Estructura de Datos y de la Información

Árboles Binarios de Búsqueda

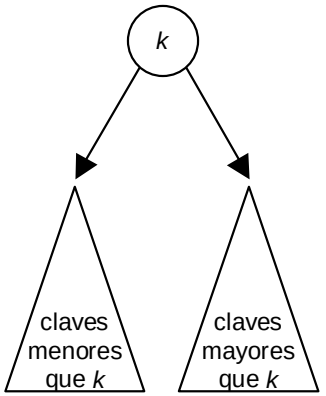
Eduardo Mosqueira Rey
Bertha Guijarro Berdiñas
Mariano Cabrero Canosa

LIDIA
Laboratorio de Investigación y Desarrollo
en Inteligencia Artificial

Departamento de Computación
Universidade da Coruña

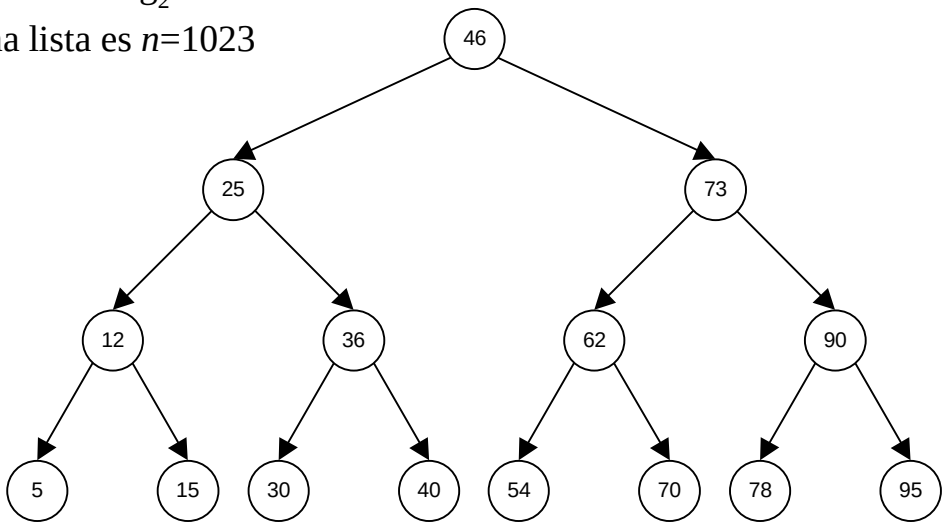
Árboles
Árboles Binarios de Búsqueda

- **Motivación**
  - **La búsqueda en una lista dinámica ordenada es poco eficiente:**
    - Por término medio habrá que recorrerse la mitad de la lista para verificar si un elemento está o no en ella.
    - Si la lista está en memoria principal no habrá problema, pero si la lista es una tabla de una base de datos almacenada en memoria secundaria el rendimiento se verá severamente afectado
  - **En una lista ordenada estática se pueden aplicar algoritmos de búsqueda mejores (p.e., dicotómica), pero el almacenamiento es limitado**
- **Solución: Árboles Binarios de Búsqueda o ABB**
  - **Un ABB es un árbol binario en el que sus TODOS sus nodos verifican:**
    - Tienen asociada una clave de ordenación
    - Su subárbol izquierdo (si existe) contiene valores menores que su clave y el subárbol derecho (si existe) contiene valores mayores.



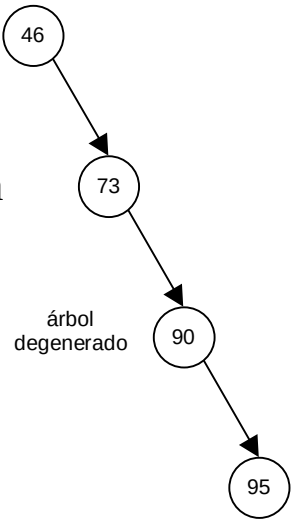
Árboles
Árboles Binarios de Búsqueda

- **Ventajas**
  - **El número de accesos al árbol es menor que en una lista enlazada**
  - **Por ejemplo, en un árbol lleno que tenga *n* nodos el camino más largo que hay que recorrer es $\log_2(n+1)$,**
    - Si n=15,
      - recorrido máximo en un ABB=$\log_2 16$=4
      - recorrido máximo en una lista es n=15
    - Si n=1023,
      - recorrido máximo en un ABB=$\log_2 1024$=10
      - recorrido máximo en una lista es n=1023



Árboles
Árboles Binarios de Búsqueda

- **Inconvenientes**
  - **Árboles equilibrados**
    - Las búsquedas son más eficientes cuando el árbol está equilibrado
    - Un árbol equilibrado es aquel en el que 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 izq y dch de la misma altura)
    - Si los nodos que se van insertando en el árbol aparecen en orden aleatorio el árbol tenderá a ser equilibrado
  - **Árboles "degenerados"**
    - El caso peor ocurre cuando el árbol está "degenerado", es decir, sigue siendo un árbol pero su estructura es equivalente a una lista enlazada (todos los nodos sólo tienen un hijo)
    - Si los nodos del árbol que se van insertando en el árbol aparecen con un orden determinado el árbol tenderá a ser degenerado.
  - **Los datos en la realidad suelen tener un cierto grado de orden por lo que para que los árboles BB sean eficientes es necesario hacer reequilibrados ⇒ árboles AVL**



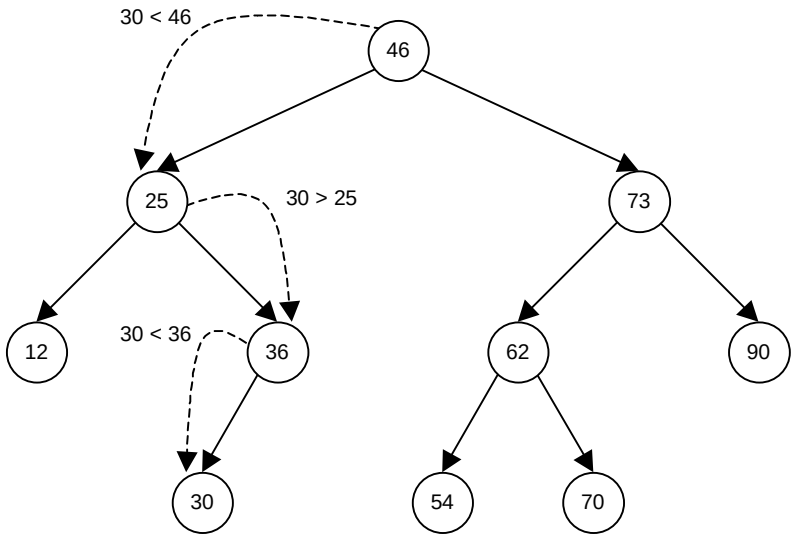
Árboles
Árboles Binarios de Búsqueda

- **Especificación del TAD Árbol Binario de Búsqueda**
  - **Tipos**
    - tArbolBin y tInfo
  - **Funciones (se resaltan las nuevas)**
    - Generadoras
      - ArbolVacio → tArbolBin
      - **InsertarABB (tArbolBin, tInfo) → tArbolBin, boolean**
    - Observadoras
      - HijoIzquierdo(tArbolBin) → tArbolBin
      - HijoDerecho(tArbolBin) → tArbolBin
      - Raiz (tArbolBin) → tInfo
      - EsArbolVacio (tArbolBin) → boolean
      - **BusquedaABB (tArbolBin, tInfo) → tArbolBin**
    - Destructoras
      - EliminarArbol (tArbolBin) → tArbolBin
      - **EliminarABB (tArbolBin, tInfo) → tArbolBin, boolean**
  - Funciones internas de la implementación
      - ConstruirArbol (tArbolBin, tInfo, tArbolBin) → tArbolBin, boolean

Árboles
Árboles Binarios de Búsqueda

- **Búsqueda (pseudocódigo)**
  - **Se compara la clave a buscar con la raíz del arbol**
    - Si el árbol es NULO 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
- **Búsqueda (código y ejemplo)**

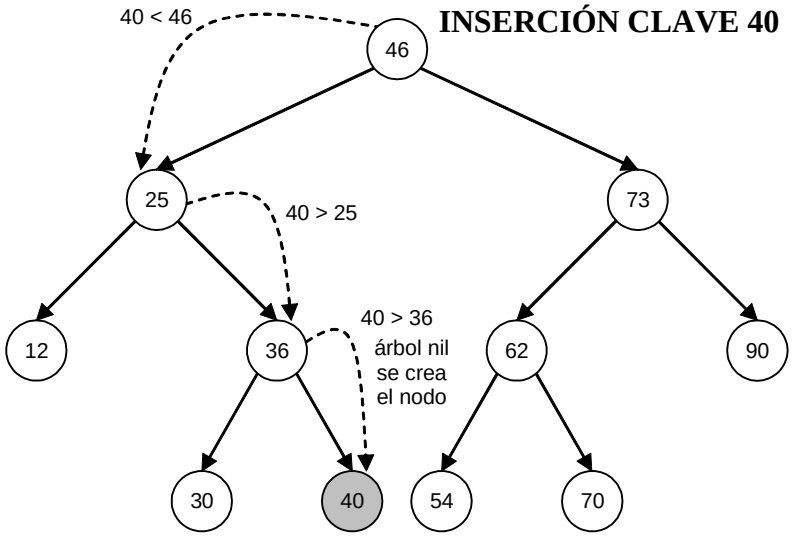
```
function BuscaABB(A:tArbolBin; valor:tInfo): tArbolBin;
begin
  if A=nil
    then BuscaABB:=nil
    else
      if valor=A^.Info
        then BuscaABB:=A
        else if valor<A^.info
               then BuscaABB:=BuscaABB(A^.HI, valor)
               else BuscaABB:=BuscaABB(A^.HD, valor)
end;
```



Árboles
Árboles Binarios de Búsqueda

- **Inserción (pseudocódigo)**
  - **Se compara la clave a insertar con la raíz del arbol**
    - Si el árbol es NULO insertamos una hoja con la clave en esa posición
    - 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
    - Si clave = valor (claves repetidas) no se hace nada. Este funcionamiento podría ser distinto: reemplazar el antiguo con el nuevo, lanzar un error, permitir duplicados, etc.
- **Inserción (código y ejemplo)**

```
function InsertarABB(var A:tArbolBin; valor:tInfo):boolean;
begin
  if (A=nil) then
    InsertarABB:=ConstruirArbol(nil, nil, valor, A)
  else
    if valor<A^.info
      then InsertarABB:=InsertarABB(A^.HI, valor)
      else if valor>A^.info
             then InsertarABB:=InsertarABB(A^.HD, valor)
             else InsertarABB:=true
end;
```



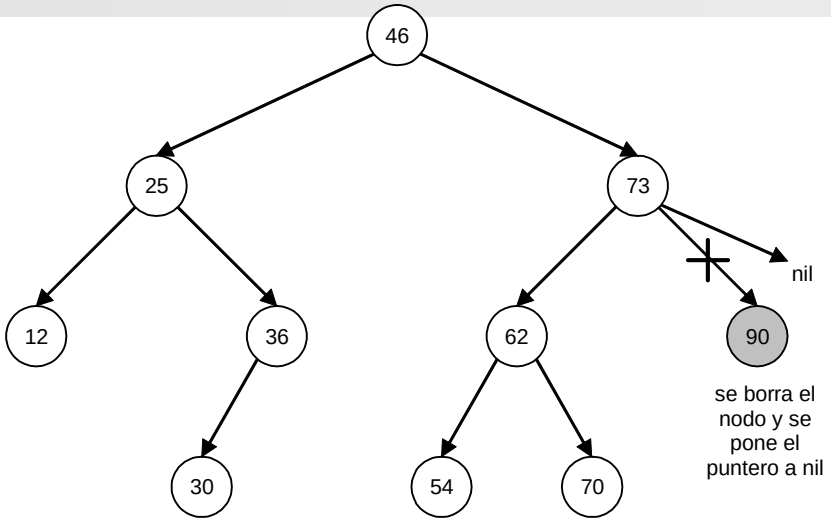
Árboles
Árboles Binarios de Búsqueda

- **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 nil 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**
  - **Si es un nodo con dos hijos**
    - Se intercambia el nodo con el mayor de su subárbol izquierdo (nodo anterior en el orden) o con el menor de su subarbol derecho (nodo posterior en el orden) y se borra el nodo.
    - El mayor de su subárbol izquierdo 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 mayor). Por lo tanto es un nodo sencillo de borrar
    - El menor de su subárbol derecho 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 menor). Por lo tanto es un nodo sencillo de borrar
    - Al substituir un nodo por su anterior o su posterior la propiedad de árbol binario de búsqueda se sigue cumpliendo.

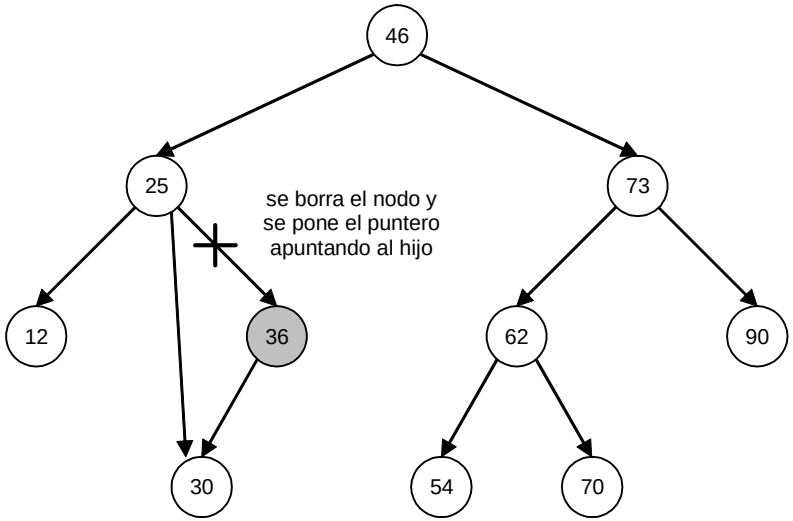
Árboles
Árboles Binarios de Búsqueda

- **Borrado (ejemplos)**

  * **Nodo hoja (90)**

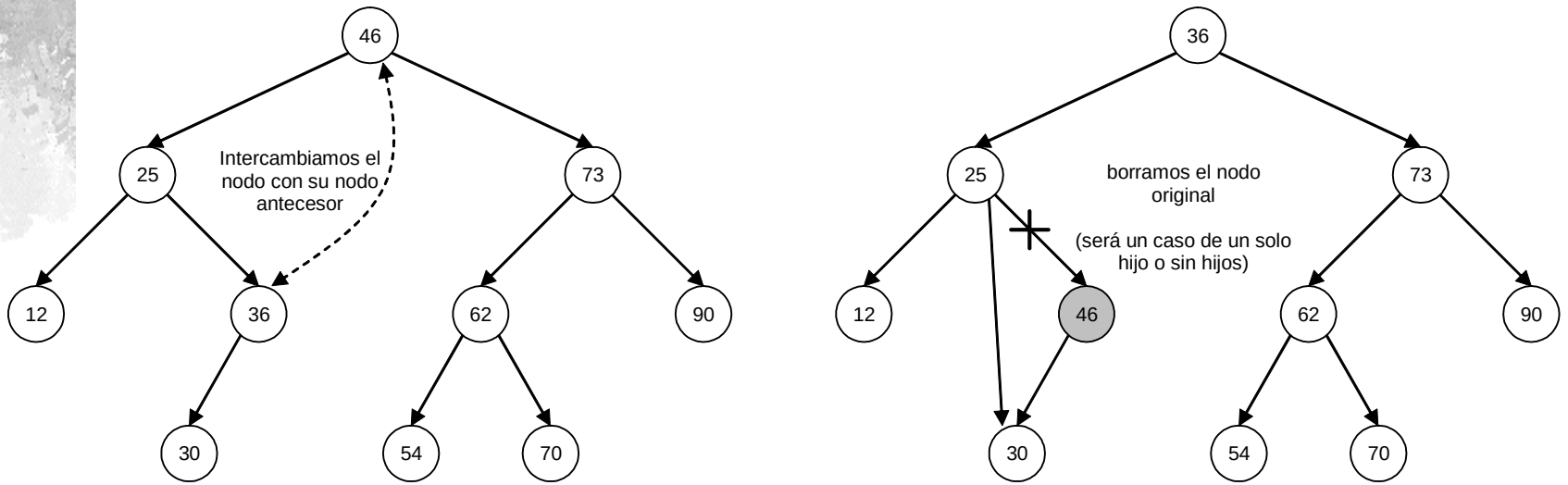


  * **Nodo con un solo hijo (36)**



Árboles
Árboles Binarios de Búsqueda

- **Borrado (ejemplo)**
  - **Nodo con dos hijos (46)**



Árboles

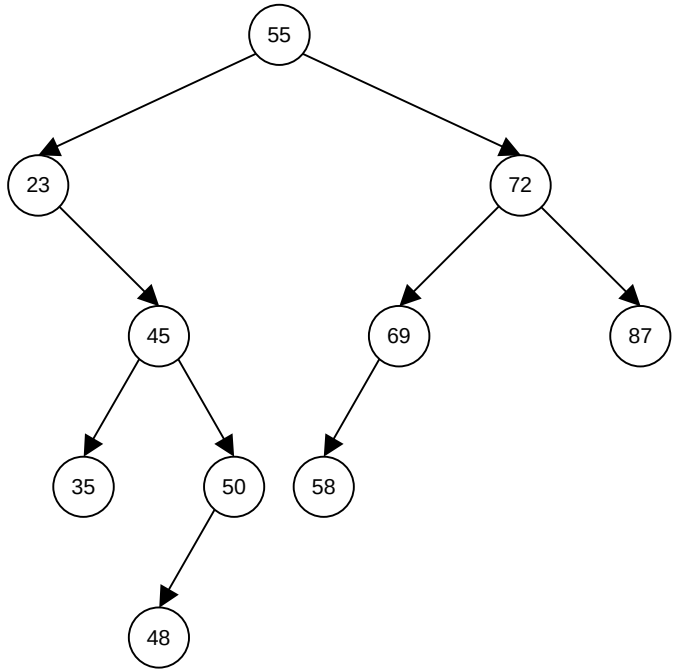
Árboles Binarios de Búsqueda

- **Borrado (código)**

```
function EliminarABB(var A:tArbolBin; valor:tInfo):boolean;
var
  aux:tArbolBin;

{-----------------------------------------------------}

// reemplaza el nodo por su anterior
procedure reemplazar(var aux, N:tArbolBin);
begin
  if N^.HD <> nil
    then reemplazar(N^.HD)     // Bajamos por la rama derecha
    else begin
           aux^.info:=N^.info;// reemplazamos los campos de información
           aux:=N;            // Marcamos el nodo sobre el que se hará un dispose
           N:=N^.HI           // y reenlazamos la estructura "saltando" al nodo que se va a eliminar
         end
end;

{-----------------------------------------------------}

begin
  if A=nil
    then EliminarABB:=false                 // Si A es VACIO, el valor no existe y se devuelve false
    else if valor<A^.info                                 // Si el valor es menor mandamos borrar en subárbol izq
           then EliminarABB:=EliminarABB(A^.HI, valor)
           else if valor>A^.info                          // Si el valor es mayor mandamos borrar en subárbol dch
                  then EliminarABB:=EliminarABB(A^.HD, valor)
                  else begin                              // Si el valor es igual borramos el nodo
                         EliminarABB:=true;               // Se devuelve true porque el valor existe
                         aux:=A;
                         if A^.HD = nil                   // Si no tiene hijo dch lo sustituimos por el izq
                           then A:=A^.HI                  // (incluye el caso de los dos hijos vacios)
                           else if A^.HI = nil            // Si no tiene hijo izq lo sustituimos por el dch
                                  then A:=A^.HD                 // Si tiene dos hijos llamamos a reemplazar
                                  else reemplazar (aux, A^.HI); // pasando el nodo a eliminar y su subarbol izq
                         dispose(aux);
                       end
end;
```

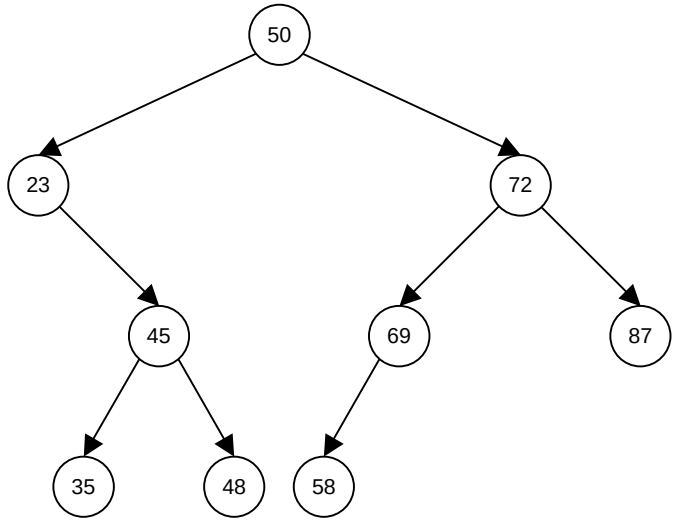
Árboles
Árboles Binarios de Búsqueda

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

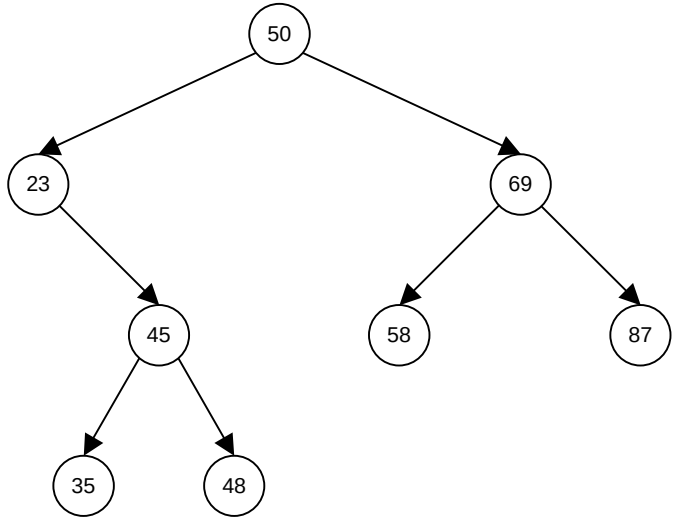


Árboles
Árboles Binarios de Búsqueda

- **Ejemplo:**
  - **Borramos: 55**

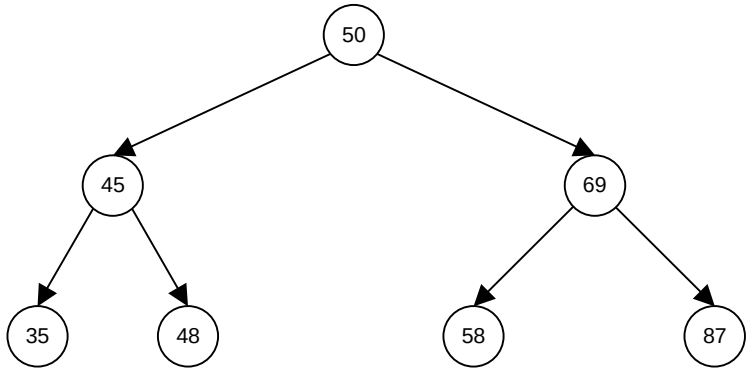


  - **Borramos: 72**



Árboles
Árboles Binarios de Búsqueda

- **Ejemplo:**
  - **Borramos: 23**



  - **Borramos: 35**