Árboles Binarios de Búsqueda

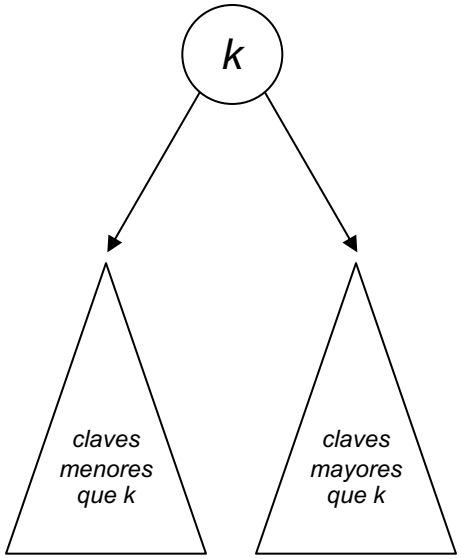
Programación II

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

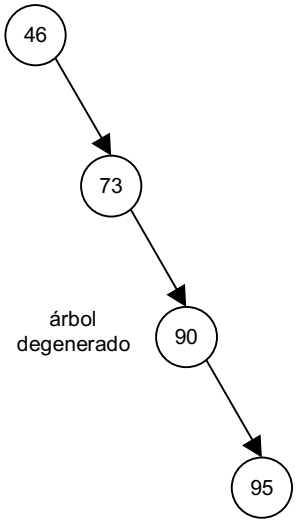


ABB: Búsqueda de una clave

- 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

```
tBST findKey(tBST tree, tKey key)
{
   if (isEmptyTree(tree))
      return NULLBST;
   else if (key == tree->key)
      return tree;
   else if (key < tree->key)
      return findKey(tree->left, key);
   else // (key > tree->key)
      return findKey(tree->right, key);
}
```

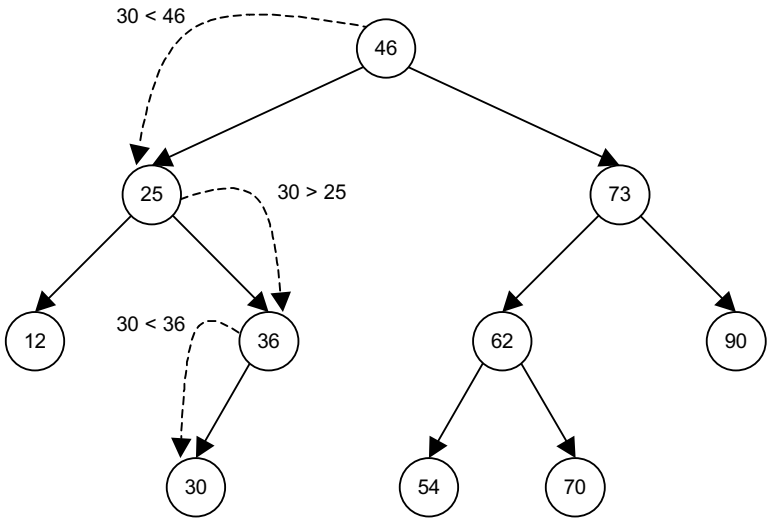


ABB: Inserción de una clave

- Pseudocódigo
  - Se compara la clave a insertar con la raíz del árbol
    - Si el árbol está vacío 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

```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

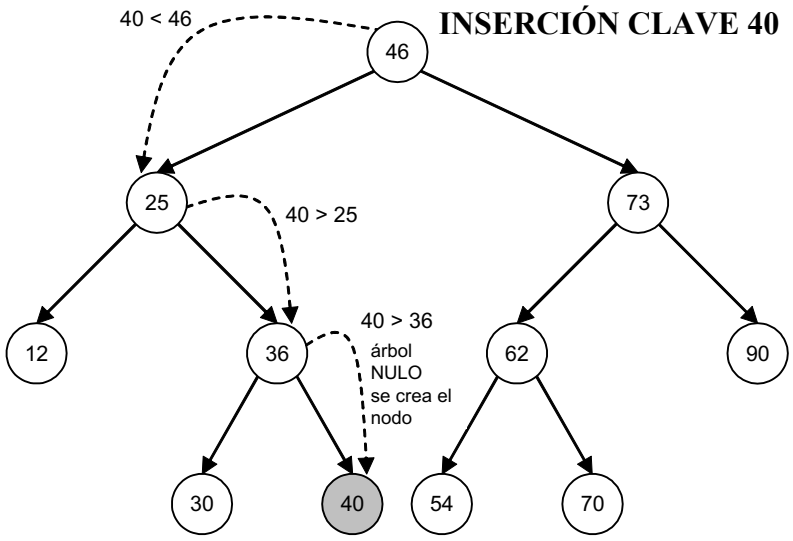
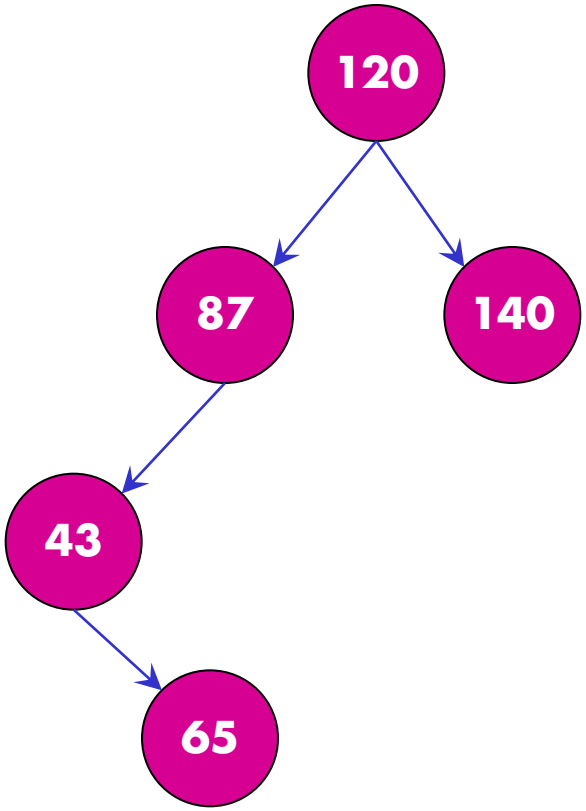
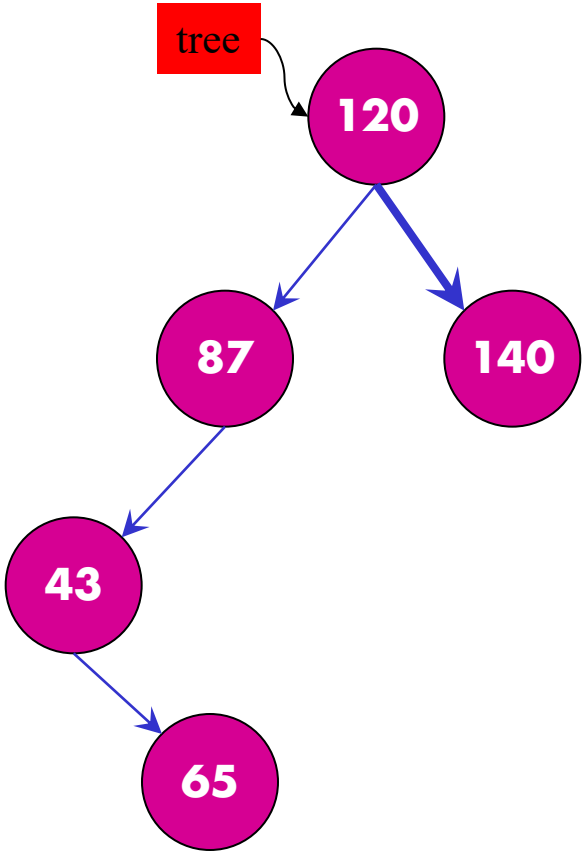


ABB: Insertar la clave 130



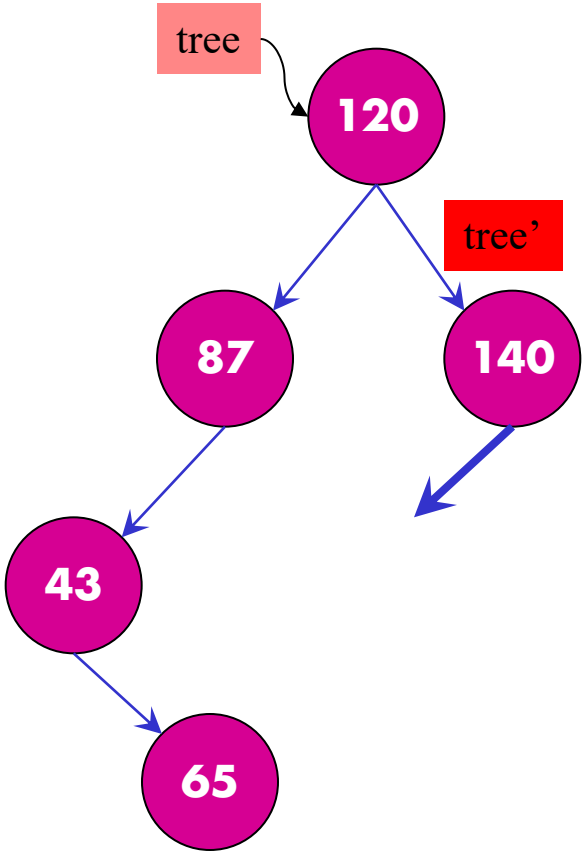

```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

ABB: Insertar la clave 130



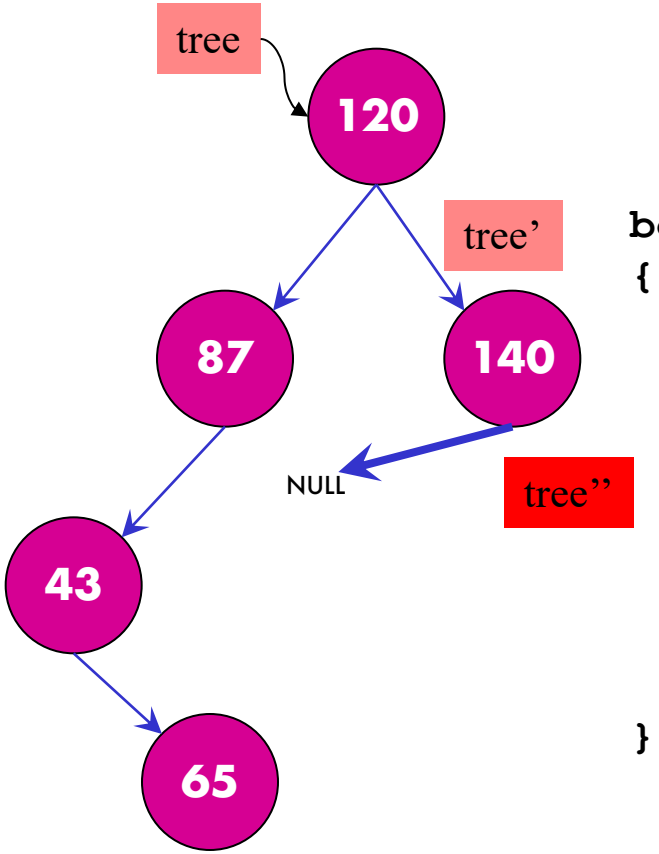

```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

ABB: Insertar la clave 130



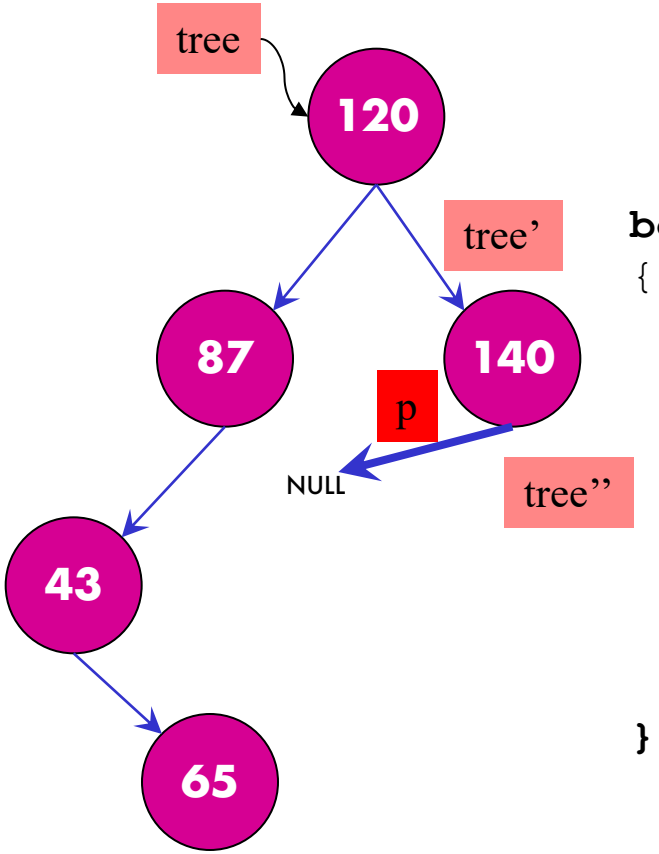

```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

ABB: Insertar la clave 130



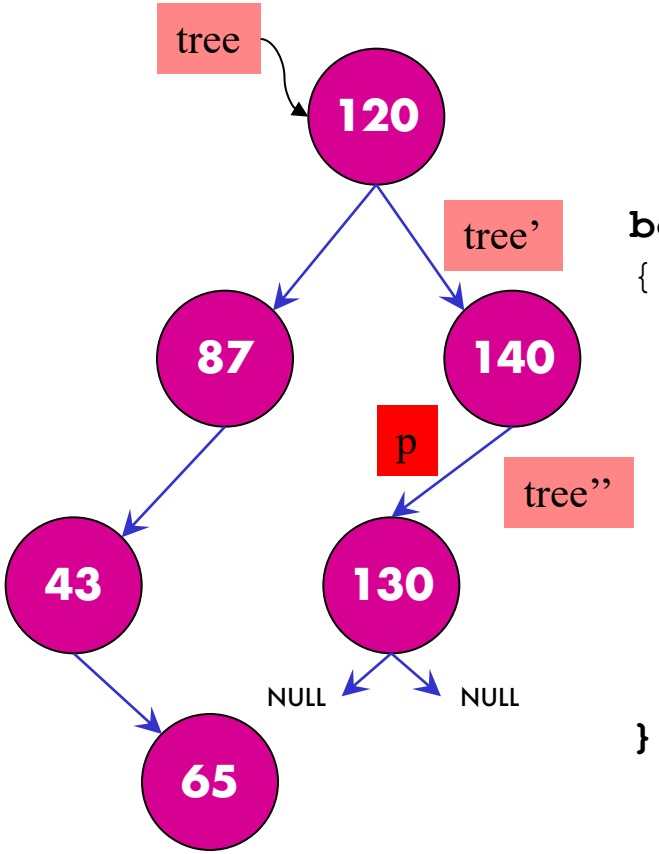

```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

ABB: Insertar la clave 130



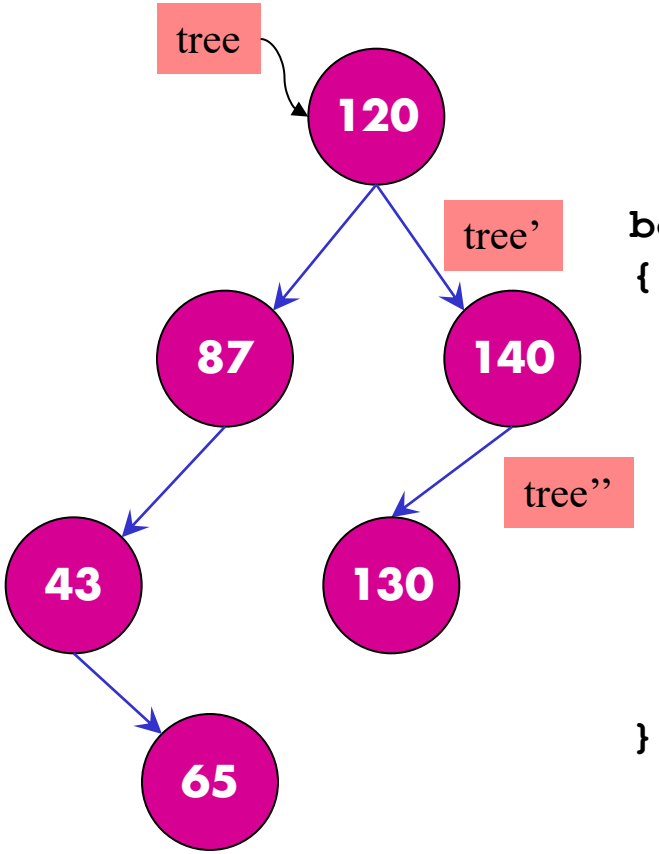

```
bool createBSTNode(tBSTPos* p, tKey key)
{
    *p = malloc(sizeof(struct tBSTNode));
    if (*p!=NULLBST)
    {
        (*p)->key = key;
        (*p)->left = NULLBST;
        (*p)->right = NULLBST;
    }
    return *p!=NULLBST;
}
```

ABB: Insertar la clave 130




```
bool createBSTNode(tBSTPos* p, tKey key)
{
    *p = malloc(sizeof(struct tBSTNode));
    if (*p!=NULLBST)
    {
        (*p)->key = key;
        (*p)->left = NULLBST;
        (*p)->right = NULLBST;
    }
    return *p!=NULLBST;
}
```

ABB: Insertar la clave 130




```
bool insertKey(tBST* tree, tKey key)
{
   if (isEmptyTree(*tree))
      return createBSTNode(tree, key);
   else if (key == (*tree)->key)
      return true;
   else if (key < (*tree)->key)
      return insertKey(&(*tree)->left, key);
   else // (key > (*tree)->key)
      return insertKey(&(*tree)->right, key);
}
```

ABB: Borrado (Pseudocódigo)

- Si el nodo a eliminar es HOJA ⇒ se actualiza el puntero del padre a NULO y se borra el nodo hoja
- Si el nodo a eliminar sólo tiene UN HIJO ⇒ se actualiza el puntero del padre para que apunte al hijo y se borra el nodo
- Si el nodo a eliminar tiene DOS HIJOS⇒ …

ABB: Borrado (Pseudocódigo)

- Si el nodo a eliminar tiene DOS HIJOS:
  - se sustituye su clave por la clave anterior por orden, es decir, la mayor de entre sus hijos menores (subárbol izquierdo)
  - se borra el nodo que la contenga

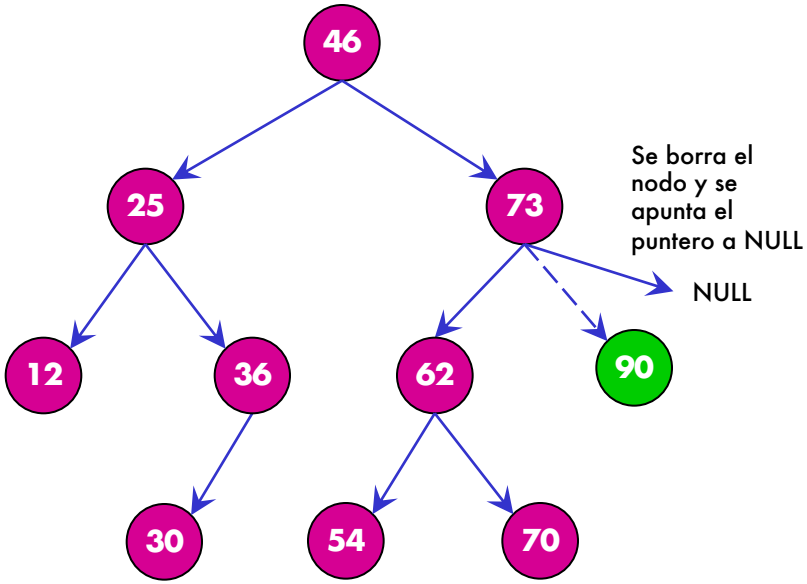
- El nodo del subárbol izquierdo con mayor clave es un nodo hoja o con un hijo a la izquierda (si tuviera un hijo a la derecha ya no sería el mayor). Por lo tanto es un nodo sencillo de borrar [1]

- Al sustituir la clave de un nodo por la de su anterior la propiedad de árbol binario de búsqueda se sigue cumpliendo.

[1] Otra posibilidad sería sustituir la clave a eliminar por la menor contenida entre los hijos mayores (subárbol derecho). El nodo con esta clave se caracteriza por ser un hoja o con un hijo a la derecha (si tuviera un hijo a la izquierda ya no sería el menor).

ABB: Borrado. Ejemplo

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



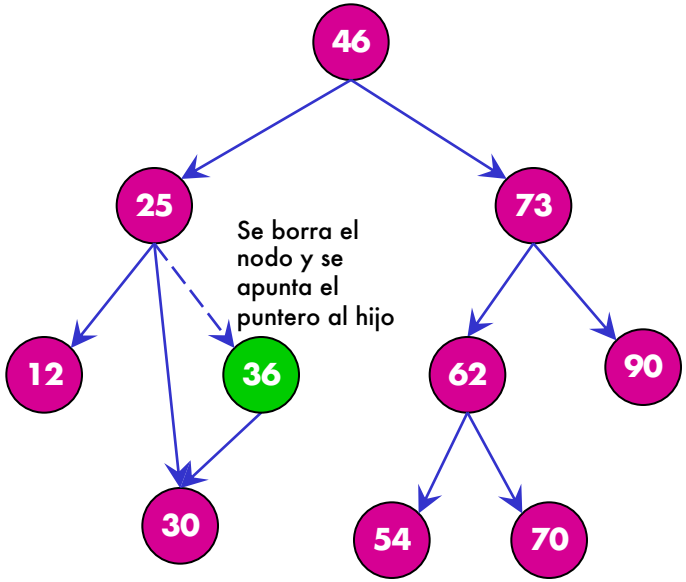


ABB: Borrado. Ejemplo

- Nodo con dos hijos (46)

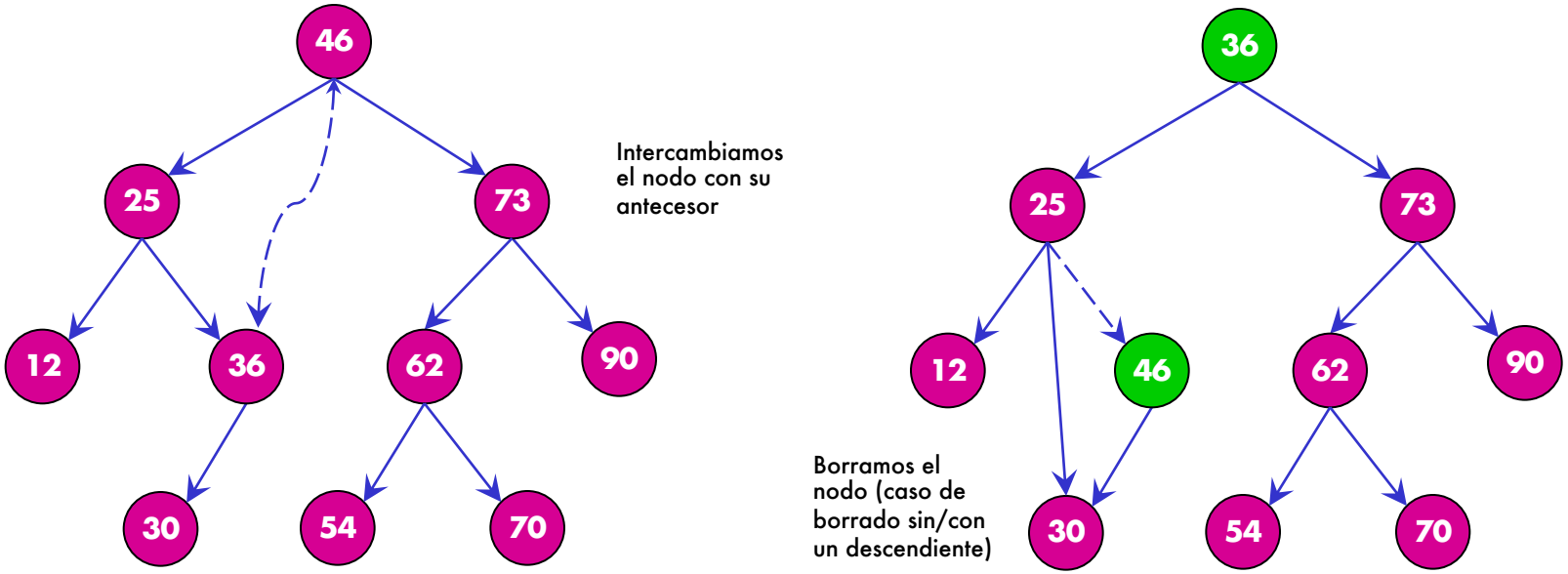


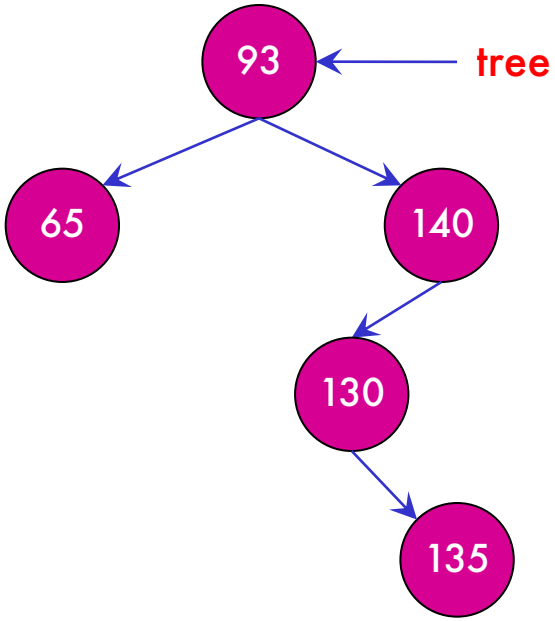
ABB: Borrado (Código)

```
// Auxiliary function for removeKey
void replace(tBST* subTree, tBST* auxTree)        // Replace the content of a node by its predecessors'
{
   if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);        // Going down the right branch
   else
   {
     (*auxTree)->key = (*subTree)->key;           // We replace the data fields of the node
     *auxTree = *subTree;                         // We mark the node on which we will call free()
     (*subTree) = (*subTree)->left;               // We re-link the tree structure by "skiping" the
   }                                              // node to be deleted
}

void removeKey(tBST* tree, tKey key)
{
   tBST aux;
   if (key < (*tree)->key)                        // If the key is smaller, continue in the left subtree
     removeKey(&(*tree)->left, key);
   else if (key > (*tree)->key)                   // If the key is larger, continue in the right subtree
     removeKey(&(*tree)->right, key);
   else                                           // Key located, proceed to delete the node
   {
     aux = *tree;
     if (isEmptyTree((*tree)->right))             // If it has no right child, replace by the left one
       *tree = (*tree)->left;                     // (this covers the no-children case, too)
     else if (isEmptyTree((*tree)->left))         // If it has no left child, replace by the right one
       *tree = (*tree)->right;
     else                                         // If it has two children, we call replace() passing
       replace(&(*tree)->left, &aux);             // its left subtree as first argument
     free(aux);
   }
}
```

ABB. Borrado con un descendiente (140)

removeKey (tree, 140)




```
...
if (key > (*tree)->key)   // 140>93
   removeKey(&(*tree)->right, key);
else
...
```

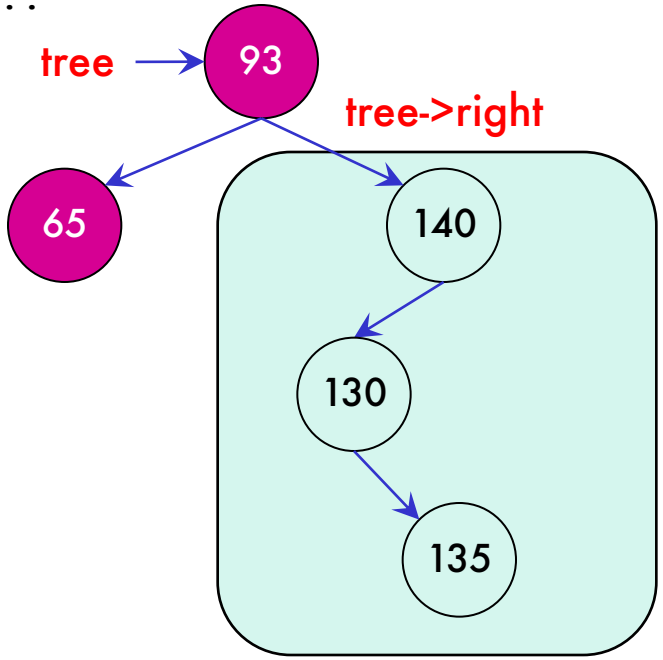
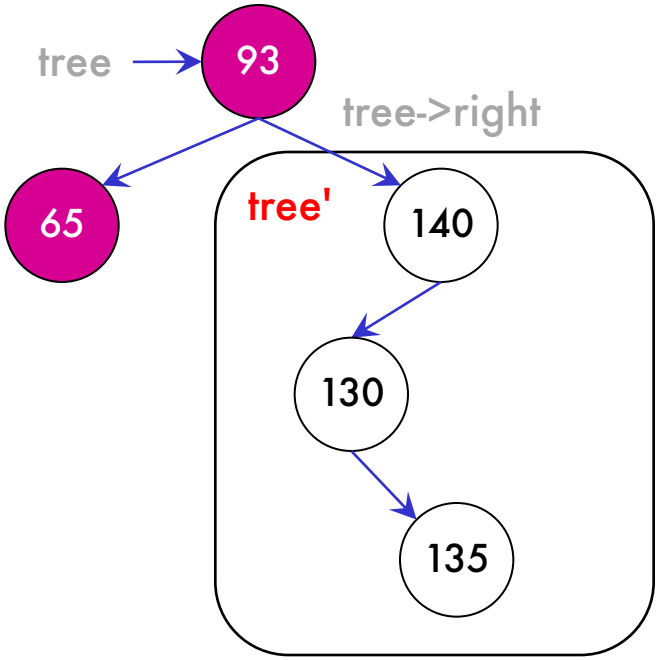


ABB. Borrado con un descendiente (140)

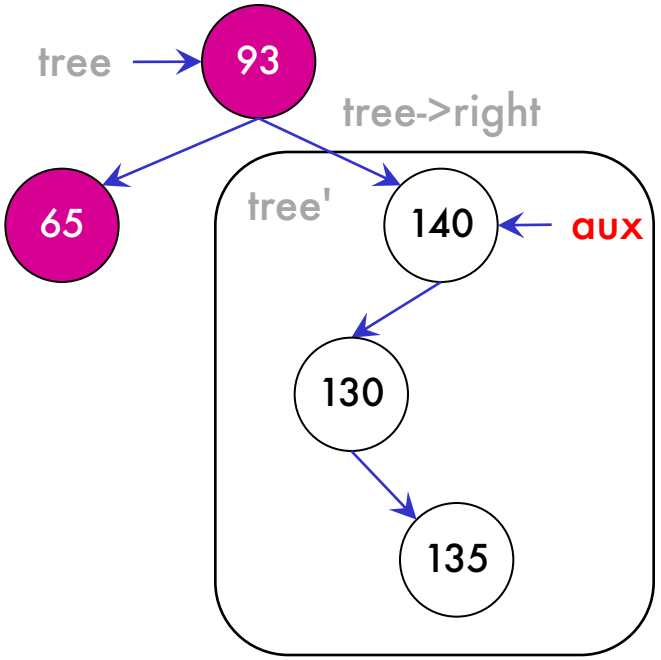
removeKey (tree', "140")




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con un descendiente (140)

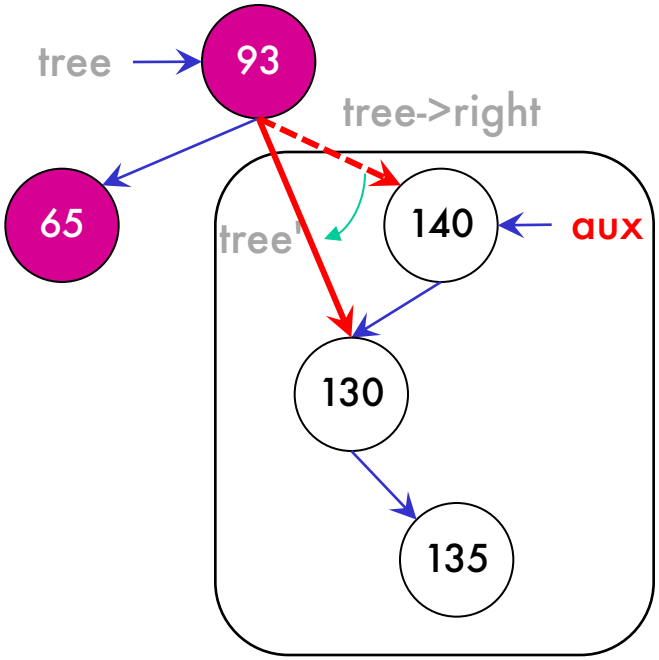
removeKey (tree', "140")




```
else
  {
   aux = *tree;
   if (isEmptyTree((*tree)->right))
     *tree = (*tree)->left;
   else if (isEmptyTree((*tree)->left))
     *tree = (*tree)->right;
   else
     replace(&(*tree)->left, &aux);
   free(aux);
  }
```

ABB. Borrado con un descendiente (140)

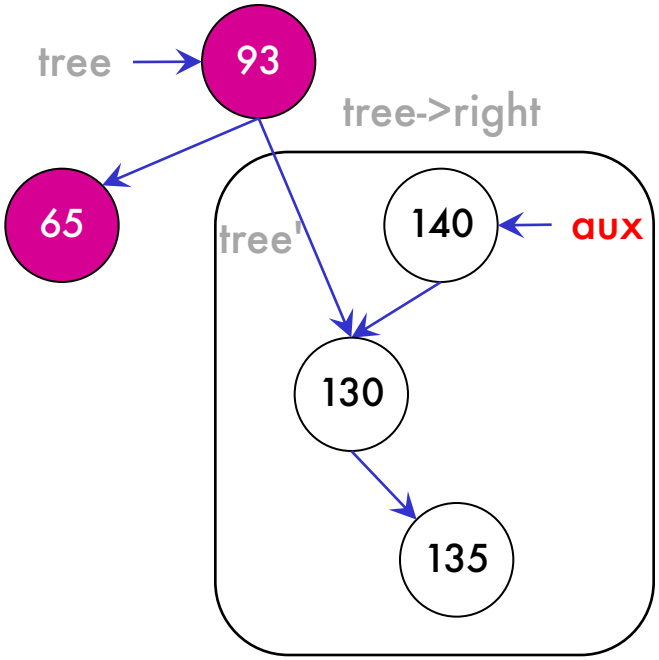
removeKey (tree', "140")




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con un descendiente (140)

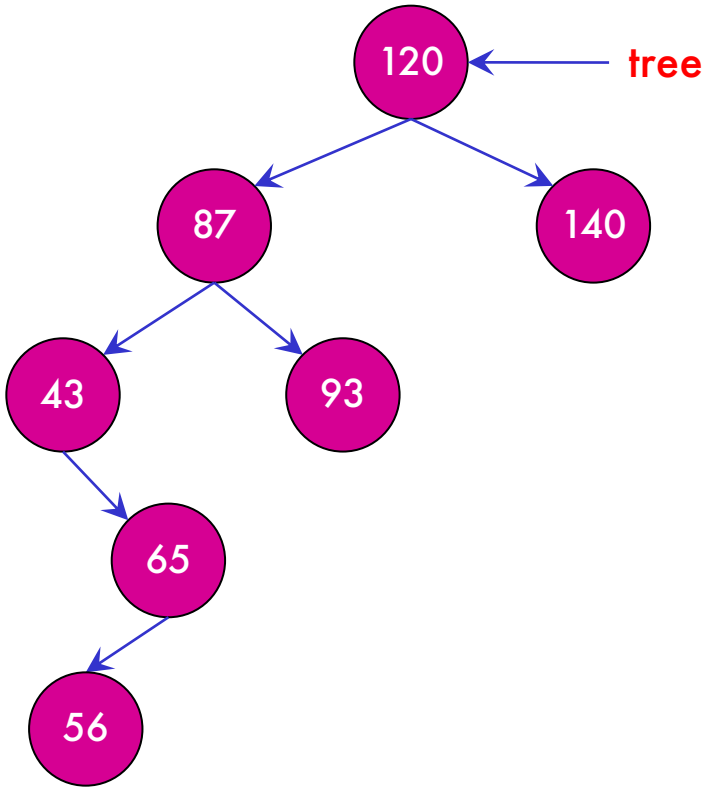
removeKey (tree', "140")




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con dos descendientes (87)

removeKey (tree, 87)




```
...
if (key < (*tree)->key)    // 87<120
     removeKey(&(*tree)->left, key);
else
...
```

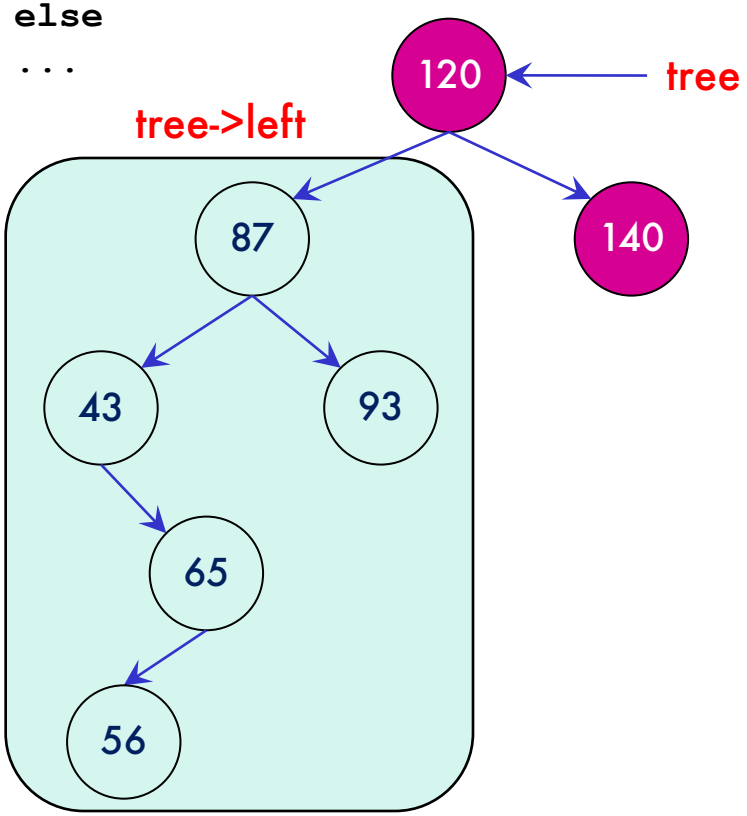
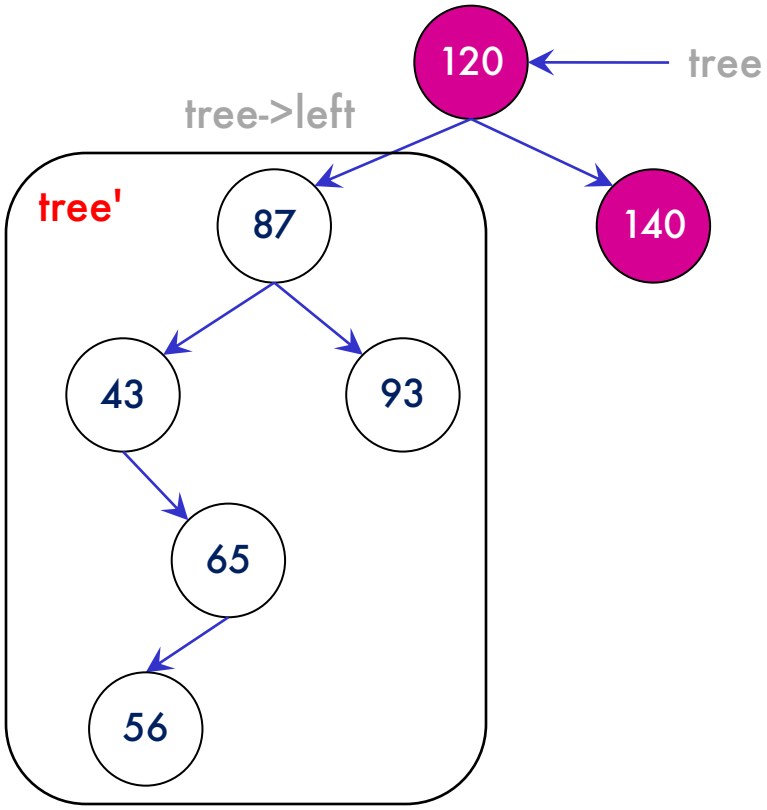


ABB. Borrado con dos descendientes (87)

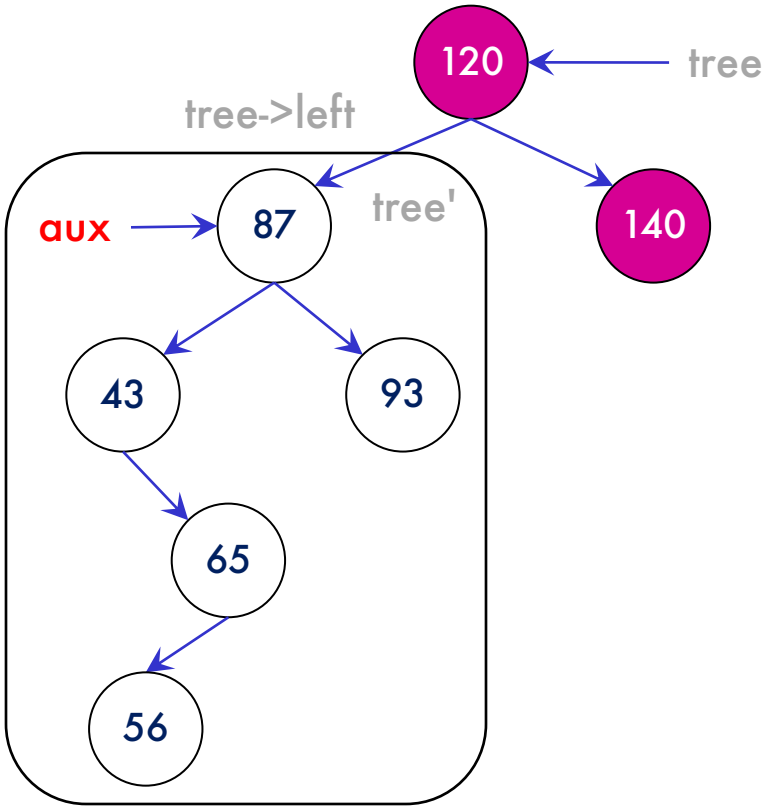
removeKey (tree', 87)




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con dos descendientes (87)

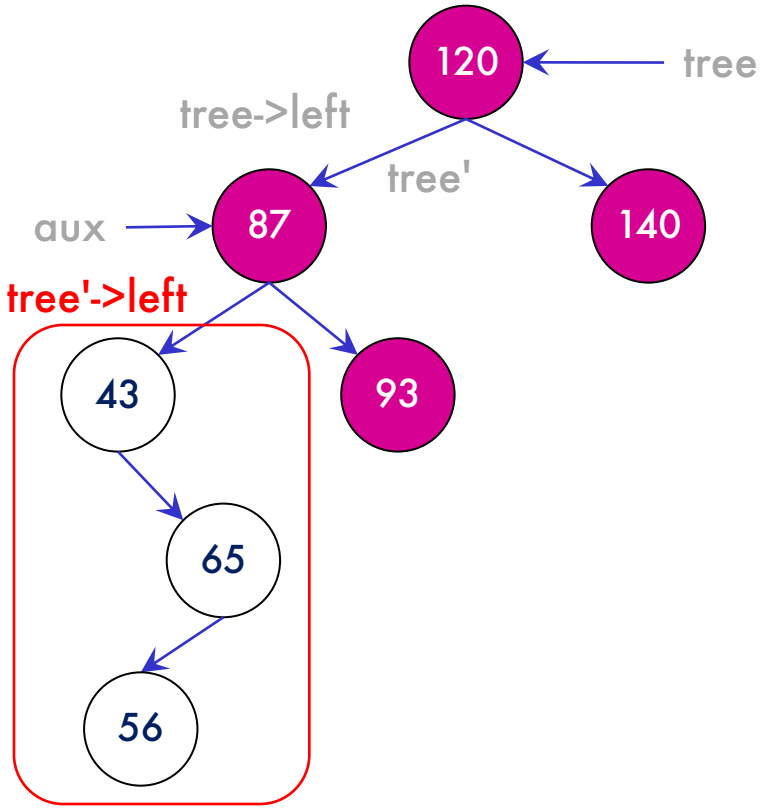
removeKey (tree', 87)




```
else
  {
   aux = *tree;
   if (isEmptyTree((*tree)->right))
     *tree = (*tree)->left;
   else if (isEmptyTree((*tree)->left))
     *tree = (*tree)->right;
   else
     replace(&(*tree)->left, &aux);
   free(aux);
  }
```

ABB. Borrado con dos descendientes (87)

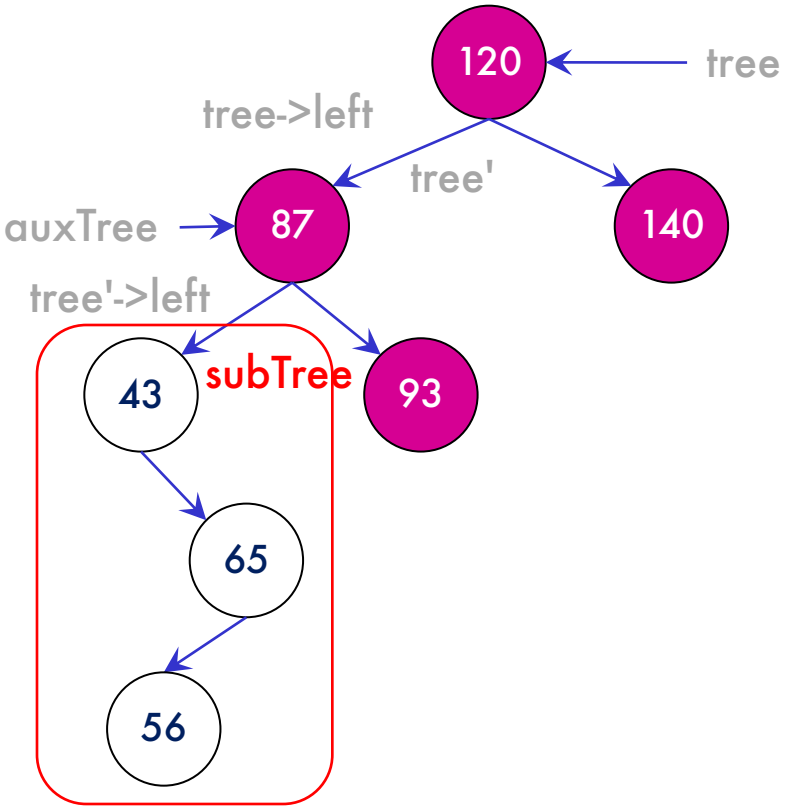
removeKey (tree', 87)




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con dos descendientes (87)

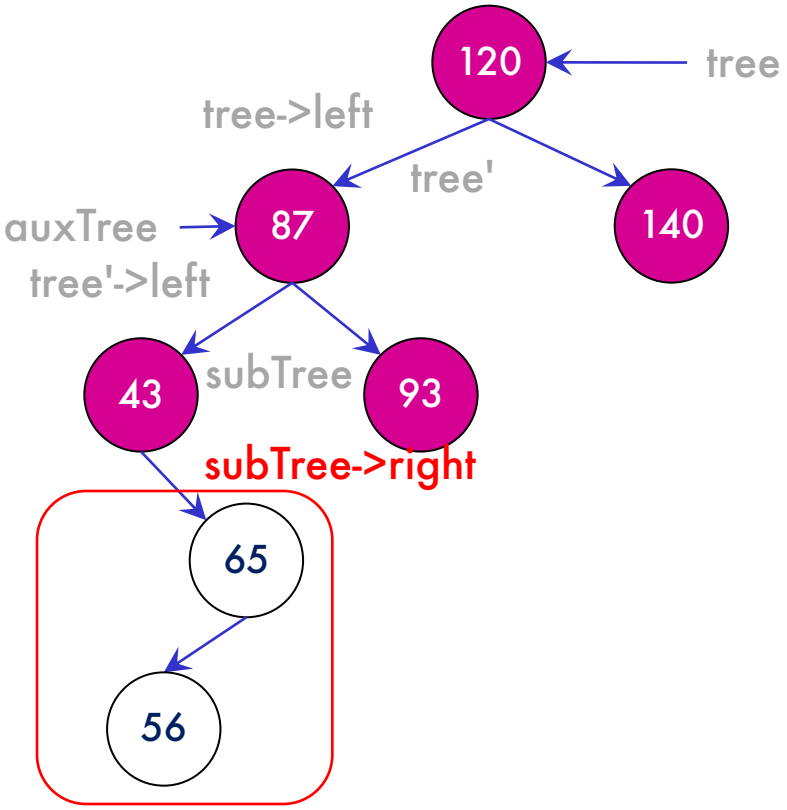
replace (tree'', auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);
  else
  {
     (*auxTree)->key = (*subTree)->key;
     *auxTree = *subTree;
      (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

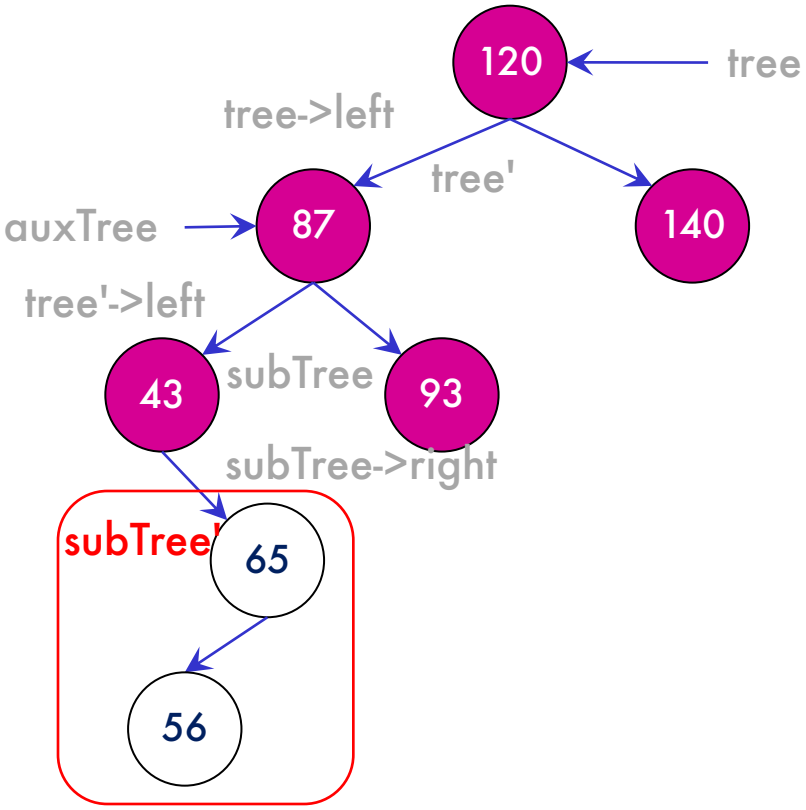
replace (subTree->rigth, auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
    replace(&(*subTree)->right, auxTree);
  else
  {
    (*auxTree)->key = (*subTree)->key;
    *auxTree = *subTree;
     (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

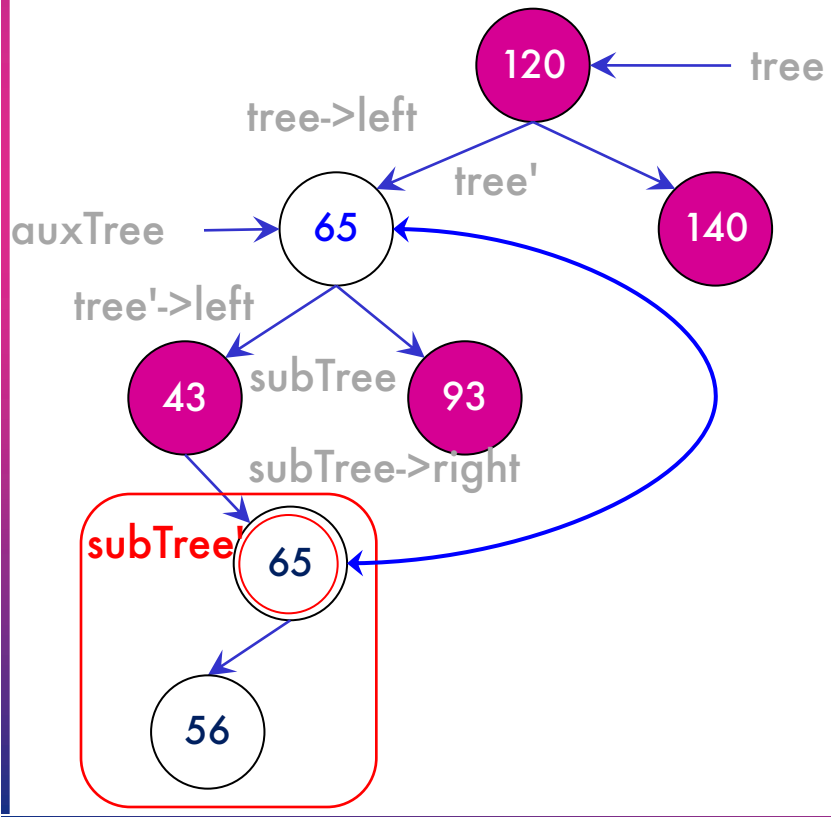
replace (subTree', auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);
  else
  {
     (*auxTree)->key = (*subTree)->key;
     *auxTree = *subTree;
     (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

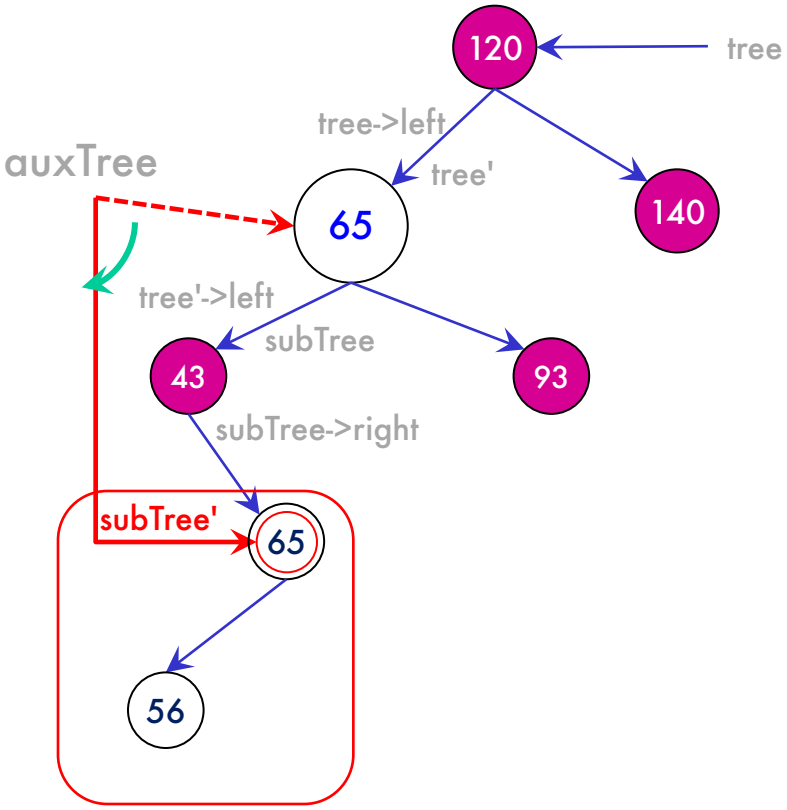
replace (subTree', auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);
  else
  {
    (*auxTree)->key = (*subTree)->key;
    *auxTree = *subTree;
    (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

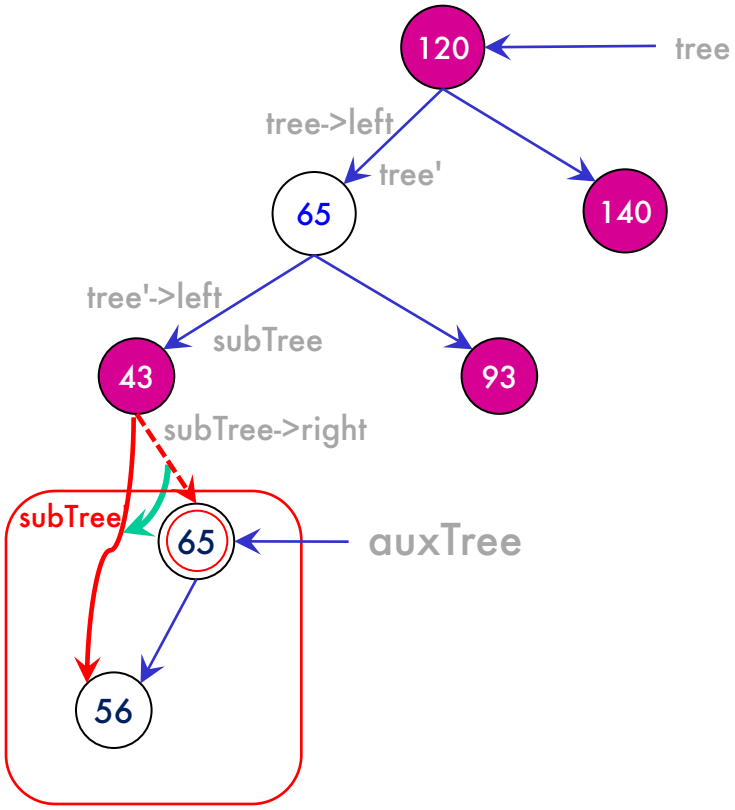
replace (subTree', auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);
  else
  {
     (*auxTree)->key = (*subTree)->key;
     *auxTree = *subTree;
     (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

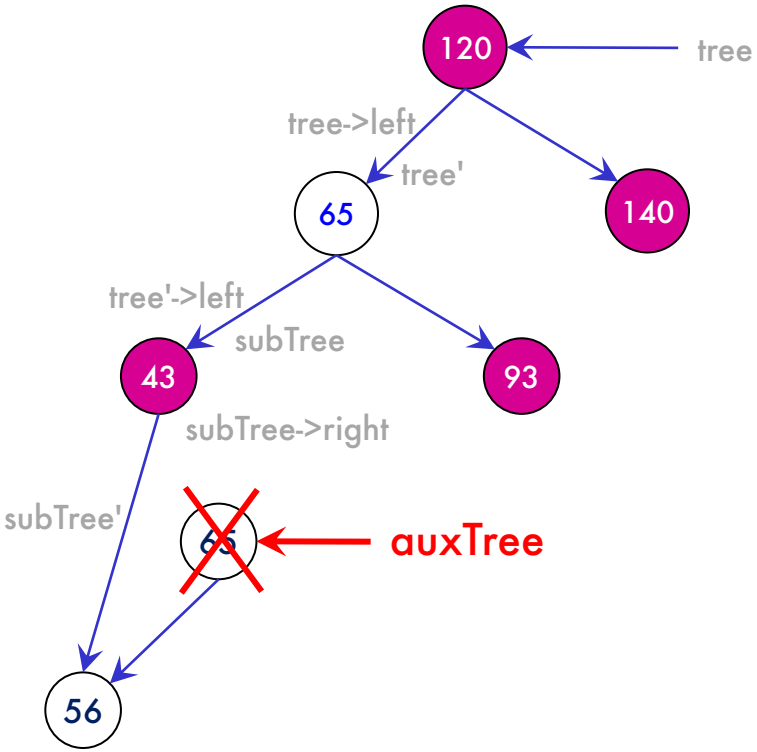
replace (subTree', auxTree)




```
void replace(tBST* subTree, tBST* auxTree)
{
  if(!isEmptyTree((*subTree)->right))
     replace(&(*subTree)->right, auxTree);
  else
  {
     (*auxTree)->key = (*subTree)->key;
     *auxTree = *subTree;
     (*subTree) = (*subTree)->left;
  }
}
```

ABB. Borrado con dos descendientes (87)

replace (subTree', auxTree)




```
else
  {
    aux = *tree;
    if (isEmptyTree((*tree)->right))
      *tree = (*tree)->left;
    else if (isEmptyTree((*tree)->left))
      *tree = (*tree)->right;
    else
      replace(&(*tree)->left, &aux);
    free(aux);
  }
```

ABB. Borrado con dos descendientes (87)

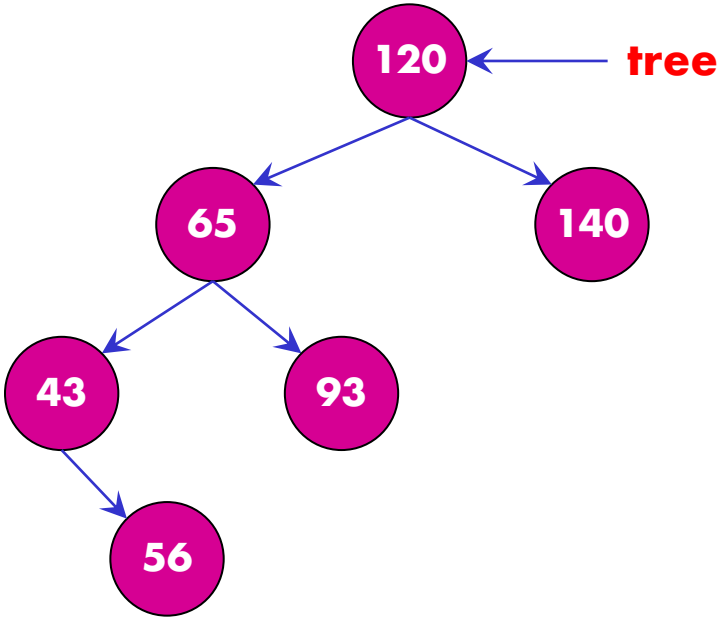


ABB: Ejemplo

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

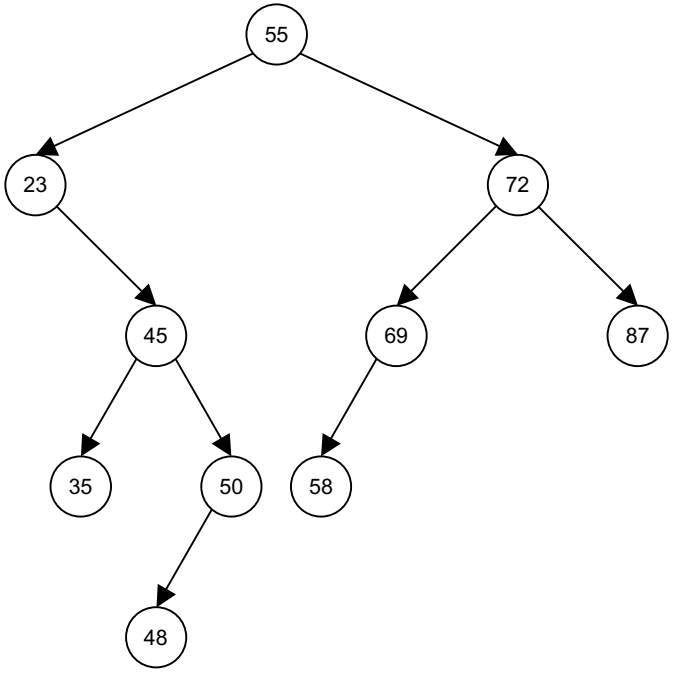
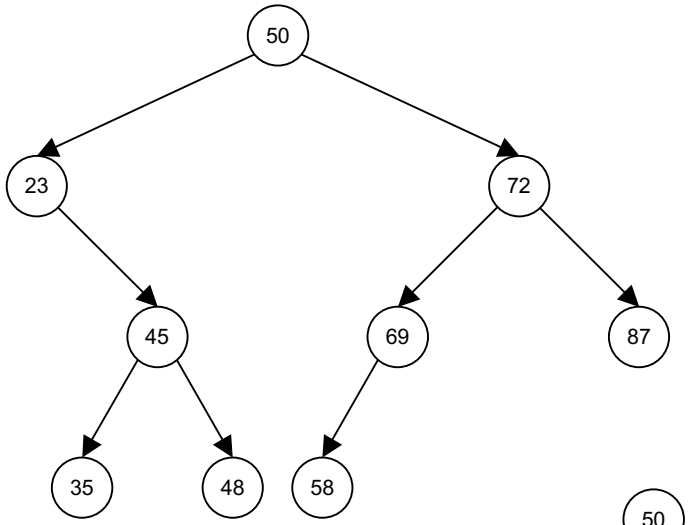


ABB: Ejemplo (II)

- Borramos: 55



- Borramos: 72

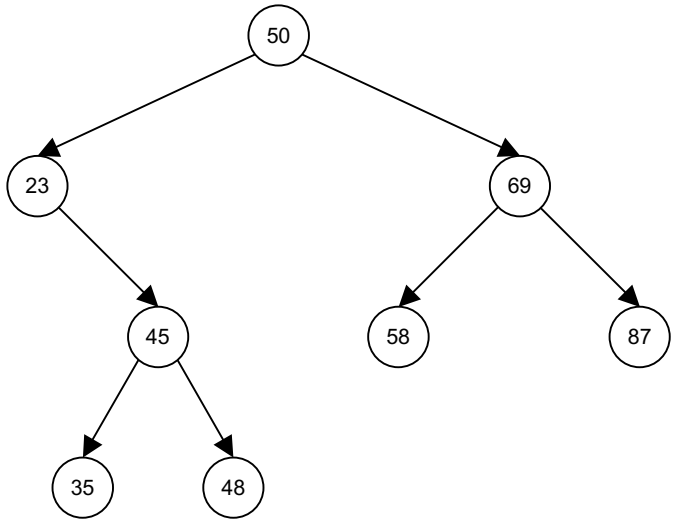
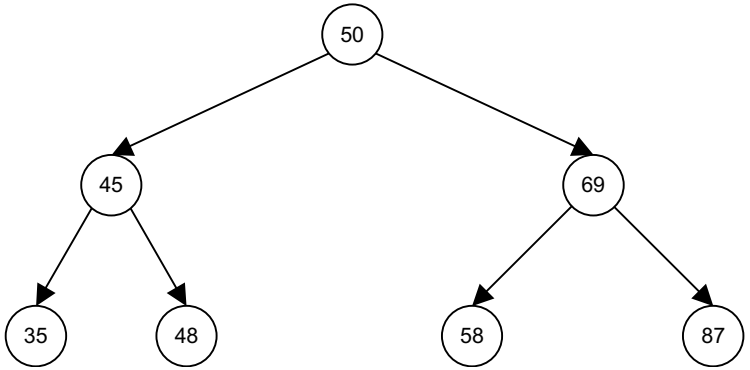


ABB: Ejemplo (III)

- Borramos: 23



- Borramos: 35