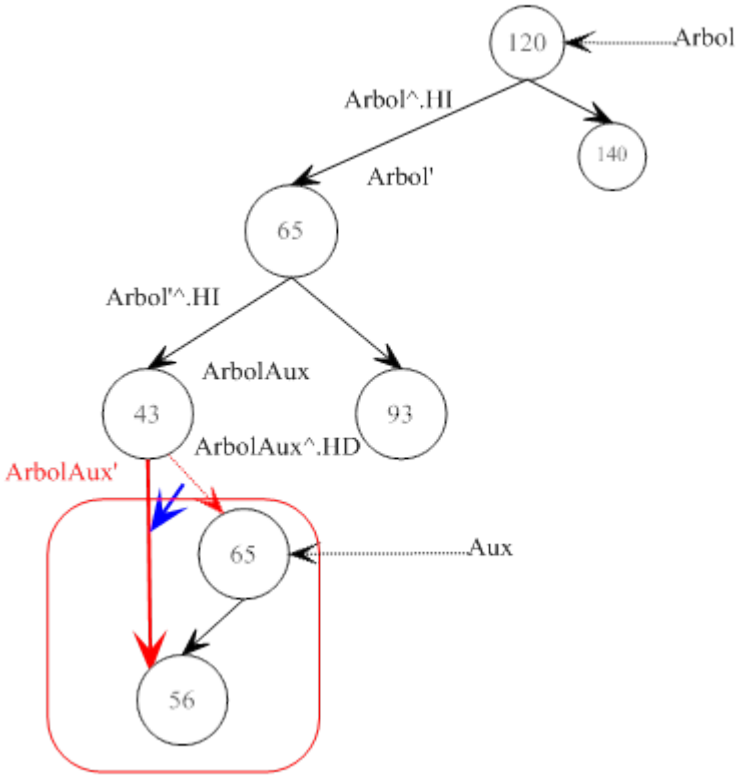



```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.clave:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

Reemplazar (ArbolAux')

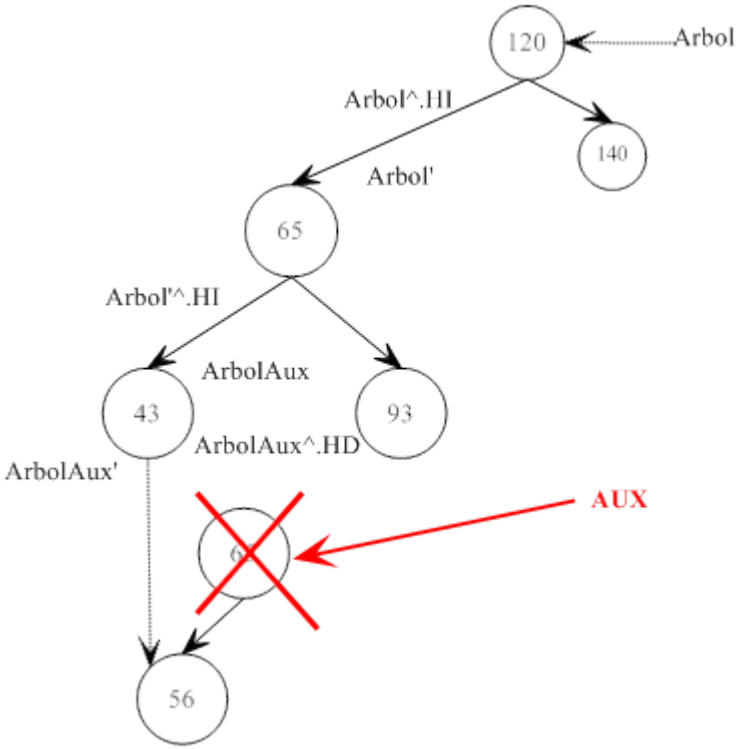



```
procedure reemplazar (var ArbolAux: tArbolBin);
begin
  if not EsArbolVacio (ArbolAux^.dcho) then
     reemplazar (ArbolAux^.dcho)
  else begin
     aux^.clave:= ArbolAux^.clave;
     aux:= ArbolAux;
     ArbolAux:= ArbolAux^.izdo
  end;
end;
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

Reemplazar (Arbol'')




```
...
{Localizado el elemento a borrar}
Aux:= Arbol;


{Mirar si tiene algún descendiente}
if EsArbolVacio(Arbol^.izdo) then
   Arbol:= Arbol^.dcho
else if EsArbolVacio(Arbol^.dcho) then
   Arbol:= Arbol^.izdo
else reemplazar (Arbol^.izdo);
dispose(Aux);
```

NOTA: Donde dice Arbol^.HI debería decir Arbol^.izdo (y donde dice Arbol^.HD debería decir Arbol^.dcho)

ABB. Borrado con dos descendientes (87)

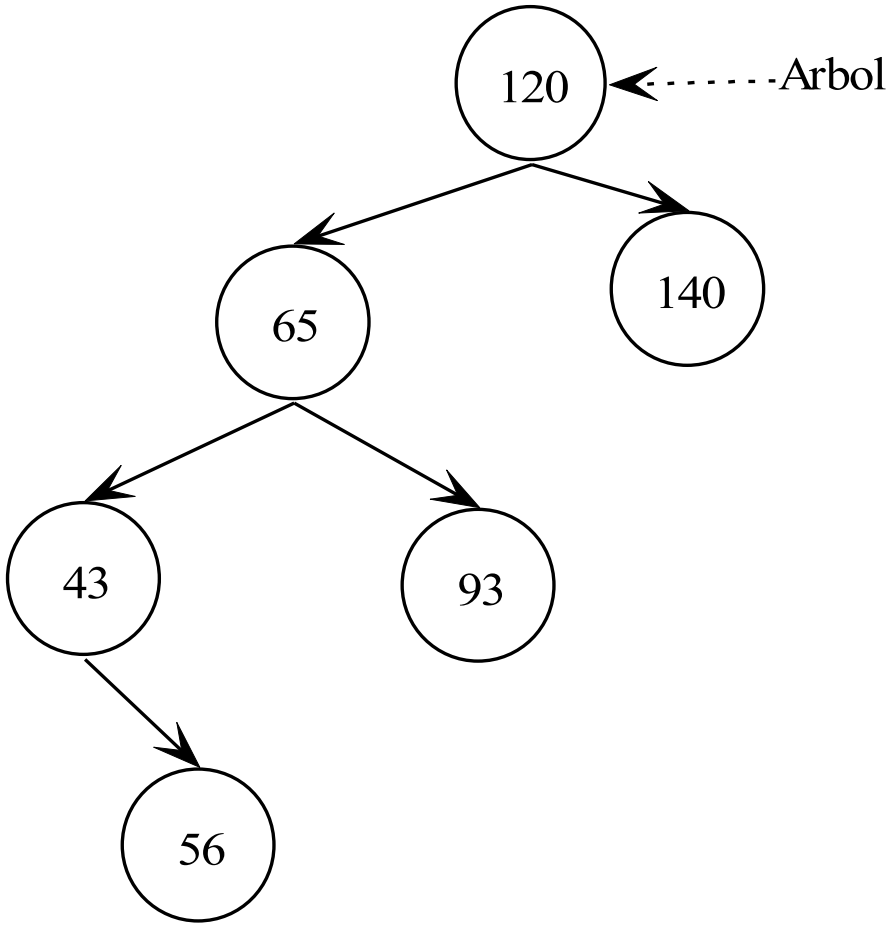


ABB: Ejemplo

- Insertamos: 55 – 23 – 72 – 45 – 87 – 35 – 69 – 58 – 50 – 48

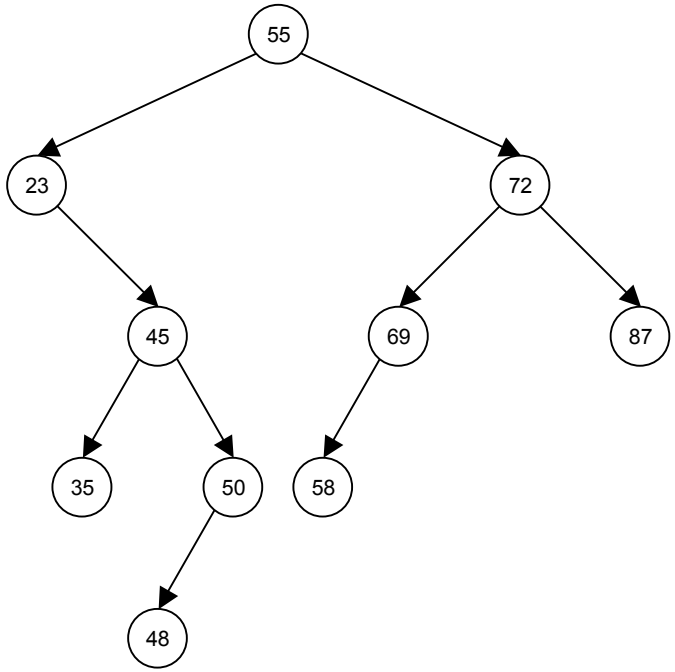
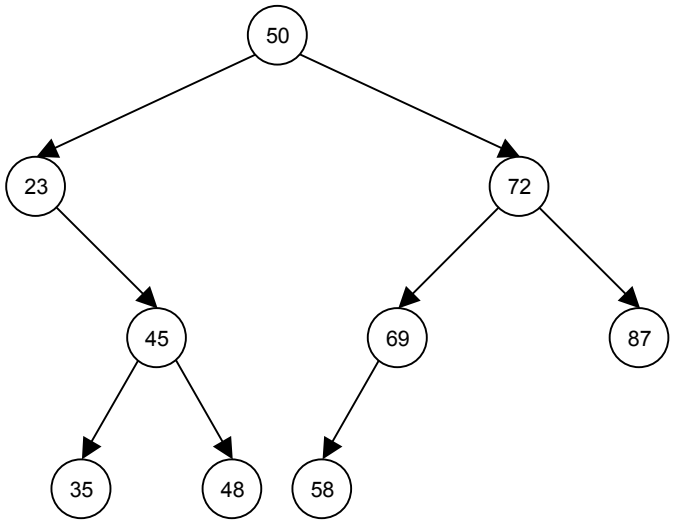


ABB: Ejemplo (II)

- Borramos: 55



- Borramos: 72

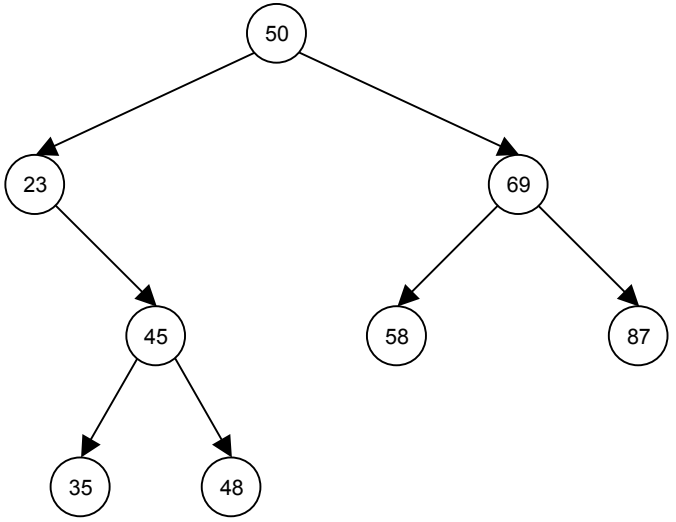
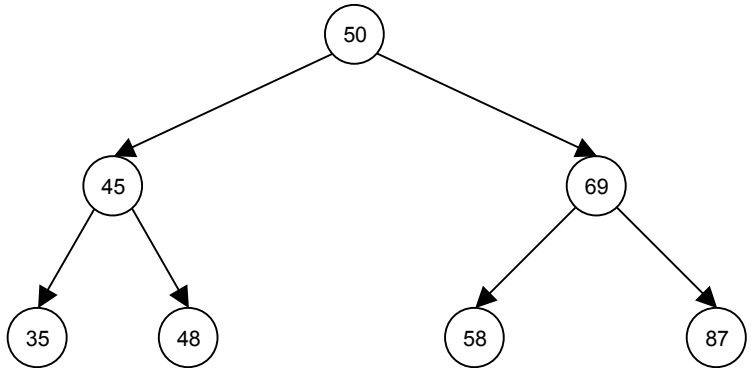


ABB: Ejemplo (III)

- Borramos: 23



- Borramos: 35