



Tema 8. Árboles

FUNDAMENTOS DE PROGRAMACIÓN II

Profesora: Raquel Martínez España

Escuela Politécnica

Contenidos

- Introducción
- Estructura de un árbol
- Tipos de recorrido de un árbol
- Operaciones básicas sobre un árbol

Contenidos

- **Introducción**
- Estructura de un árbol
- Tipos de recorrido de un árbol
- Operaciones básicas sobre un árbol

Tipos de estructuras de datos

- En el tema 5 vimos que las estructuras de datos pueden clasificarse en dos tipos:

- **Estructuras de datos Lineales**

- Las estructuras de datos lineales se caracterizan porque **sus elementos están en secuencia**, relacionados en forma lineal.
- Ejemplos: listas, pilas, colas



Ya las
hemos
estudiado

- **Estructuras de datos no lineales**

- **No existe una relación de secuencia** de almacenamiento de sus elementos.
- Cada elemento puede estar enlazado a cualquier otro componente.
- Ejemplos: **árboles** y **grafos**.



Estructuras no lineales

- Los nodos no tienen una relación uno a uno.
- Cada nodo puede tener de 0 a N sucesores y antecesores.
- Vamos a estudiar dos tipos:
 - Árboles  **TEMA 8**
 - Grafos  **TEMA 9**

Contenidos

- Introducción
- **Estructura de un árbol**
- Tipos de recorrido de un árbol
- Operaciones básicas sobre un árbol

Árboles

- Colección de elementos
- Elementos de un árbol: **nodos**
- **Estructura jerárquica:** cada nodo tiene
 - Un único predecesor
 - Dos o más sucesores
- Nodo especial: el **nodo raíz** que no tiene predecesor.
- Matemáticamente:
 - Grafo no orientado, conexo y acíclico
 - Existe un vértice destacado llamado “Raíz”

Tipos de árboles

- **Árbol A n -ario**
 - O bien es el conjunto vacío (árbol vacío)
 - O bien es no vacío:
 - Elemento Raíz
 - A_1, A_2, \dots, A_m subárboles
- **Definiciones:**
 - Árbol **ordenado**: el orden de los subárboles importa
 - La raíz de A es **padre** de la raíz de A_1, \dots, A_m
 - Las raíces de A_1, \dots, A_m son **hijos** de la raíz de A .

Tipos de árboles

■ Árbol binario **A**

- O bien es el conjunto vacío (árbol vacío)
- O bien es no vacío:
 - Elemento Raíz
 - A_1 , A_2 subárboles izquierdo y derecho

■ Definiciones:

- No es lo mismo un árbol 2-ario que un árbol binario
 - En el árbol binario se distingue entre subárboles izquierdo y derecho.
- Los términos padre e hijo descritos para árboles n-arios son también utilizados para árboles binarios.

Terminología

- **Camino** del nodo n_1 a n_k :
 - Secuencia de nodos n_1, \dots, n_k
 - Tal que n_i es padre de n_{i+1}
- **Longitud** de un camino:
 - Número de nodos del camino -1
 - Existe un camino de longitud 0 de todo nodo a sí mismo
- **Ascendente/Descendiente**:
 - Un a es ascendiente de b (y b descendiente de a), si existe un camino de a a b .
 - Todo nodo es ascendiente (y descendiente) descendiente de sí mismo.
 - Los ascendientes (y descendientes) de un nodo, excluido el propio nodo, se denominan ascendientes (y descendientes) **propios**.

Terminología

■ Hoja:

- Nodo sin descendientes propios

■ Altura:

- Longitud del camino más largo de ese nodo a una hoja.
- La **altura de un árbol** es la altura de la raíz.

■ Profundidad:

- La profundidad de un nodo es la longitud del camino único desde ese nodo a la raíz.

Árboles binarios

- Un árbol es siempre **acíclico**.
 - Un nodo no puede ser sucesor-t/predecesor-t de sí mismo.
 - No se produce **cierre transitivo**.
- Nosotros trabajaremos con árboles binarios:

```
arbol_binario ::= arbol_nulo | nodo
nodo ::= dato + hijo_derecho + hijo_izquierdo
hijo_derecho  ::= arbol_binario
hijo_izquierdo ::= arbol_binario
```

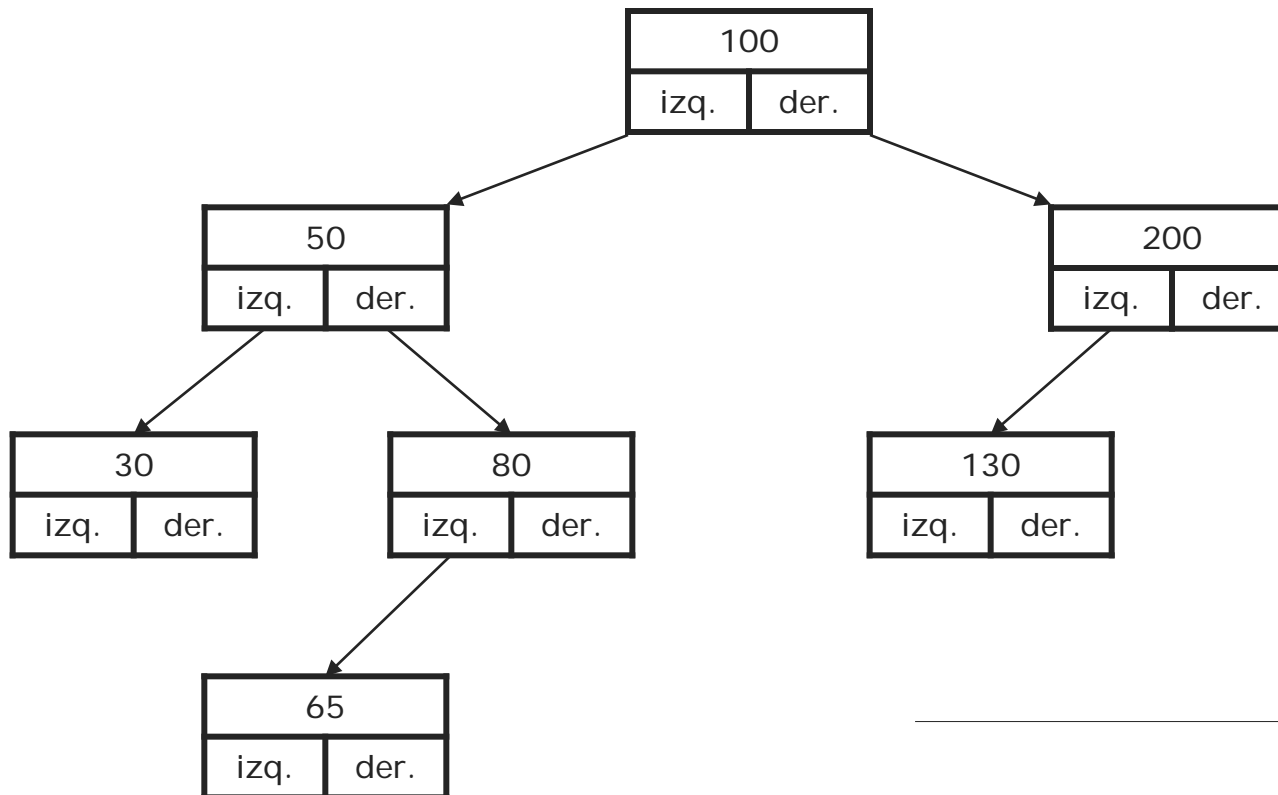
Árboles binarios ordenados

- Nada restringe los datos almacenados en un árbol.
- En la práctica, si existe una relación de orden ya que el dato de un nodo:
 - Suele ser mayor que el de todos los nodos que “cuelgan” del lado izquierdo.
 - Suele ser menor que el de todos los nodos que “cuelgan” del lado derecho.

Árboles binarios ordenados

■ Estructura

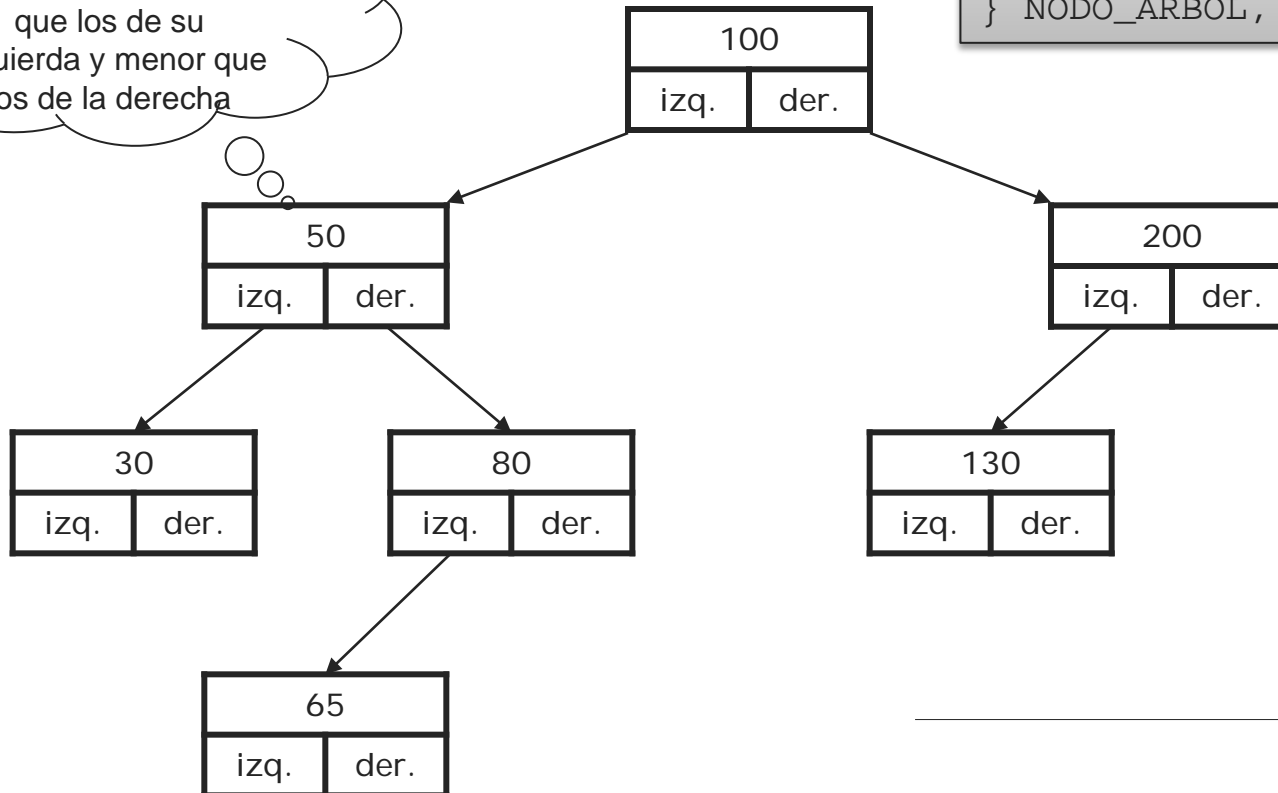
```
typedef struct nodo_arbol{  
    int valor;  
    struct nodo_arbol *izq;  
    struct nodo_arbol *der;  
} NODO_ARBOL, *P_NODO_ARBOL
```



Árboles binarios ordenados

■ Estructura

Este nodo es mayor que los de su izquierda y menor que los de la derecha

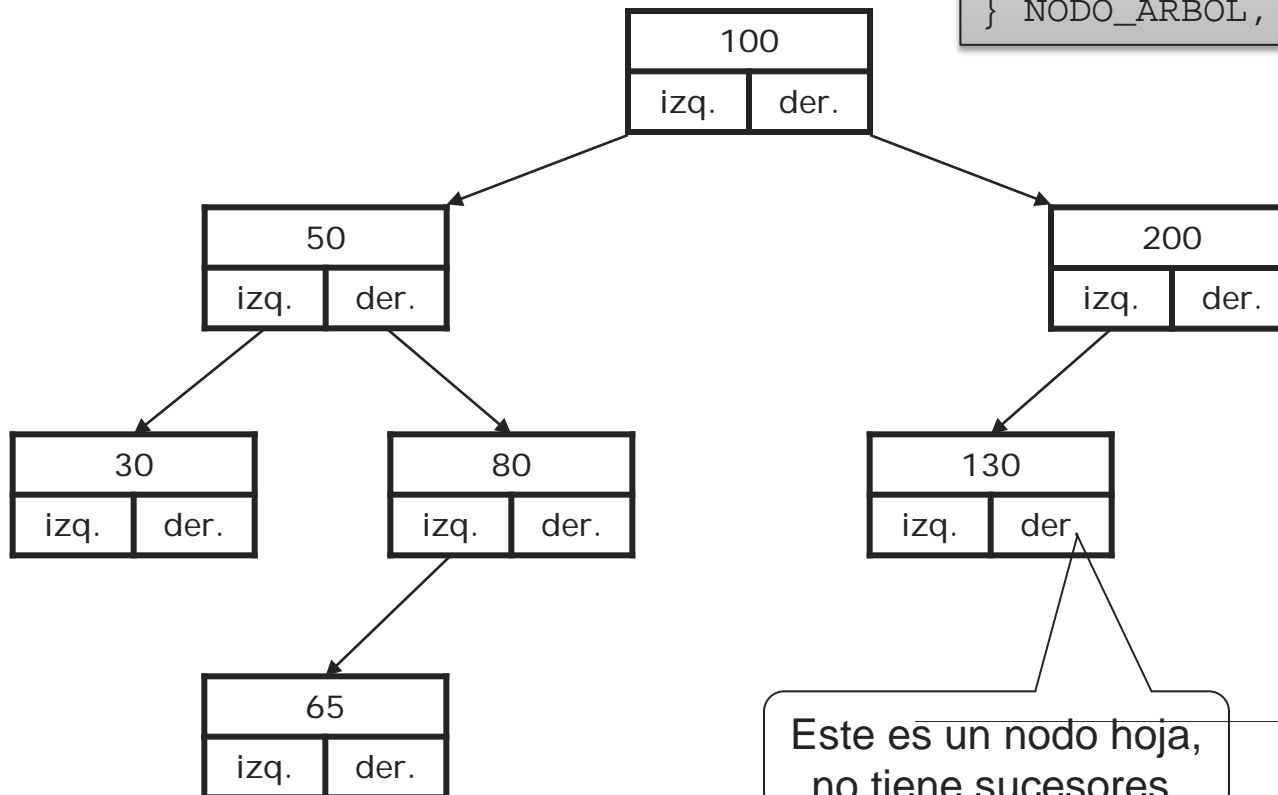


```
typedef struct nodo_arbol{
    int valor;
    struct nodo_arbol *izq;
    struct nodo_arbol *der;
} NODO_ARBOL, *P_NODO_ARBOL
```

Árboles binarios ordenados

■ Estructura

```
typedef struct nodo_arbol{  
    int valor;  
    struct nodo_arbol *izq;  
    struct nodo_arbol *der;  
} NODO_ARBOL, *P_NODO_ARBOL
```



Contenidos

- Introducción
- Estructura de un árbol
- **Tipos de recorrido de un árbol**
- Operaciones básicas sobre un árbol

Recorridos en árboles binarios

- Tipos de recorrido en un árbol binario
 - Recorrido en orden previo
 - Recorrido en orden simétrico
 - Recorrido en orden posterior

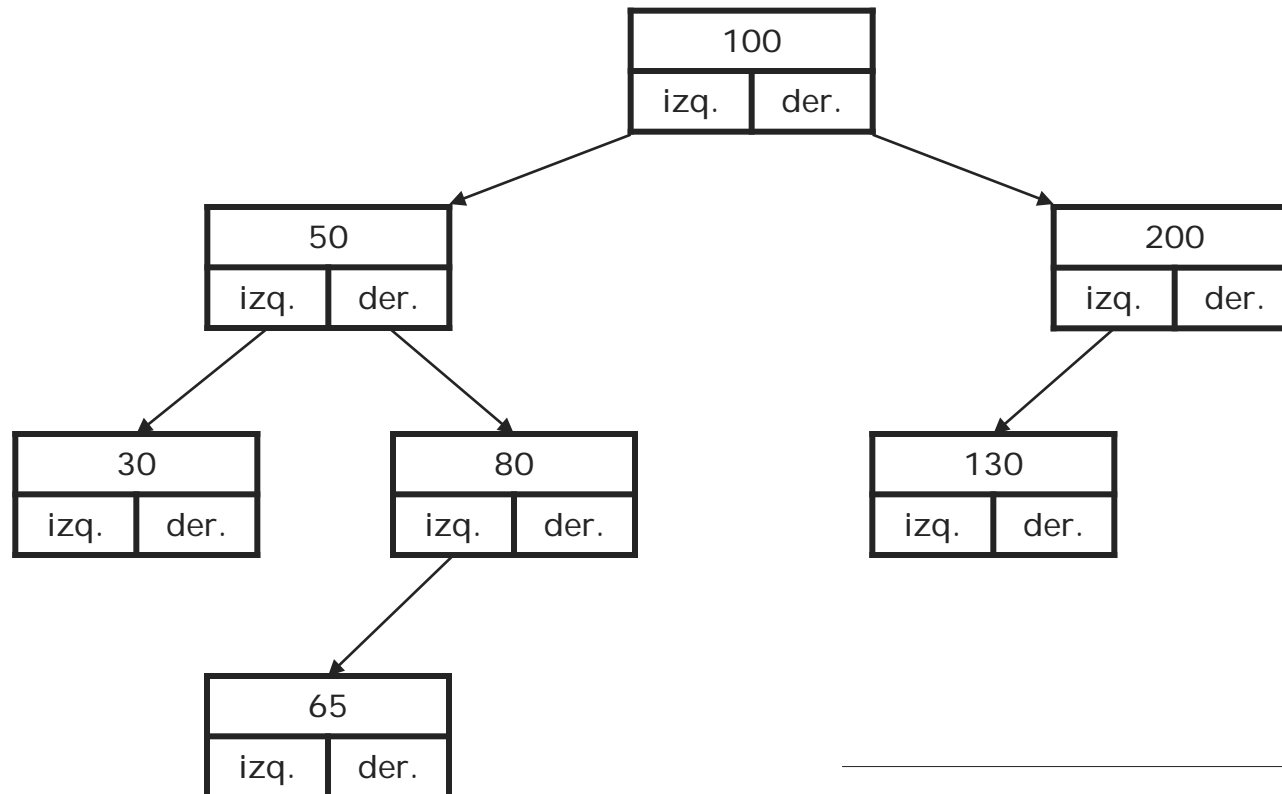
Recorridos en árboles binarios

■ Recorrido en orden previo

- Compuesto por: nodo raíz, seguido del recorrido en orden previo del subárbol izquierdo y seguido recorrido en orden previo del subárbol derecho

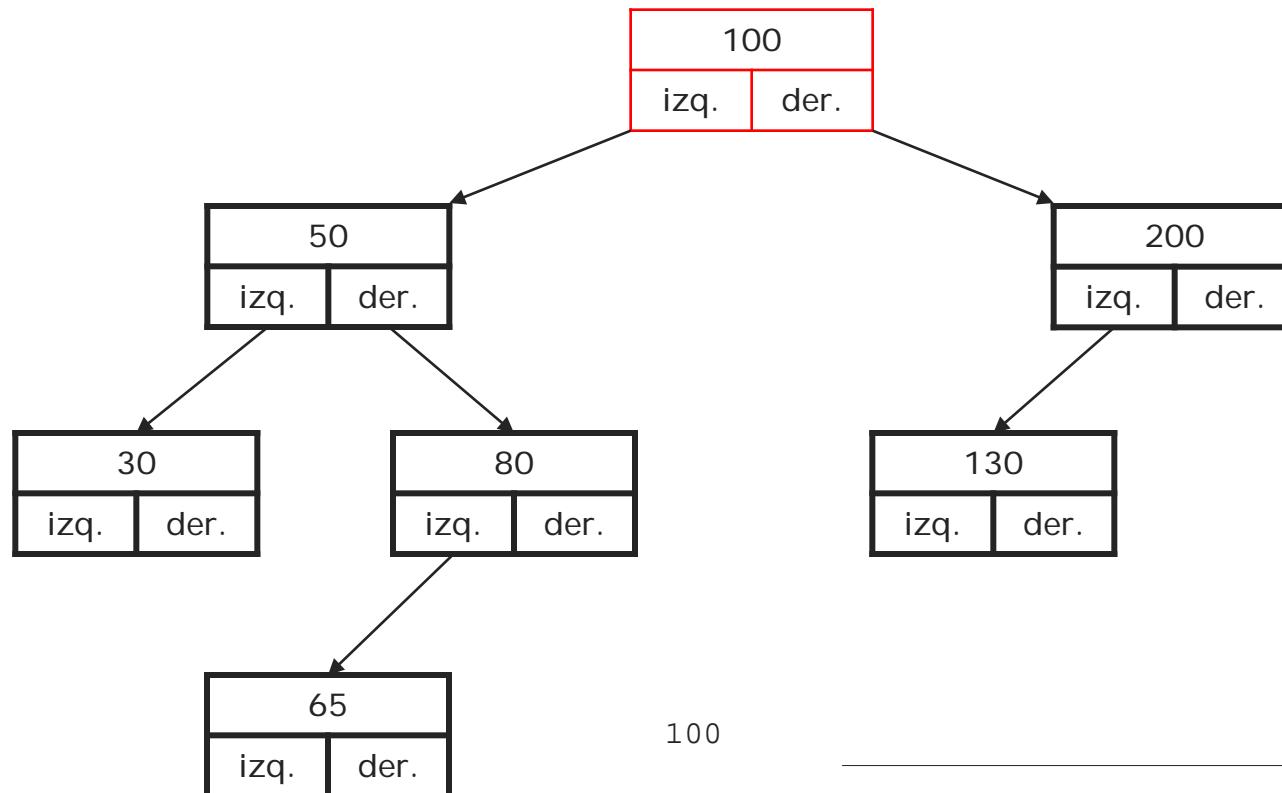
Recorridos en árboles binarios

■ Orden previo



Recorridos en árboles binarios

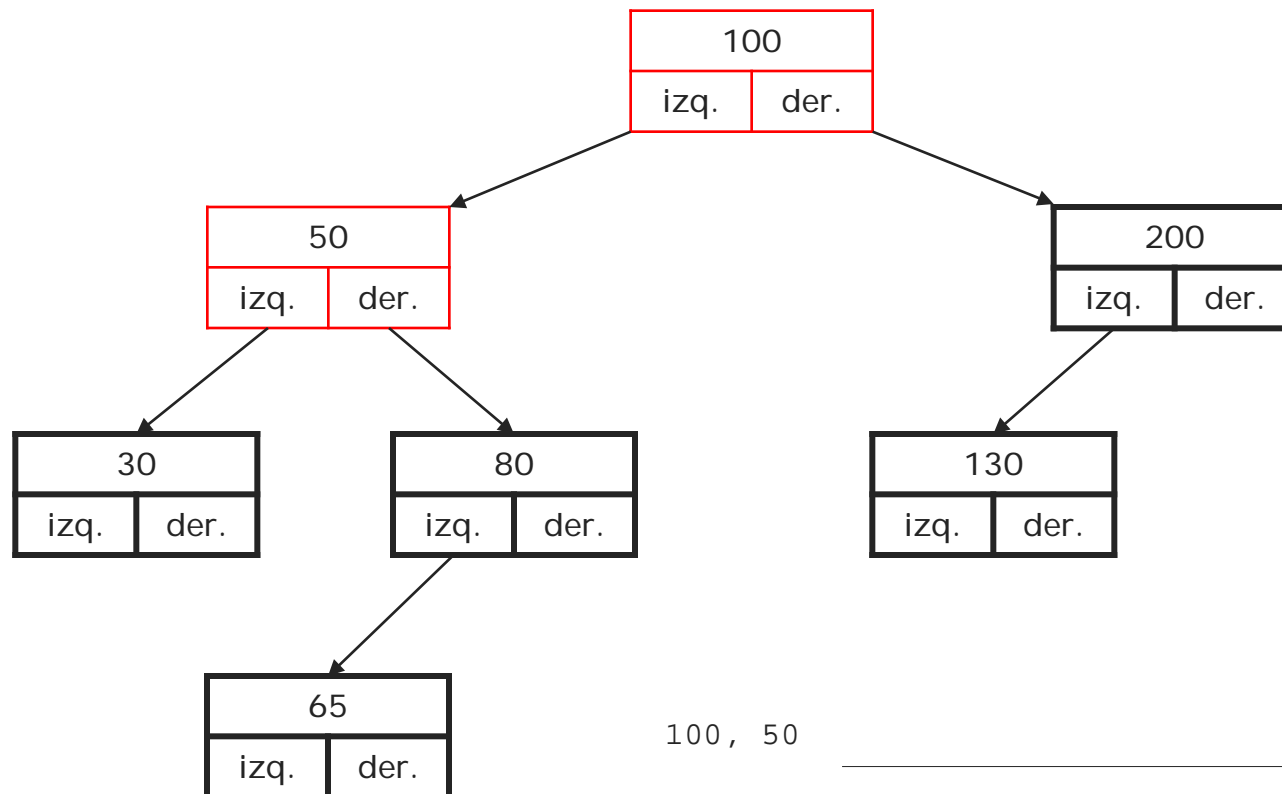
■ Orden previo



100

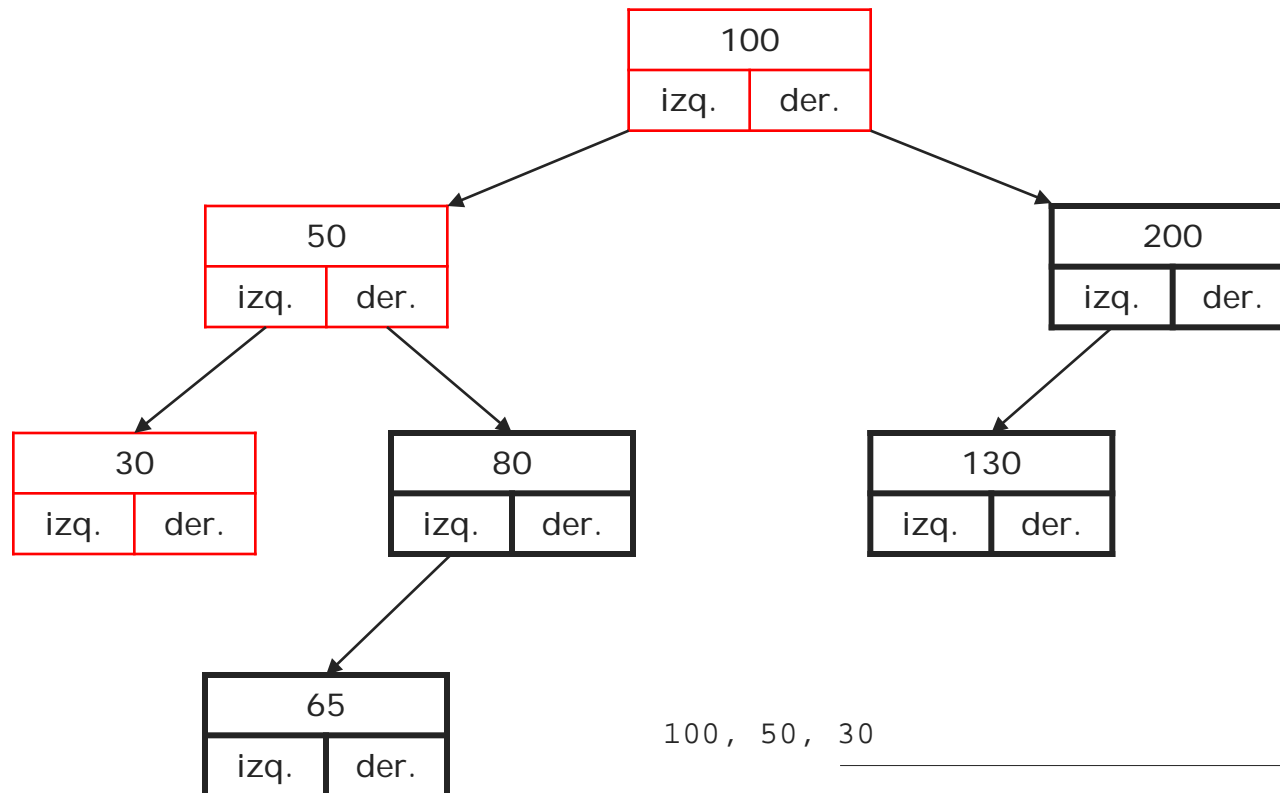
Recorridos en árboles binarios

■ Orden previo



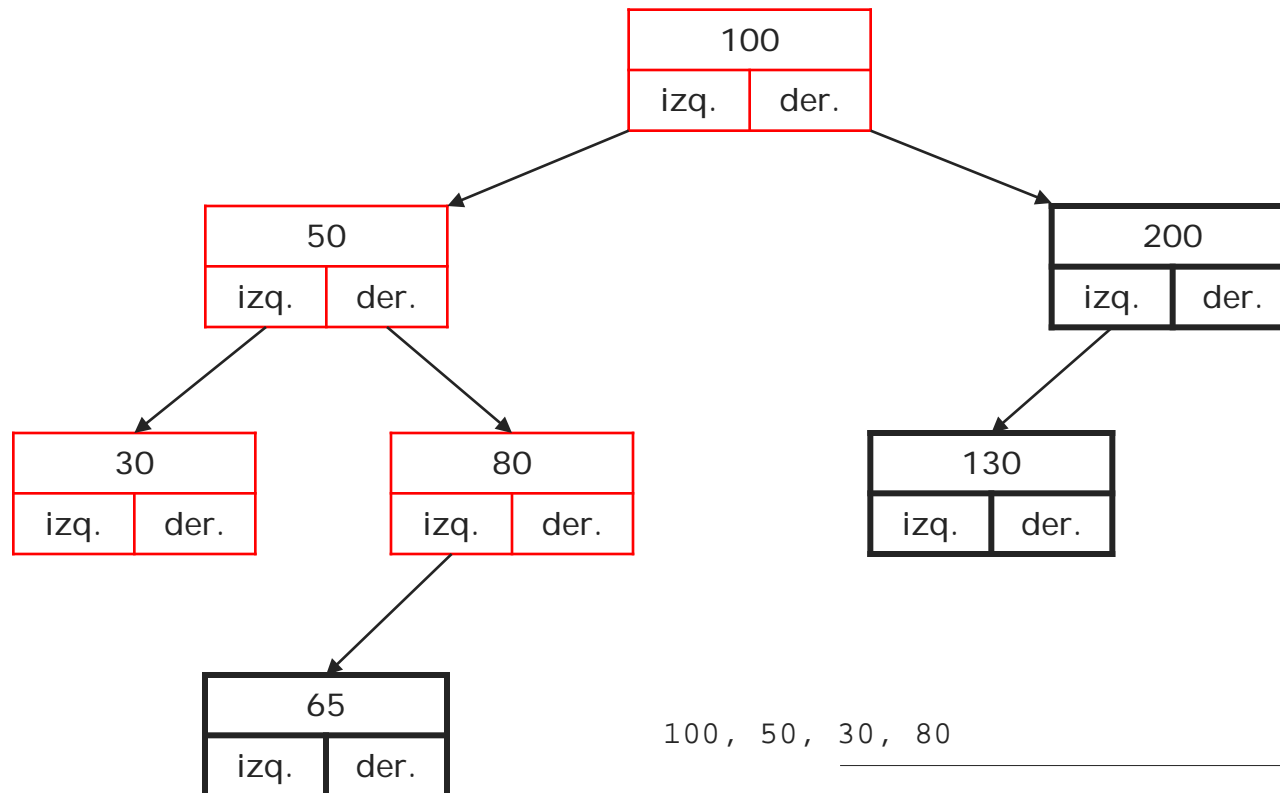
Recorridos en árboles binarios

■ Orden previo



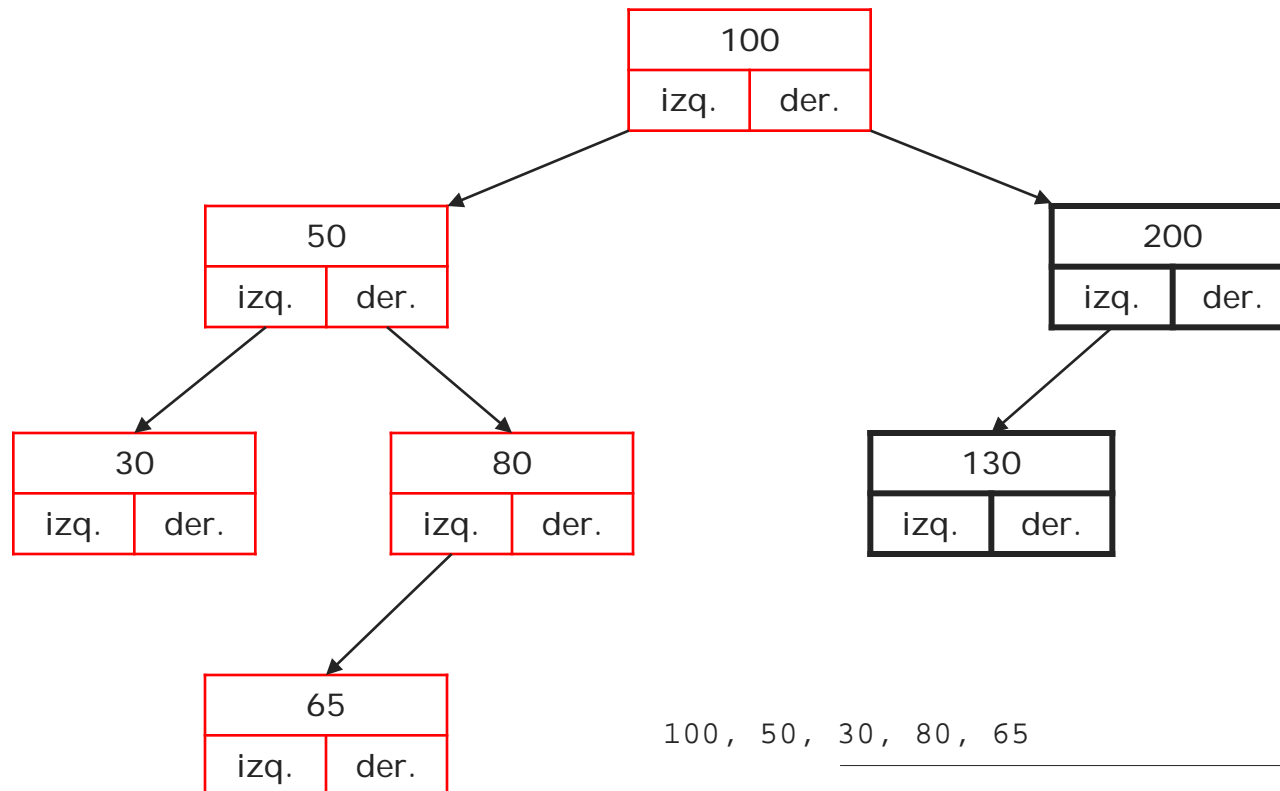
Recorridos en árboles binarios

■ Orden previo



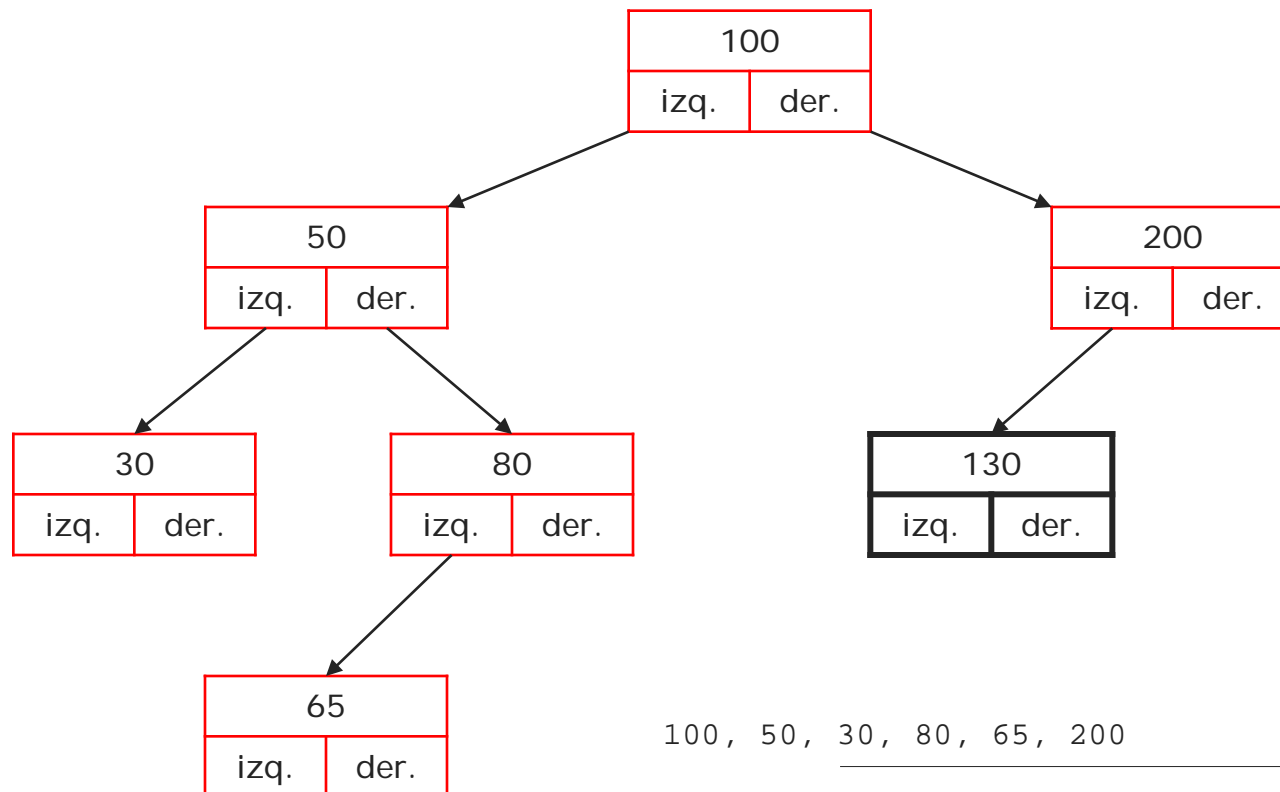
Recorridos en árboles binarios

■ Orden previo



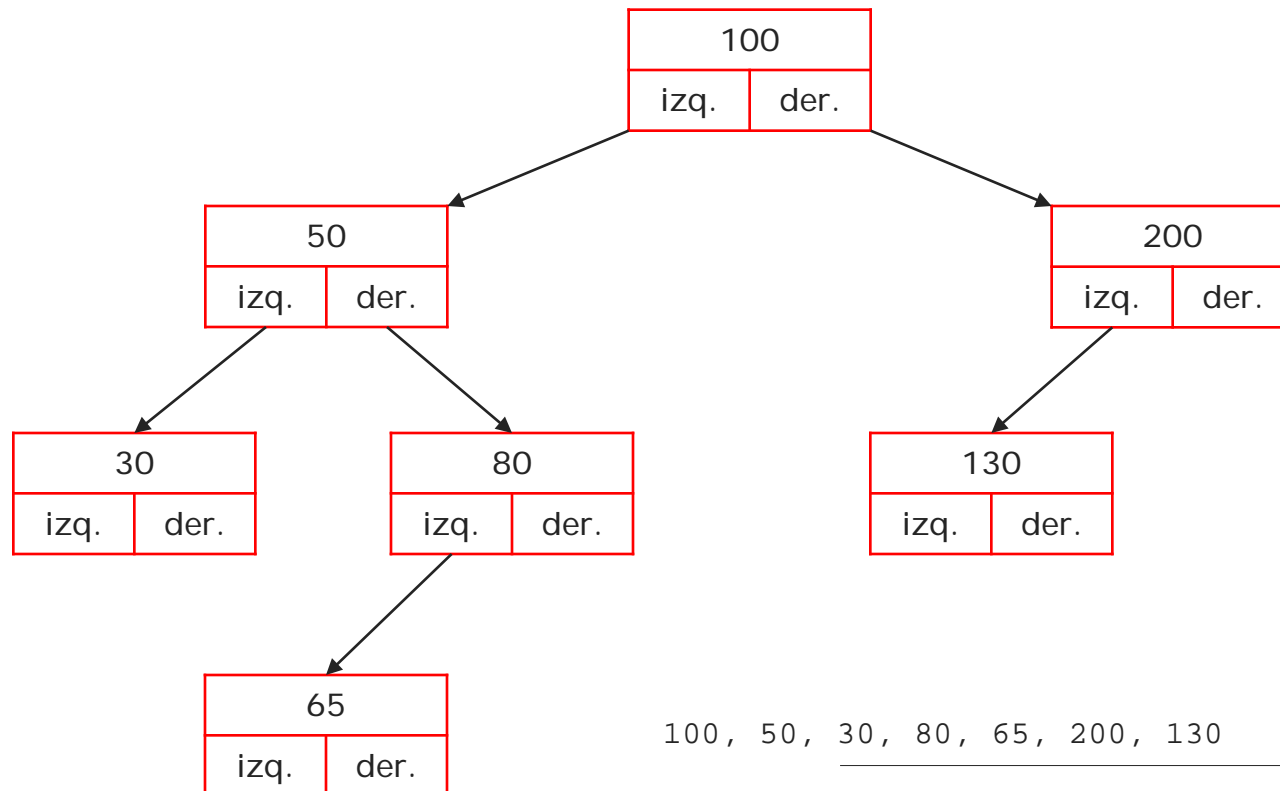
Recorridos en árboles binarios

■ Orden previo



Recorridos en árboles binarios

■ Orden previo



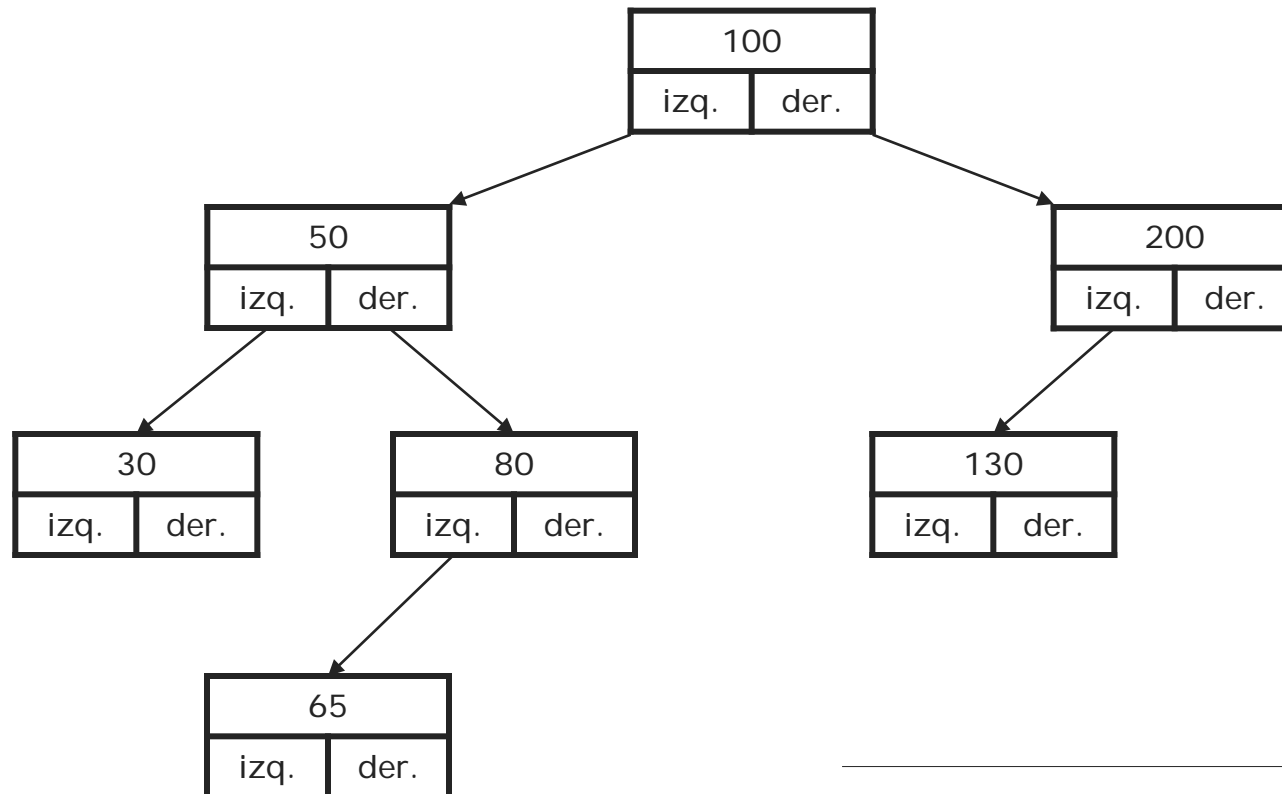
Recorridos en árboles binarios

■ Recorrido en orden simétrico

- Compuesto por: recorrido en orden simétrico del subárbol izquierdo, seguido del nodo raíz y seguido del recorrido en orden simétrico del subárbol derecho

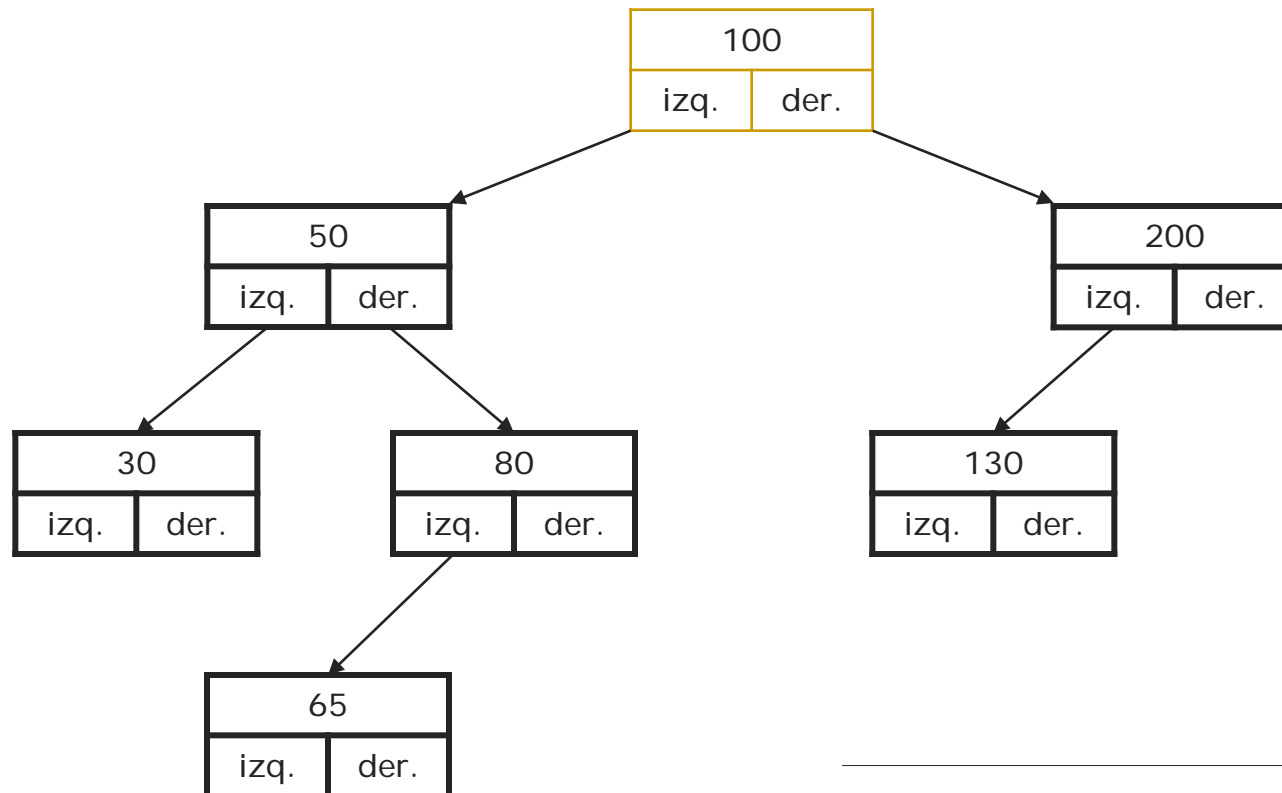
Recorridos en árboles binarios

■ Orden simétrico



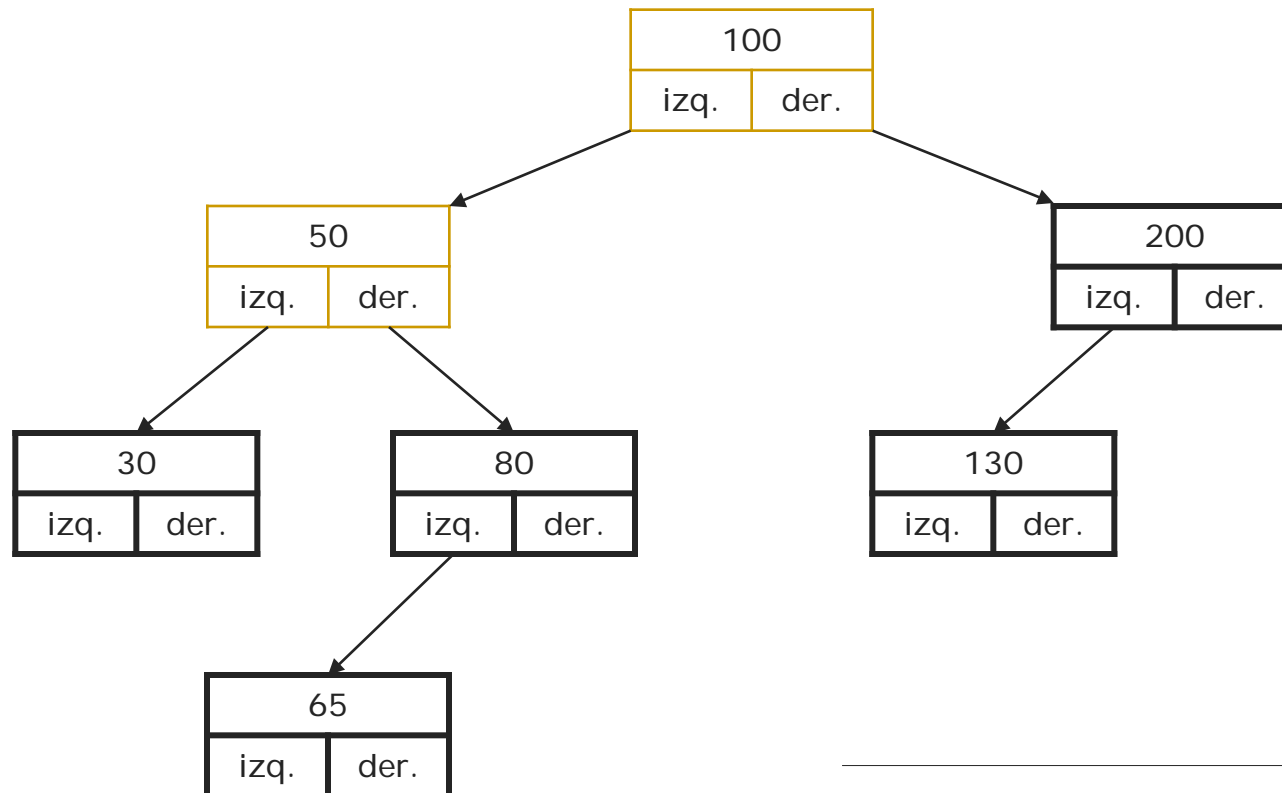
Recorridos en árboles binarios

■ Orden simétrico



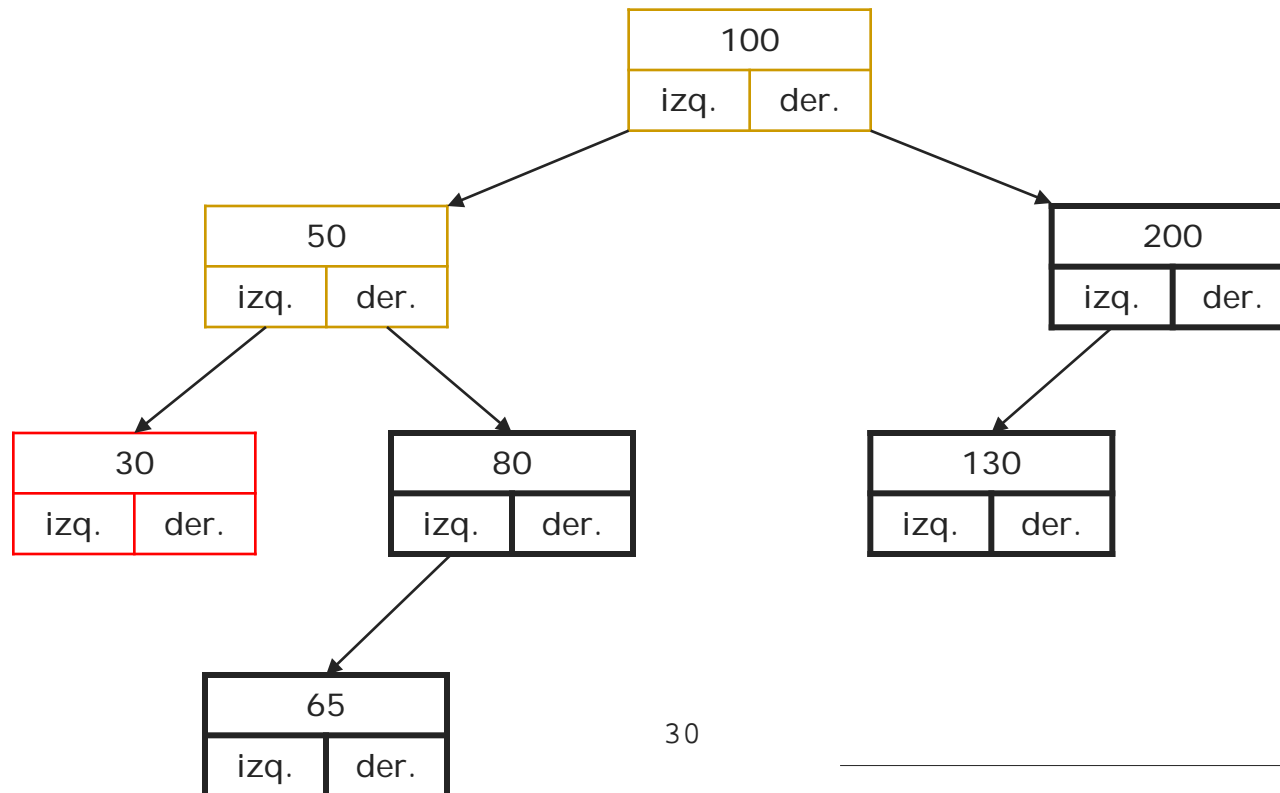
Recorridos en árboles binarios

■ Orden simétrico



Recorridos en árboles binarios

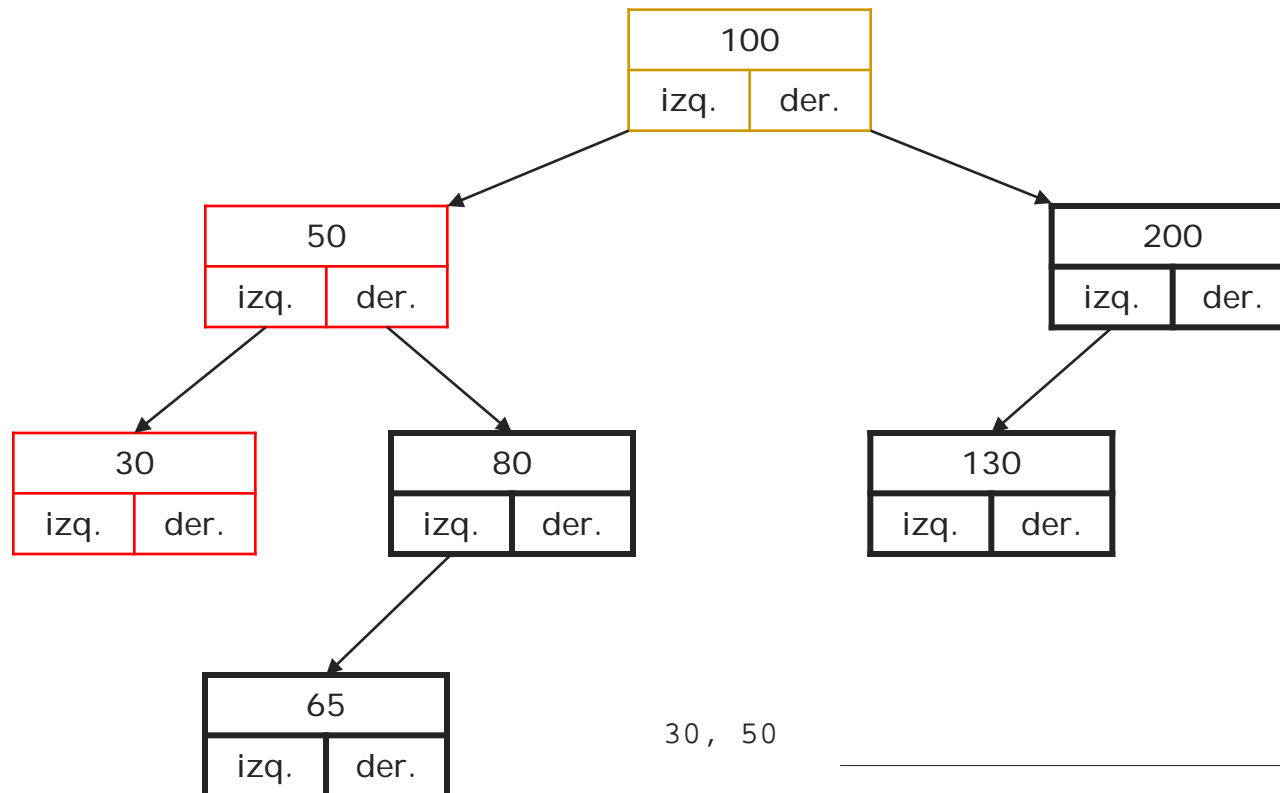
■ Orden simétrico



30

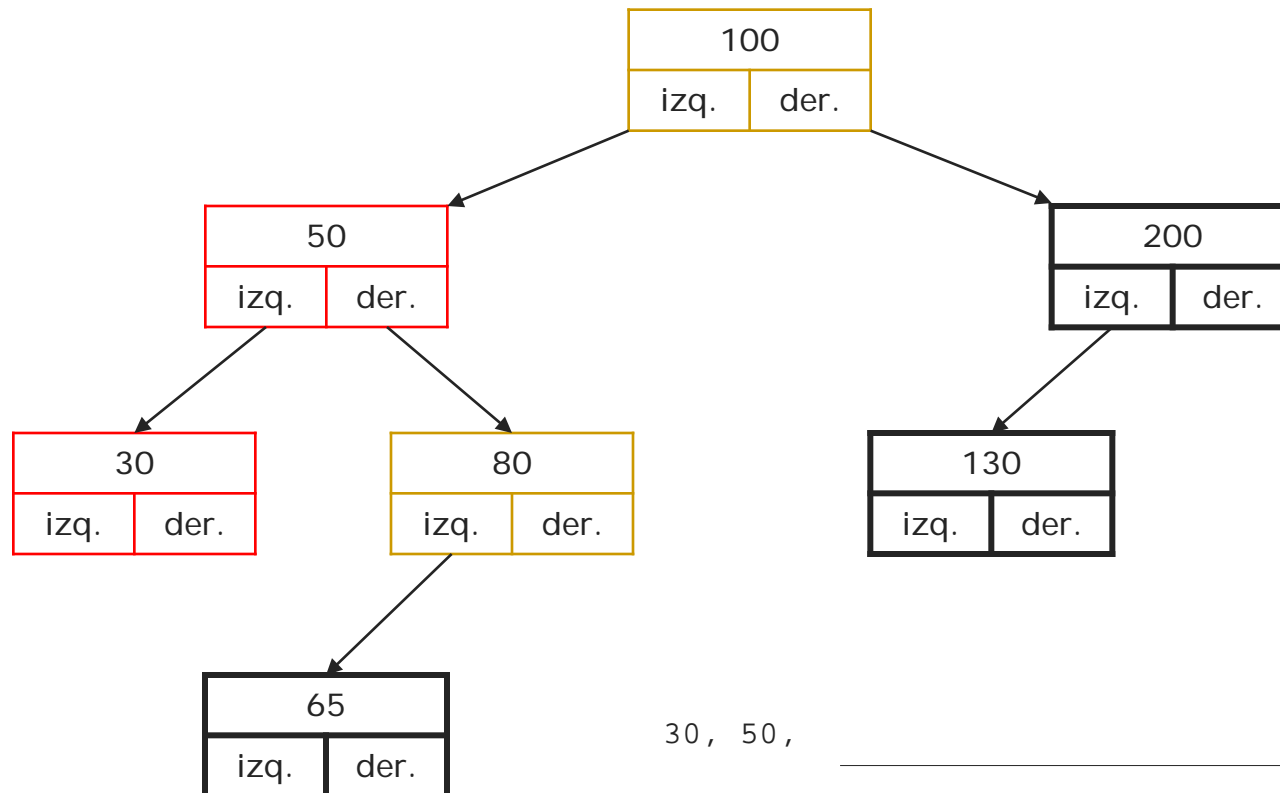
Recorridos en árboles binarios

■ Orden simétrico



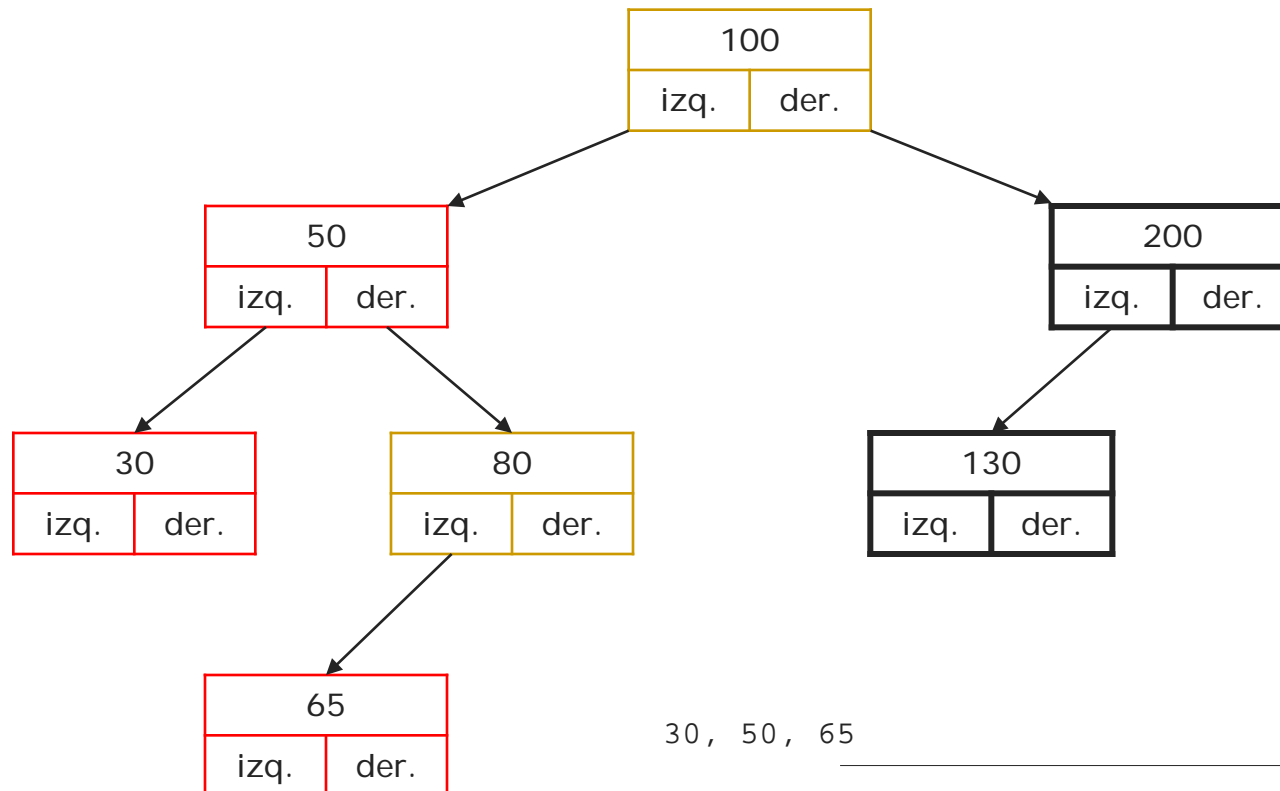
Recorridos en árboles binarios

■ Orden simétrico



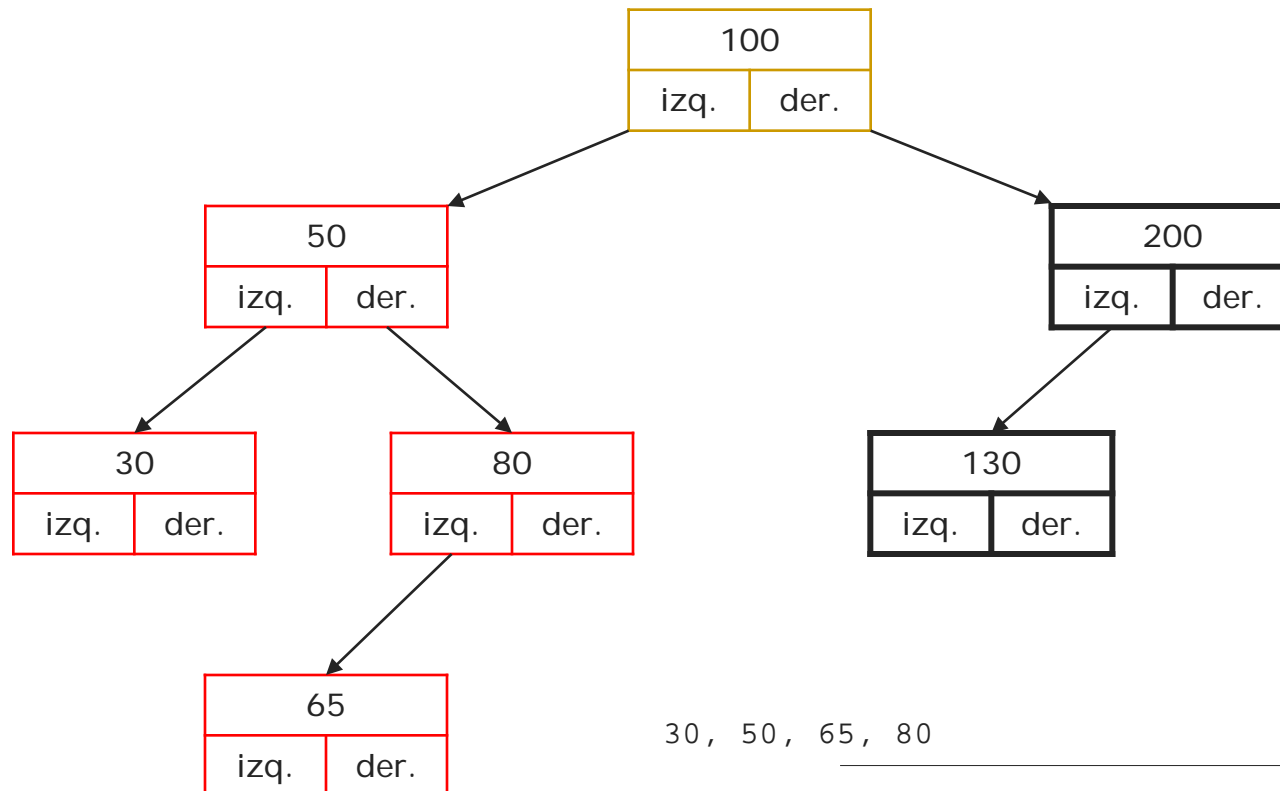
Recorridos en árboles binarios

■ Orden simétrico



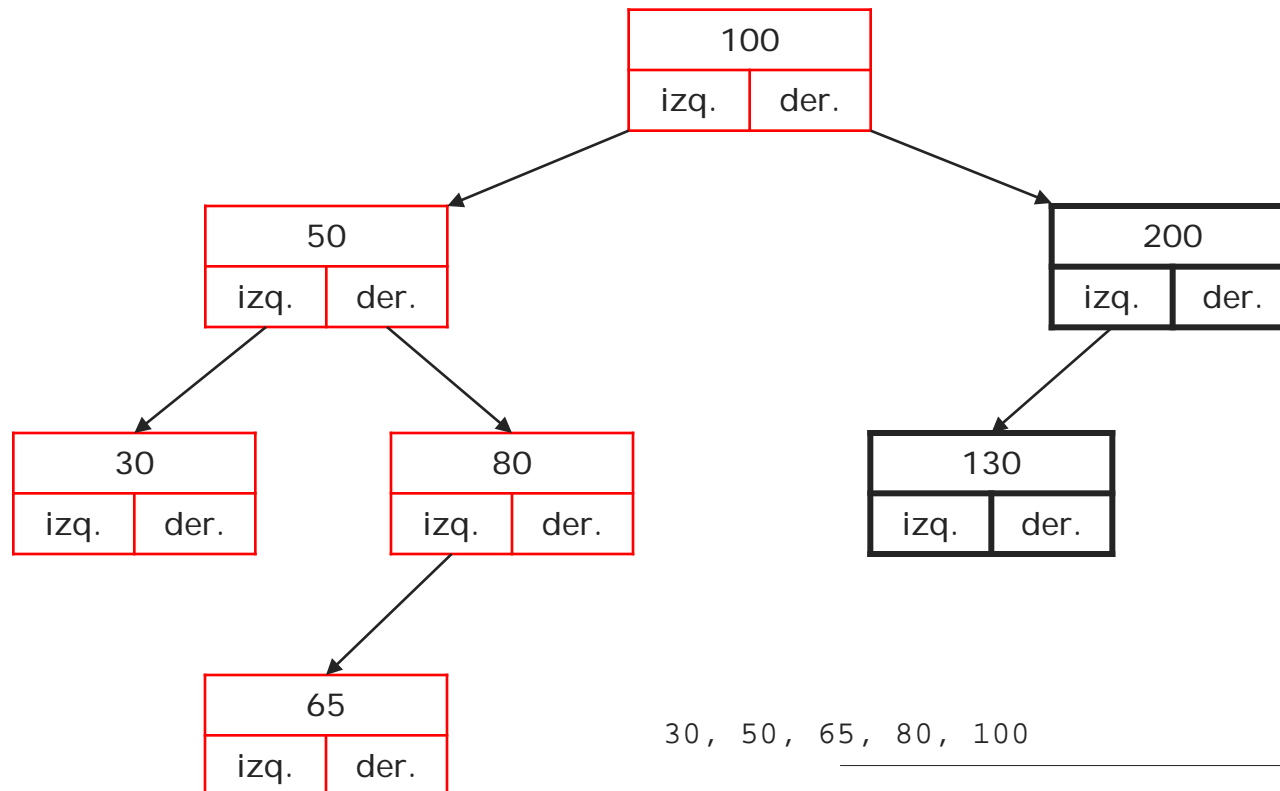
Recorridos en árboles binarios

■ Orden simétrico



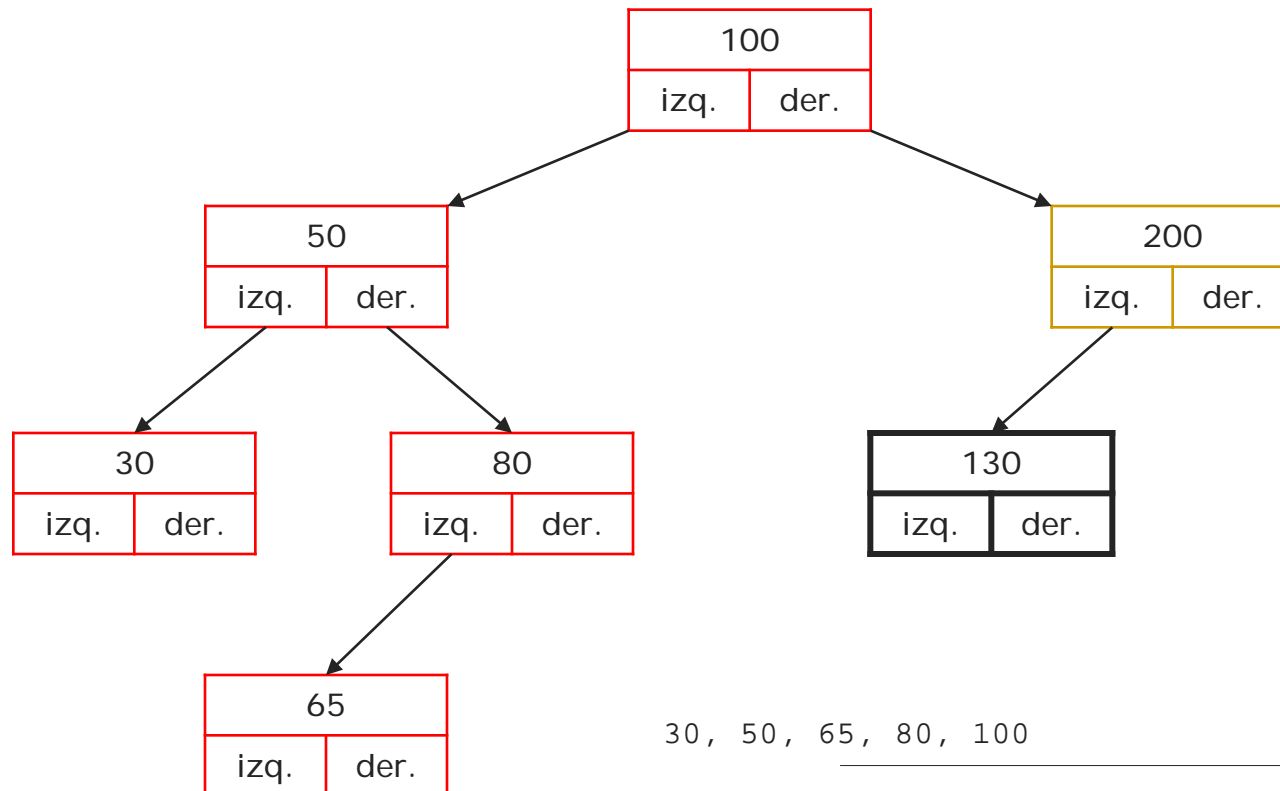
Recorridos en árboles binarios

■ Orden simétrico



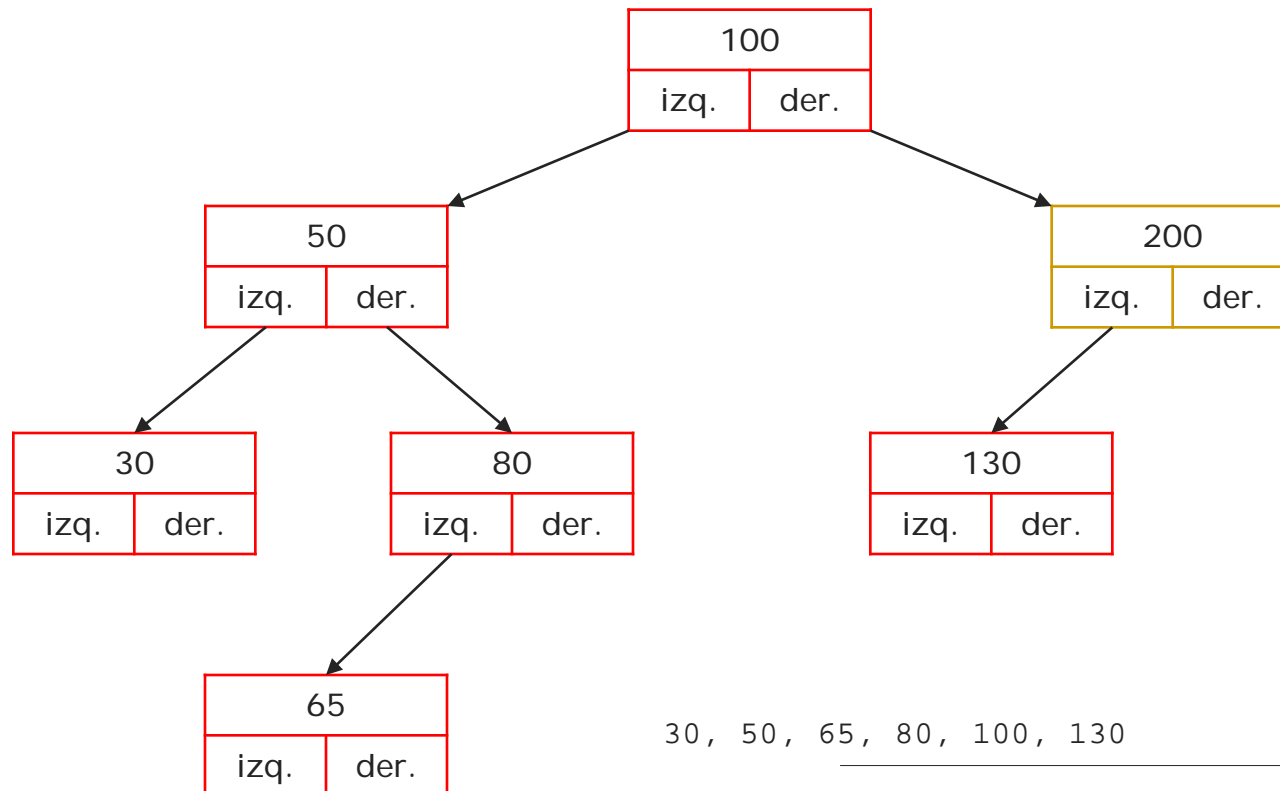
Recorridos en árboles binarios

■ Orden simétrico



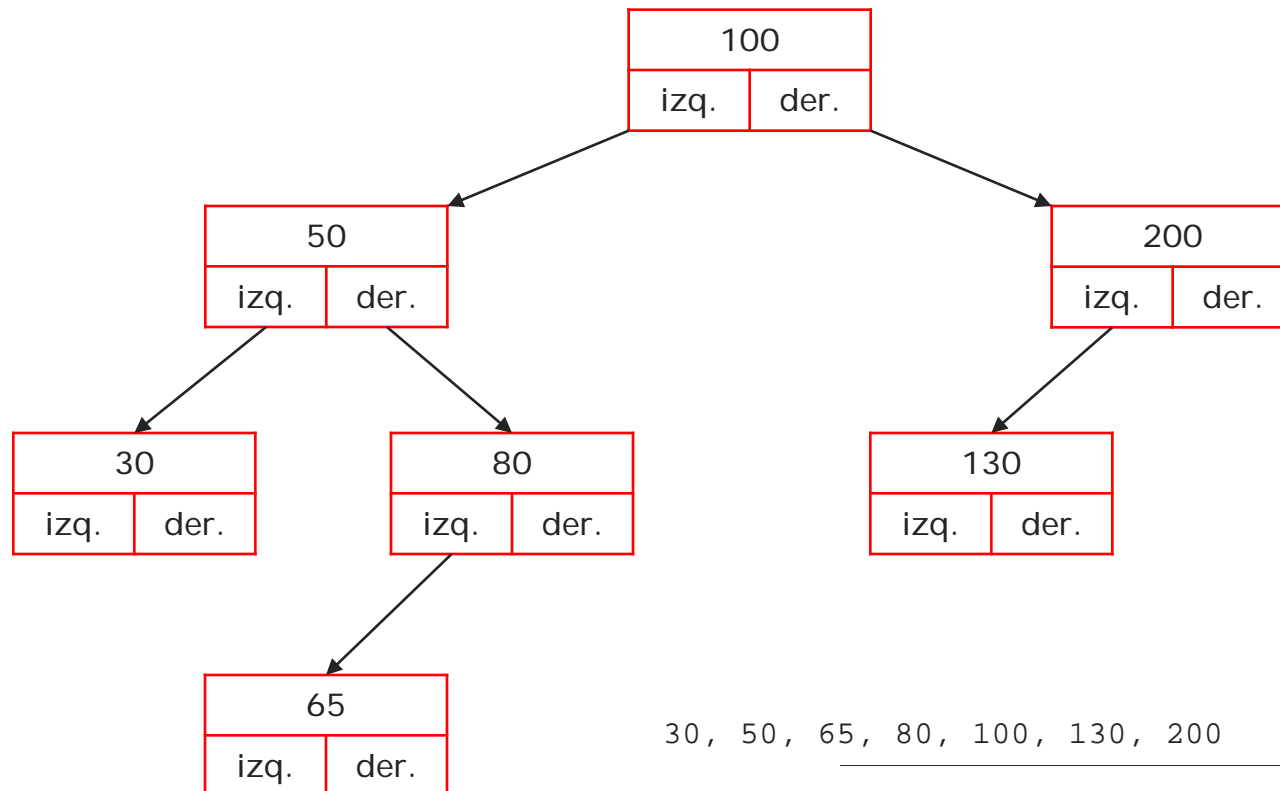
Recorridos en árboles binarios

■ Orden simétrico



Recorridos en árboles binarios

■ Orden simétrico



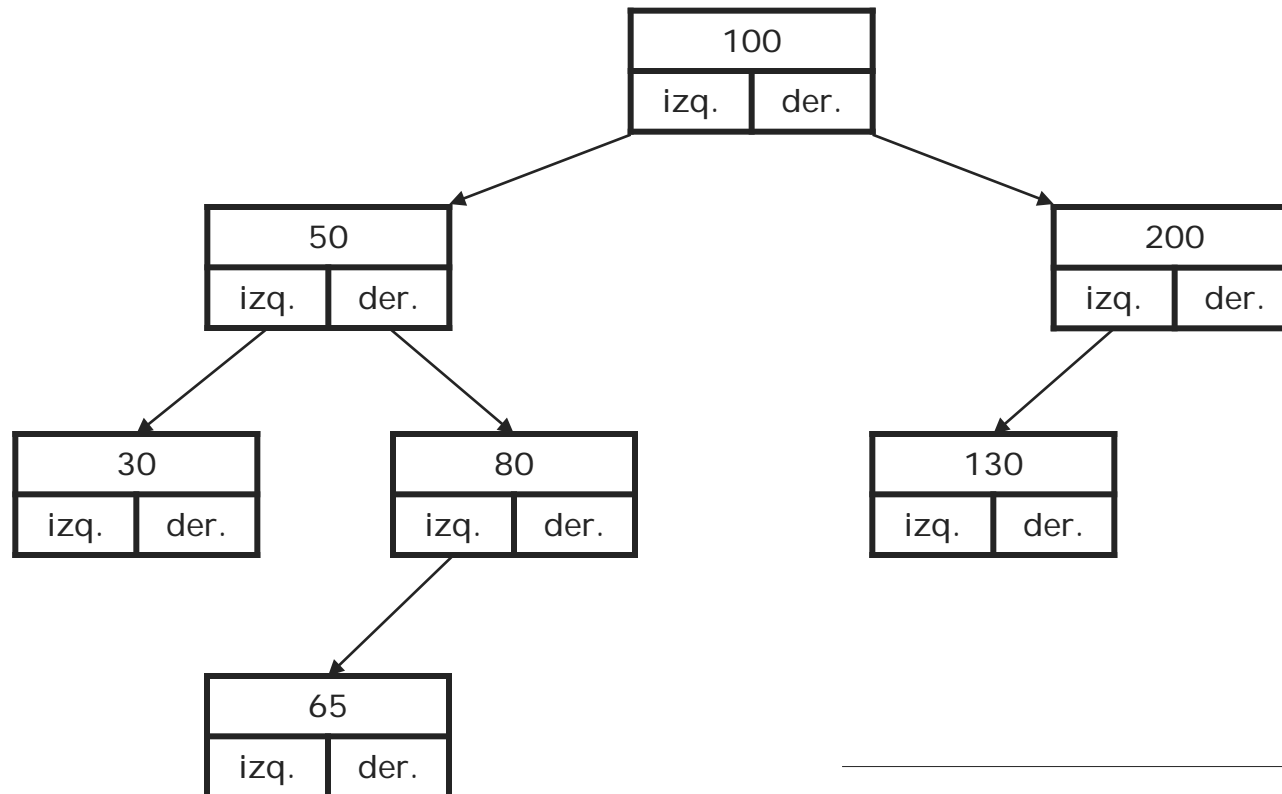
Recorridos en árboles binarios

■ Recorrido en orden posterior

- Compuesto por: recorrido en orden posterior del subárbol izquierdo, seguido del recorrido en orden posterior del subárbol derecho y seguido del nodo raíz.

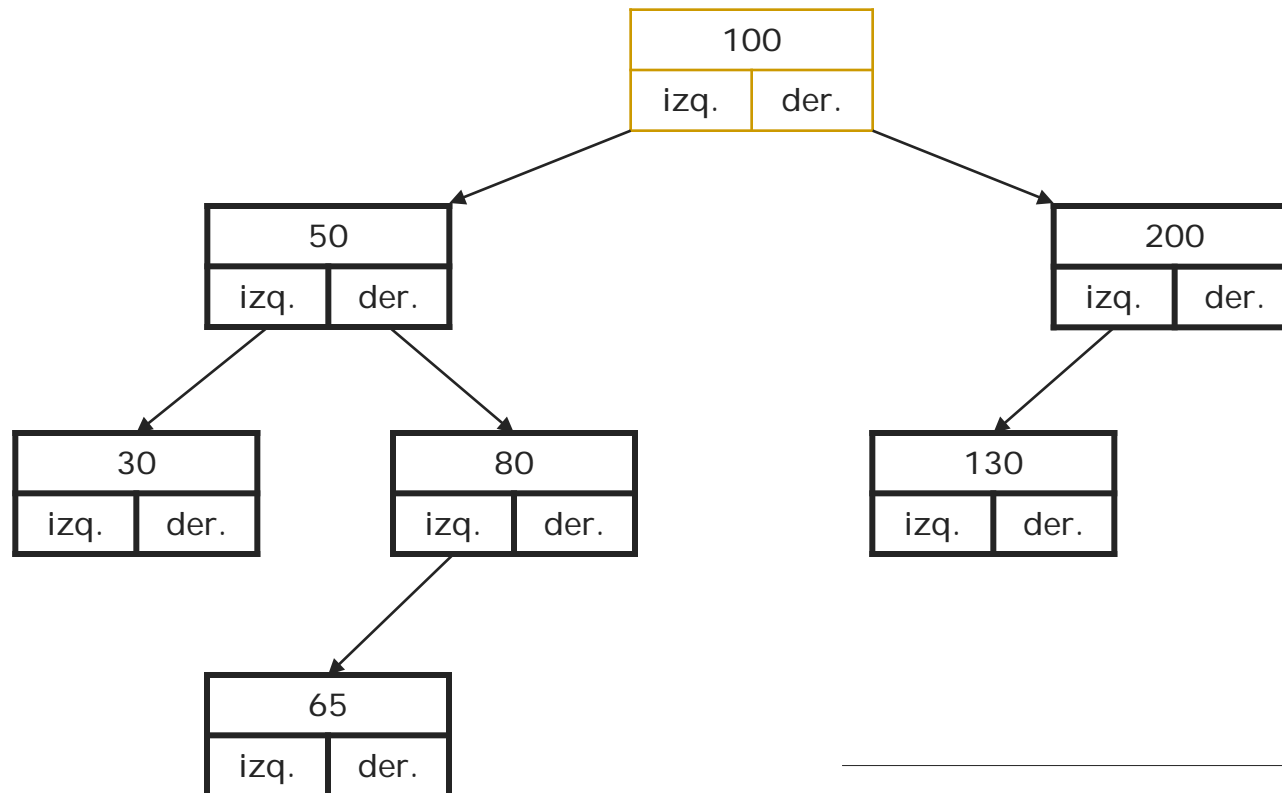
Recorridos en árboles binarios

■ Orden posterior



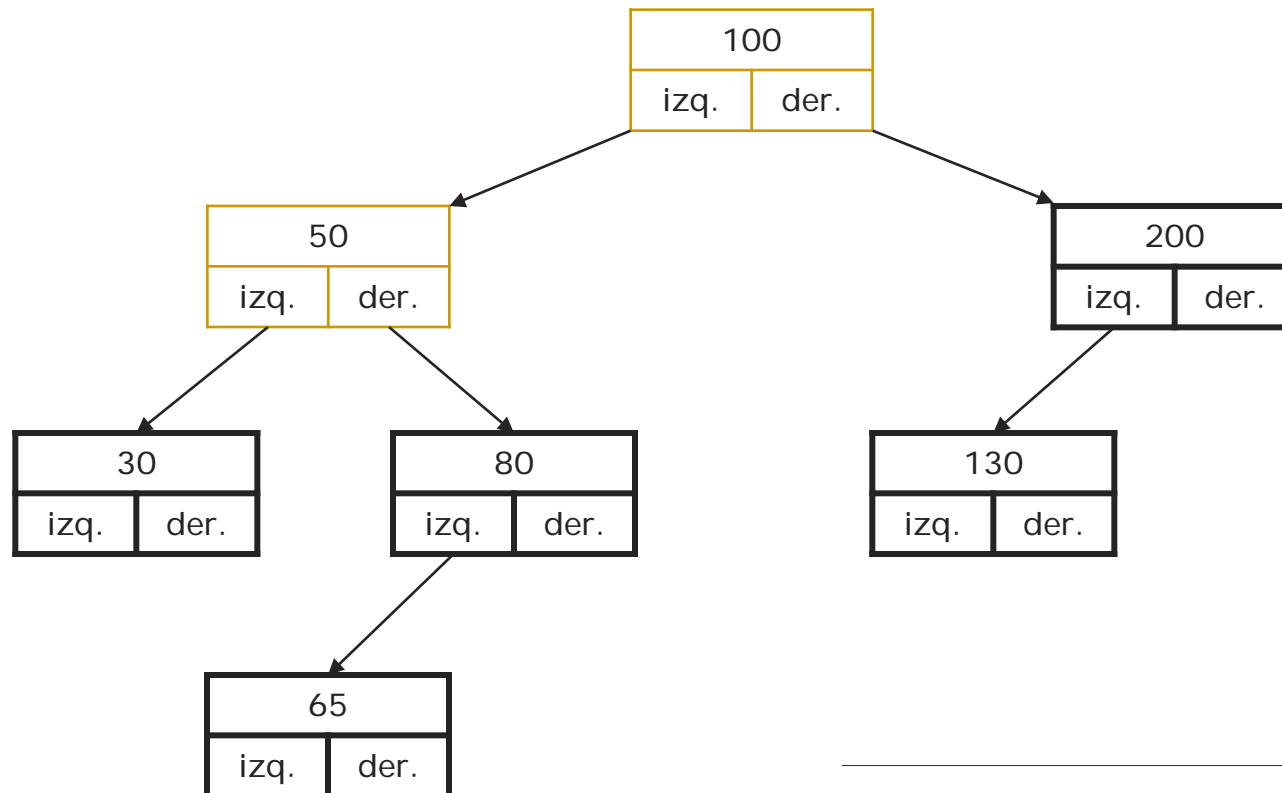
Recorridos en árboles binarios

■ Orden posterior



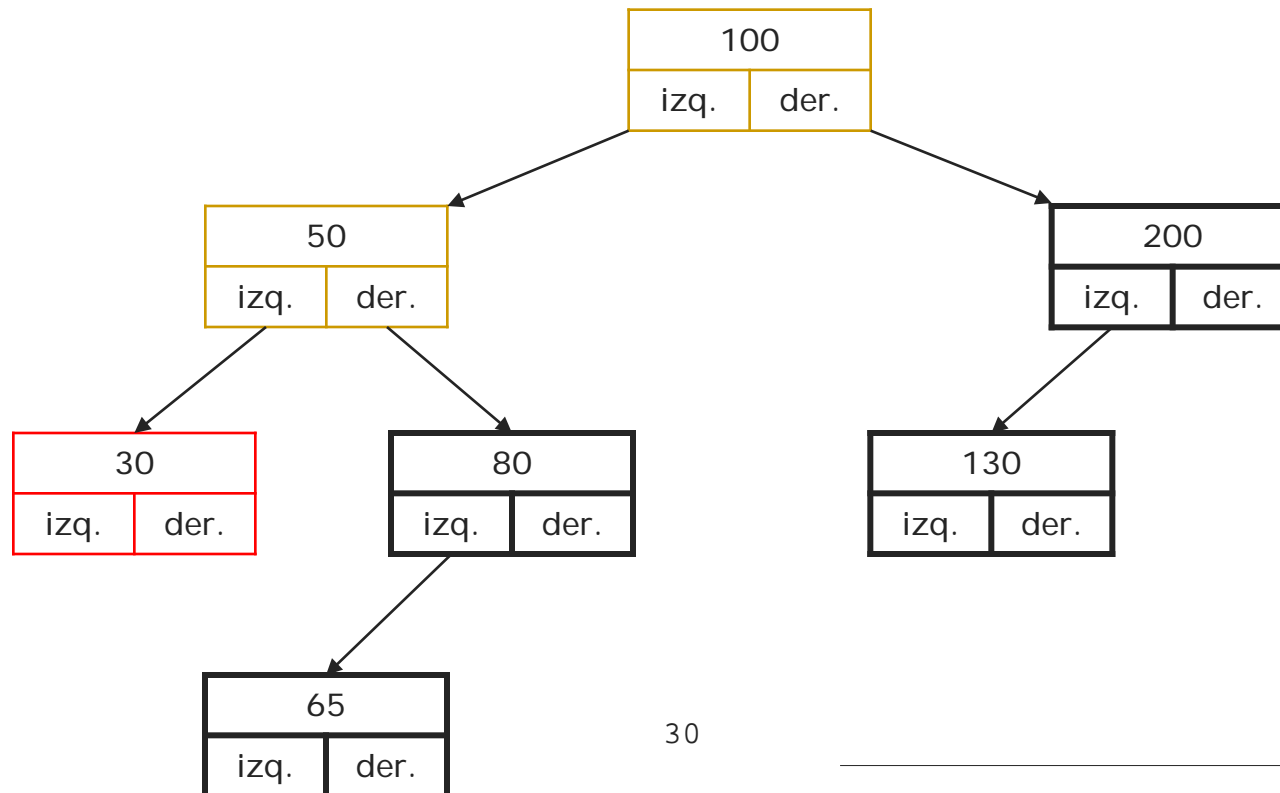
Recorridos en árboles binarios

■ Orden posterior



Recorridos en árboles binarios

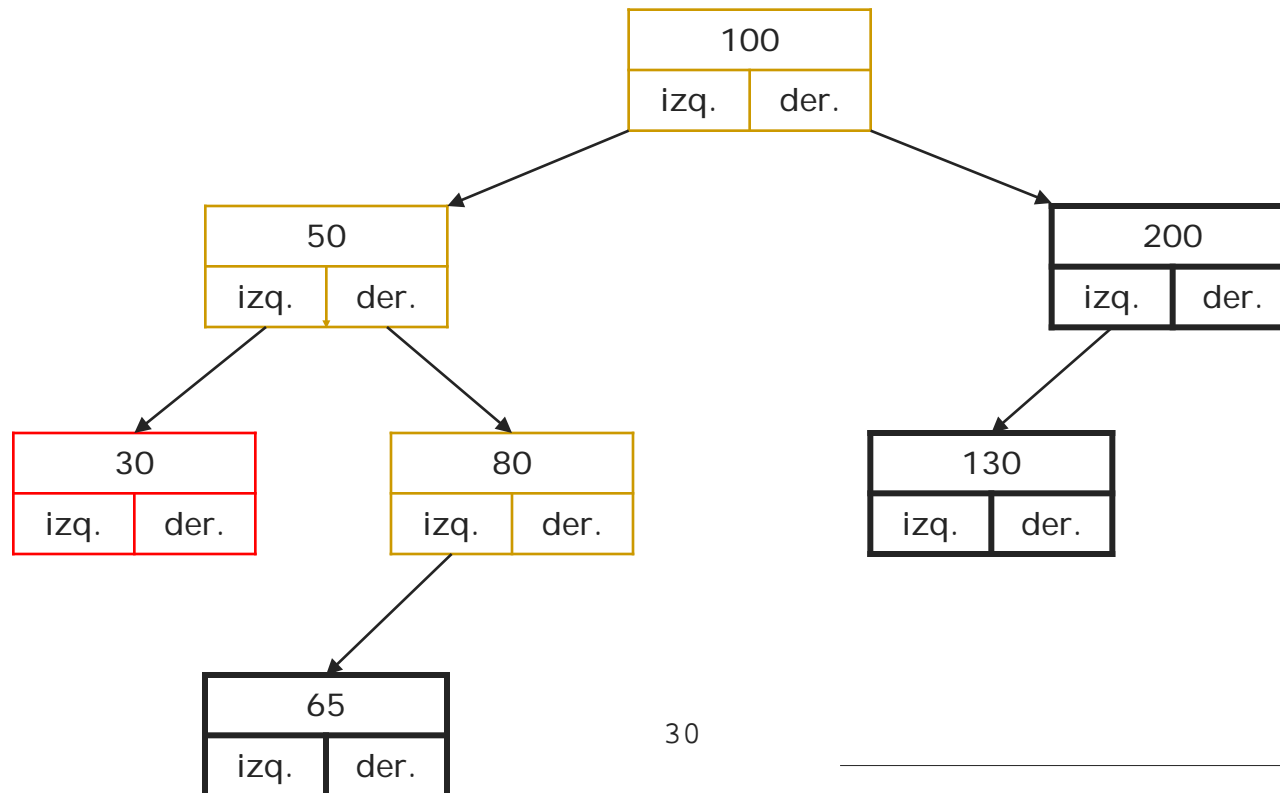
■ Orden posterior



30

Recorridos en árboles binarios

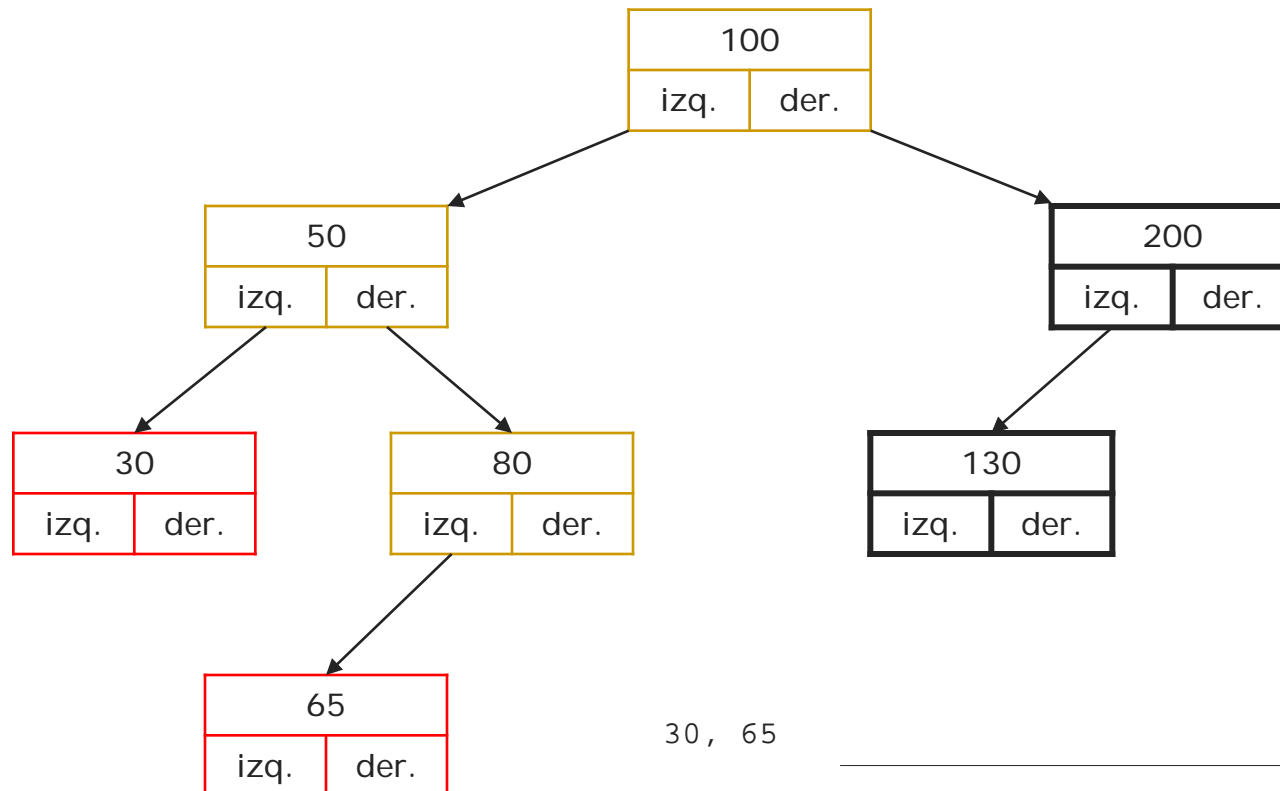
■ Orden posterior



30

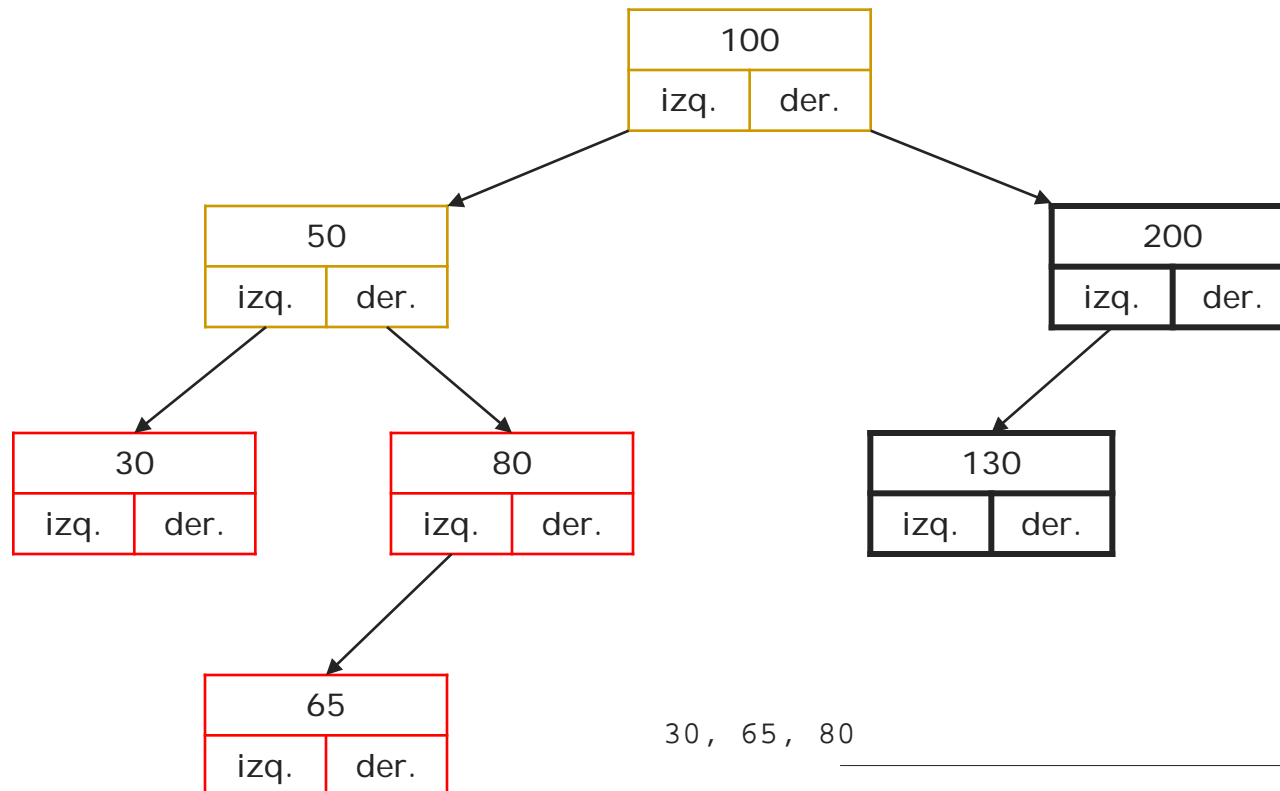
Recorridos en árboles binarios

■ Orden posterior



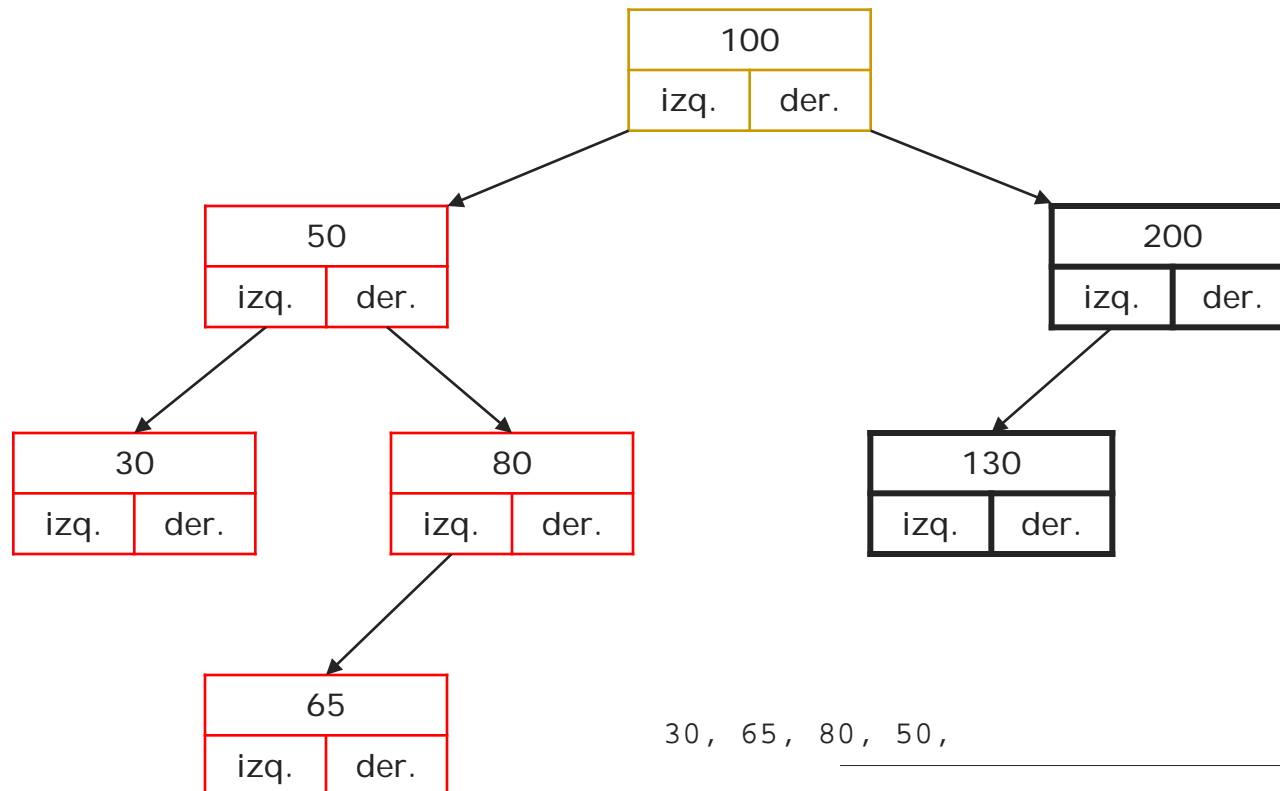
Recorridos en árboles binarios

■ Orden posterior



Recorridos en árboles binarios

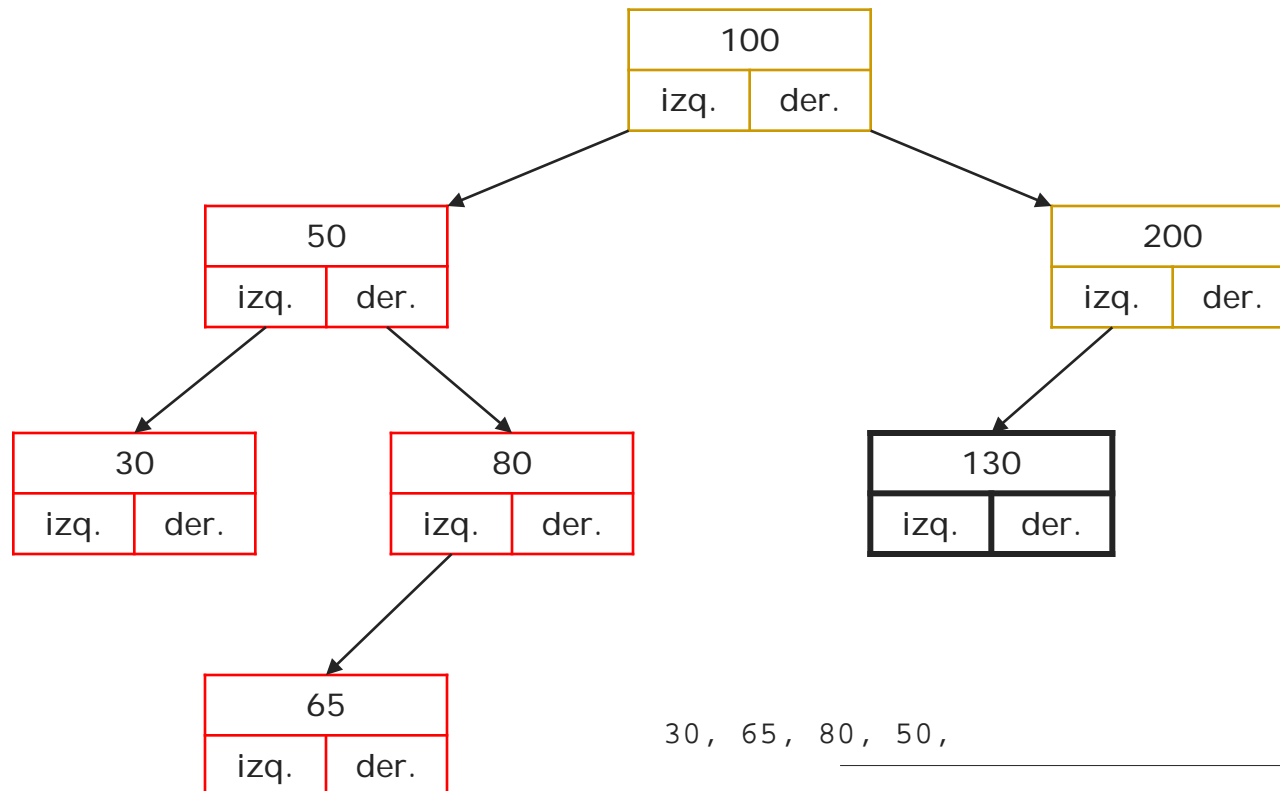
■ Orden posterior



30, 65, 80, 50,

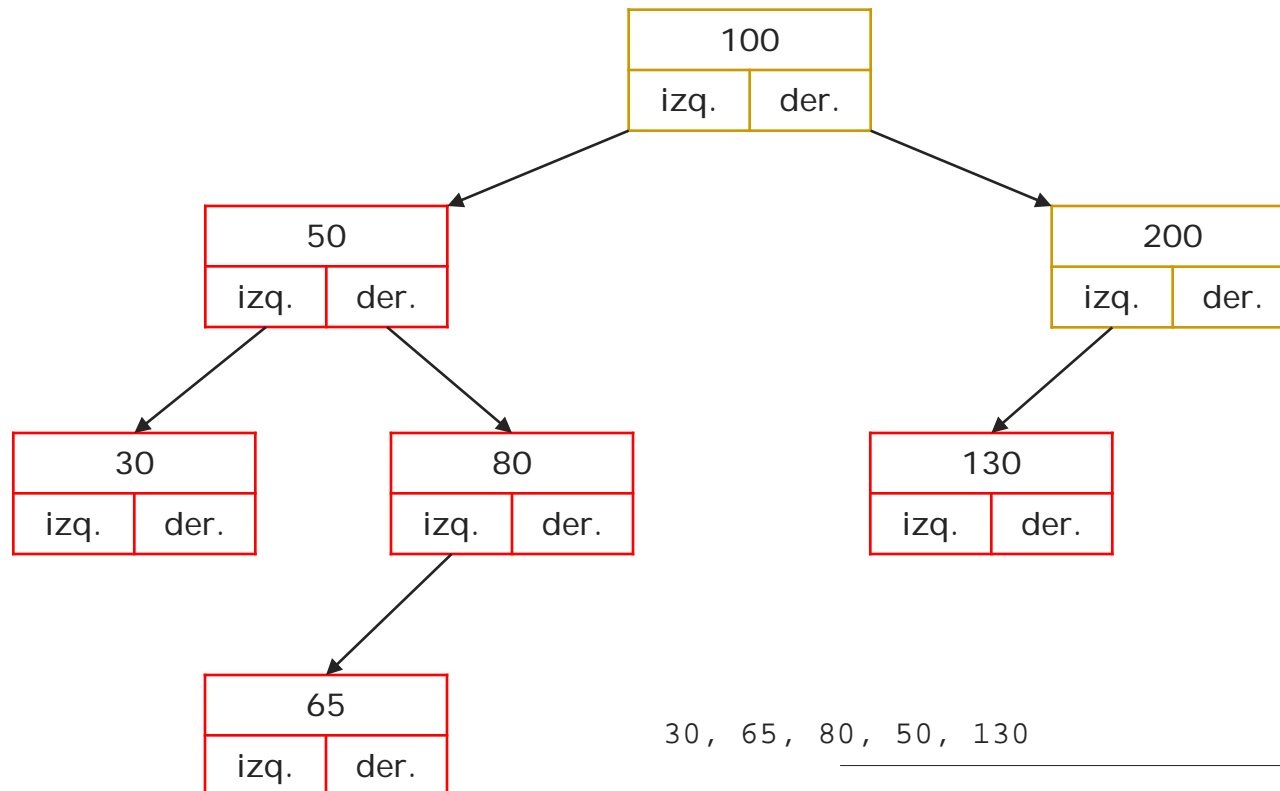
Recorridos en árboles binarios

■ Orden posterior



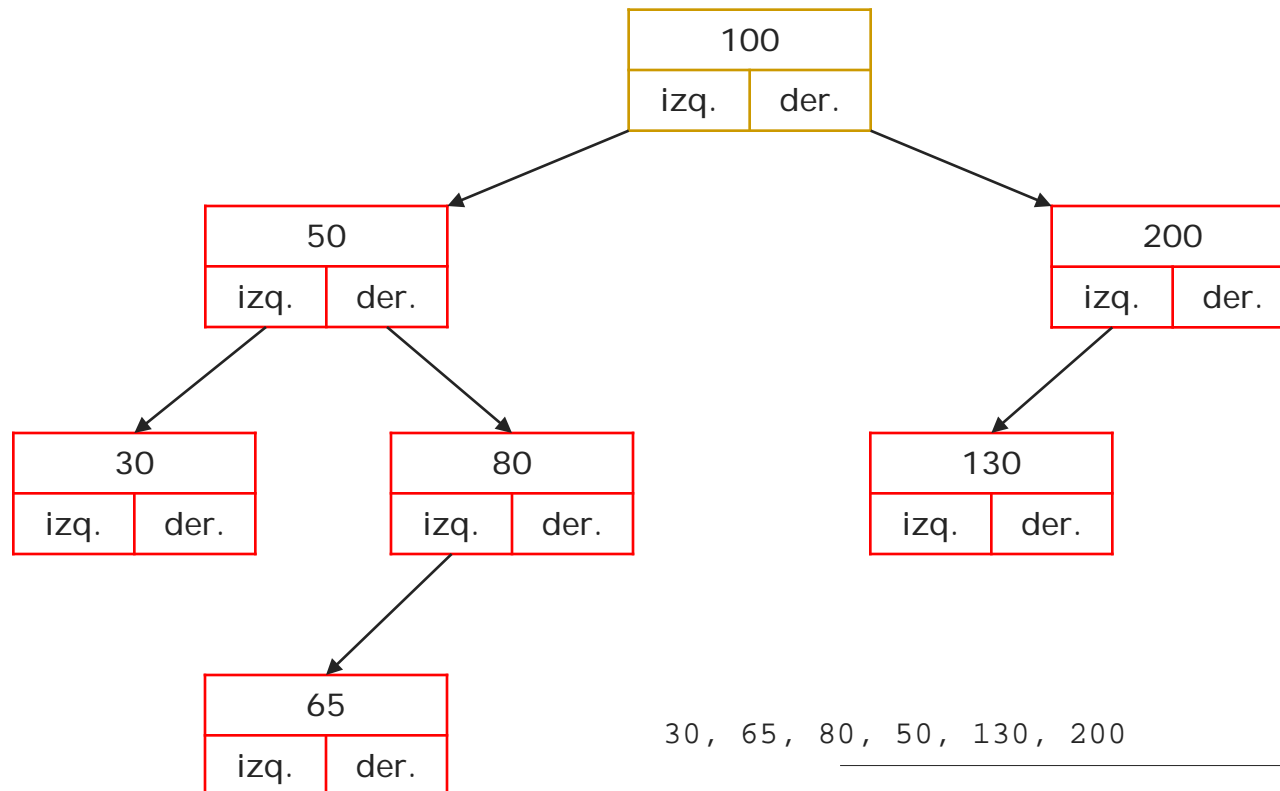
Recorridos en árboles binarios

■ Orden posterior



Recorridos en árboles binarios

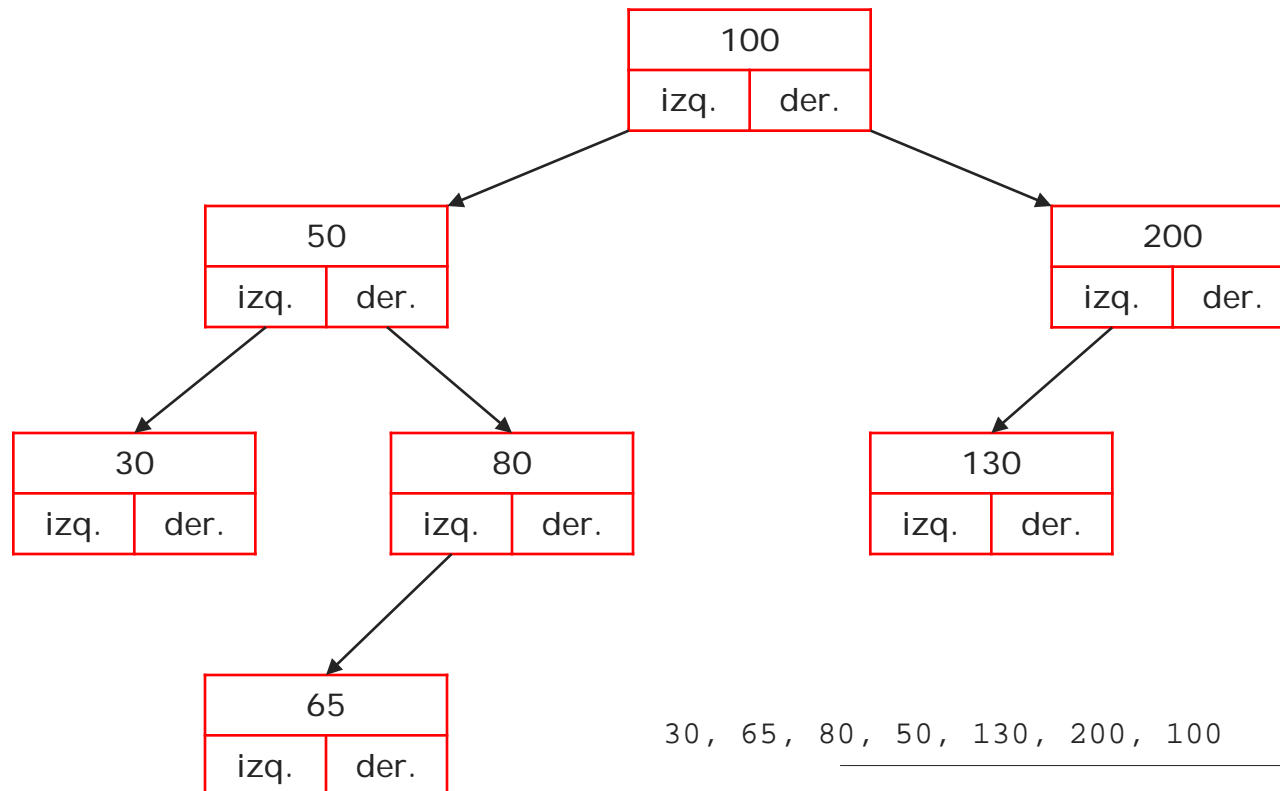
■ Orden posterior



30, 65, 80, 50, 130, 200

Recorridos en árboles binarios

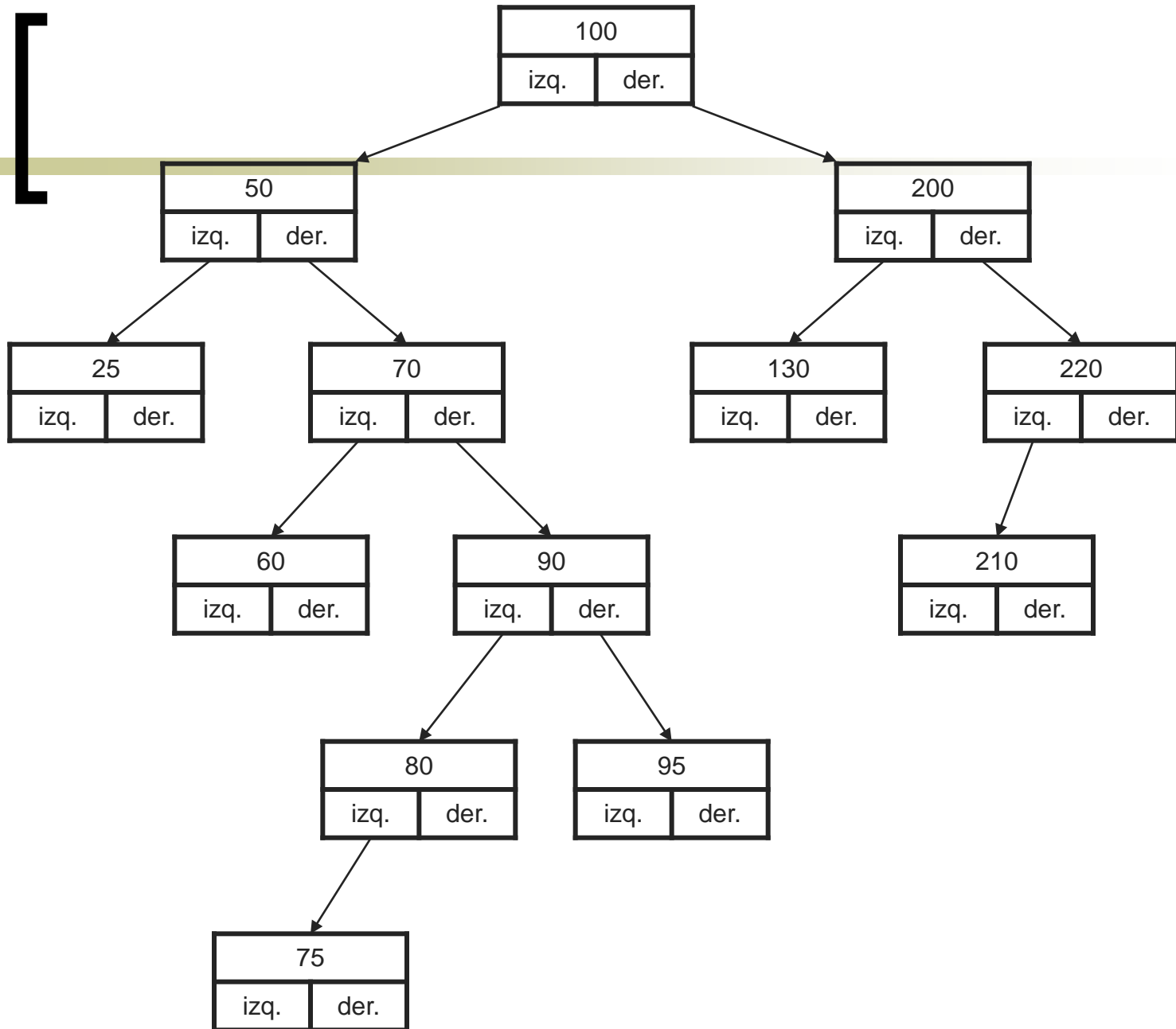
■ Orden posterior



30, 65, 80, 50, 130, 200, 100

Recorridos en árboles binarios

- Ejercicio:
 - Orden previo, simétrico y posterior del siguiente árbol binario



Solución

■ Orden previo

- 100, 50, 25, 70, 60, 90, 80, 75, 95, 200, 130, 220, 210.

■ Orden simétrico

- 25, 50, 60, 70, 75, 80, 90, 95, 100, 130, 200, 210, 220.

■ Orden posterior

- 25, 60, 75, 80, 95, 90, 70, 50, 130, 210, 220, 200, 100

Contenidos

- Introducción
- Estructura de un árbol
- Tipos de recorrido de un árbol
- **Operaciones básicas sobre un árbol**

Árboles binarios

■ Operaciones básicas:

○ Posicionamiento

- Raíz
- Hijo Izquierdo, Hijo Derecho

○ Consulta

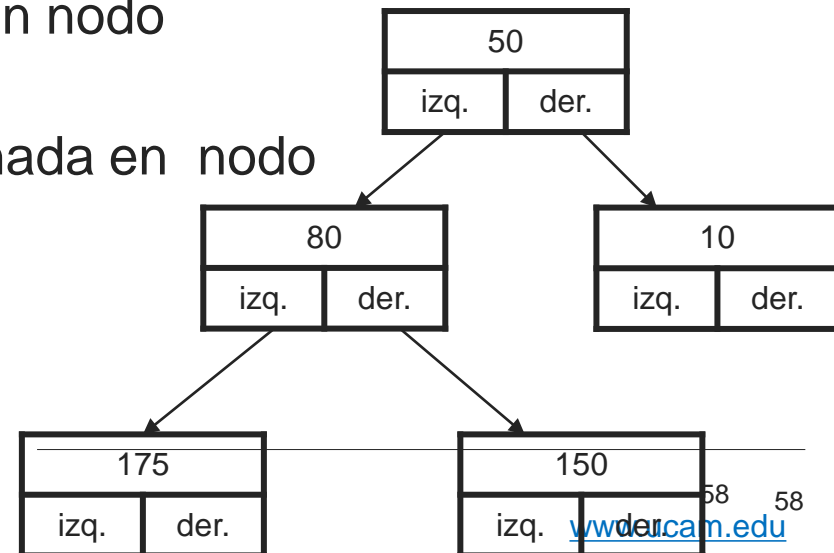
- Vacío árbol, Vacío nodo
- Información almacenada en un nodo

○ Modificación

- Modifica información almacenada en nodo
- Inserta Hijo Derecho
- Inserta Hijo Izquierdo
- Suprime Hijo Derecho
- Suprime Hijo Izquierdo
- Destruir árbol

```
typedef struct nodo_arbol_bin{
    int valor;
    struct nodo_arbol_bin *izq;
    struct nodo_arbol_bin *der;
} NODO_ARBOL_BIN, *P_NODO_ARBOL_BIN;

typedef struct nodo_arbol_bin ** ARBOL_BIN;
```



Árboles binarios: Creación

Función crea_arbol_bin

```
void crea_arbol_bin (ARBOL_BIN abin) {  
    *abin = NULL;  
}
```

Árboles binarios: Consulta

Función es_vacio_arbol_bin

```
bool es_vacio_arbol_bin( ARBOL_BIN abin ) {  
    return (*abin == NULL? true: false);  
}
```

Función es_vacio_arbol_bin

```
bool es_vacio_nodo (P_NODO_ARBOL_BIN nodo) {  
    return (nodo == NULL ? true: false);  
}
```

Función obtener_info_nodo

```
int obtener_info_nodo (P_NODO_ARBOL_BIN nodo)  
{  
    return (nodo->valor);  
}
```

Árboles binarios: Posicionamiento

Función obtener_raíz

```
P_NODO_ARBOL_BIN obtener_raiz(ARBOL_BIN abin)
{
    return ((P_NODO_ARBOL_BIN) *abin);
}
```

Función obtener_hijo_izquierdo

```
P_NODO_ARBOL_BIN obtener_hijo_izquierdo (P_NODO_ARBOL_BIN nodo)
{
    return nodo->izq;
}
```

Función obtener_hijo_derecho

```
P_NODO_ARBOL_BIN obtener_hijo_derecho (P_NODO_ARBOL_BIN nodo) {
    return nodo->der;
}
```

Árboles binarios: Modificación

Función modifica_info_nodo

```
void modifica_info_nodo (P_NODO_ARBOL_BIN nodo, int v) {  
    nodo->valor=v;  
}
```

Árboles binarios: Modificación

Función insertar_hijo_izquierdo

```
void insertar_hijo_izquierdo(ARBOL_BIN abin, P_NODO_ARBOL_BIN nodo, int v) {  
  
    P_NODO_ARBOL_BIN nuevo_nodo = (P_NODO_ARBOL_BIN)  
                                    malloc (sizeof (NODO_ARBOL_BIN));  
  
    nuevo_nodo->valor = v;  
    nuevo_nodo->der=NULL;  
    nuevo_nodo->izq=NULL;  
  
    if (es_vacio_arbol_bin(abin)){  
        // Inserción en árbol vacío (insertamos raíz)  
        (*abin)=nuevo_nodo;  
    }  
    else {  
        // Inserción normal en árbol no vacío  
        nodo->izq=nuevo_nodo;  
    }  
}
```

Árboles binarios: Modificación

Función insertar_hijo_derecho

```
void insertar_hijo_derecho(ARBOL_BIN abin, P_NODO_ARBOL_BIN nodo, int v) {  
  
    P_NODO_ARBOL_BIN nuevo_nodo = (P_NODO_ARBOL_BIN)  
                                    malloc(sizeof(NODO_ARBOL_BIN));  
  
    nuevo_nodo->valor = v;  
    nuevo_nodo->der=NULL;  
    nuevo_nodo->izq=NULL;  
  
    if (es_vacio_arbol_bin(abin)){  
        // Inserción en árbol vacío (insertamos raíz)  
        (*abin)=nuevo_nodo;  
    }  
    else {  
        // Inserción normal en árbol no vacío  
        nodo->der=nuevo_nodo;  
    }  
}
```


Árboles binarios: Modificación

Función eliminar_hijo_izquierdo

```
void eliminar_hijo_izquierdo(ARBOL_BIN abin, P_NODO_ARBOL_BIN nodo) {  
  
    P_NODO_ARBOL_BIN nodoaux = nodo->izq;  
  
    if (es_vacio_nodo(nodoaux))  
        return;  
  
    if(!es_vacio_nodo(obtener_hijo_izquierdo(nodoaux)))  
        eliminar_hijo_izquierdo(abin, obtener_hijo_izquierdo(nodoaux));  
  
    if(!es_vacio_nodo(obtener_hijo_derecho(nodoaux)))  
        eliminar_hijo_derecho(abin, obtener_hijo_derecho(nodoaux));  
  
    free(nodoaux);  
    nodo->izq=NULL;  
}
```

Árboles binarios: Modificación

Función eliminar_hijo_derecho

```
void eliminar_hijo_derecho(ARBOL_BIN abin, P_NODO_ARBOL_BIN nodo) {  
  
    P_NODO_ARBOL_BIN nodoaux = nodo->der;  
  
    if (es_vacio_nodo(nodoaux))  
        return;  
  
    if(!es_vacio_nodo(obtener_hijo_izquierdo(nodoaux)))  
        eliminar_hijo_izquierdo(abin, obtener_hijo_izquierdo(nodoaux));  
  
    if(!es_vacio_nodo(obtener_hijo_derecho(nodoaux)))  
        eliminar_hijo_derecho(abin, obtener_hijo_derecho(nodoaux));  
  
    free(nodoaux);  
    nodo->der=NULL;  
}
```

Árboles binarios: Modificación

Función destruir_arbol_bin

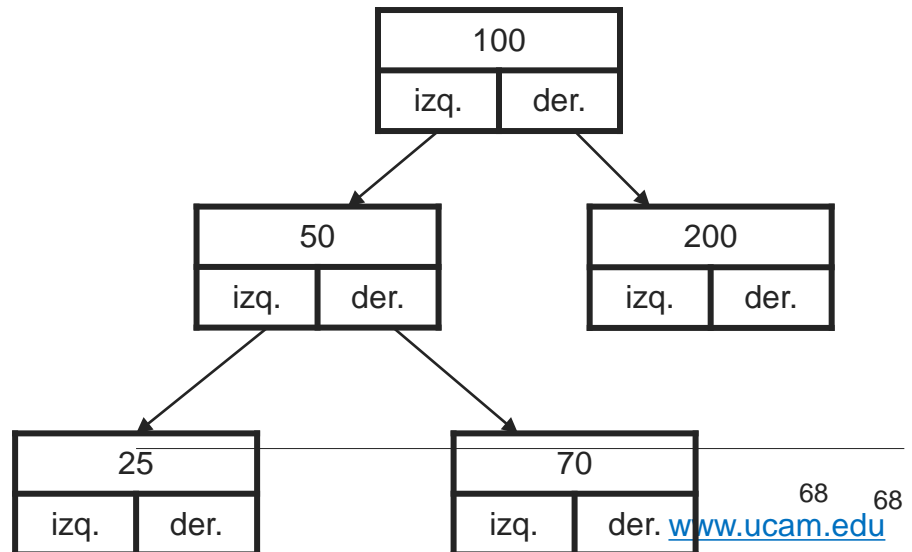
```
void destruir_arbol_bin(ARBOL_BIN abin) {  
  
    P_NODO_ARBOL_BIN raiz = *abin ;  
  
    if(!es_vacio_nodo(obtener_hijo_izquierdo(raiz)))  
        eliminar_hijo_izquierdo(abin, obtener_hijo_izquierdo(raiz));  
  
    if(!es_vacio_nodo(obtener_hijo_derecho(raiz)))  
        eliminar_hijo_derecho(abin, obtener_hijo_derecho(raiz));  
  
    free(raiz);  
    *abin=NULL;  
}
```

Árboles binarios ordenados

■ Operaciones básicas:

- Insertar
- Pertenece
- Encontrar
- Padre de
- Borrar

```
typedef struct nodo_arbol{  
    int valor;  
    struct nodo_arbol *izq;  
    struct nodo_arbol *der;  
} NODO_ARBOL, *P_NODO_ARBOL
```



Árboles binarios ordenados

Función Insertar

```
P_NODO_ARBOL insertar(P_NODO_ARBOL arbol, int i){  
  
    P_NODO_ARBOL p;  
    if (arbol == NULL){  
        p = (P_NODO_ARBOL) malloc(sizeof (struct nodo_arbol));  
        p->izq = p->der = NULL;  
        p->valor = i;  
        return p;  
    }  
    if (arbol->valor == i){  
        return arbol;  
    }  
    if (arbol->valor > i){  
        arbol->izq = insertar(arbol->izq, i);  
    }else{  
        arbol->der = insertar(arbol->der, i);  
    }  
    return arbol;  
}
```

Árboles binarios ordenados

Función Pertenece

```
int pertenece (P_NODO_ARBOL arbol, int i){  
    if (arbol == NULL){  
        return 0;  
    }  
    if (arbol->valor == i) return 1;  
  
    if (arbol->valor > i)  
        return pertenece(arbol->izq, i);  
    else  
        return pertenece(arbol->der, i);  
}
```

Árboles binarios ordenados

Función Encontrar

```
P_NODO_ARBOL encontrar (P_NODO_ARBOL arbol, int i){  
    if (arbol == NULL){  
        return NULL;  
    }  
    if (arbol->valor == i) return arbol;  
  
    if (arbol->valor > i)  
        return encontrar(arbol->izq, i);  
    else  
        return encontrar(arbol->der, i);  
}
```

Árboles binarios ordenados

Función Padre_De

```
P_NODO_ARBOL padre_de (P_NODO_ARBOL arbol, P_NODO_ARBOL nodo){  
    if (arbol == NULL){  
        return NULL;  
    }  
    if (arbol->izq == nodo || arbol->der == nodo)  
        return arbol;  
  
    else if (arbol->valor > nodo->valor)  
        return padre_de(arbol->izq, nodo);  
    else  
        return padre_de (arbol->der, nodo);  
}
```


Árboles binarios ordenados

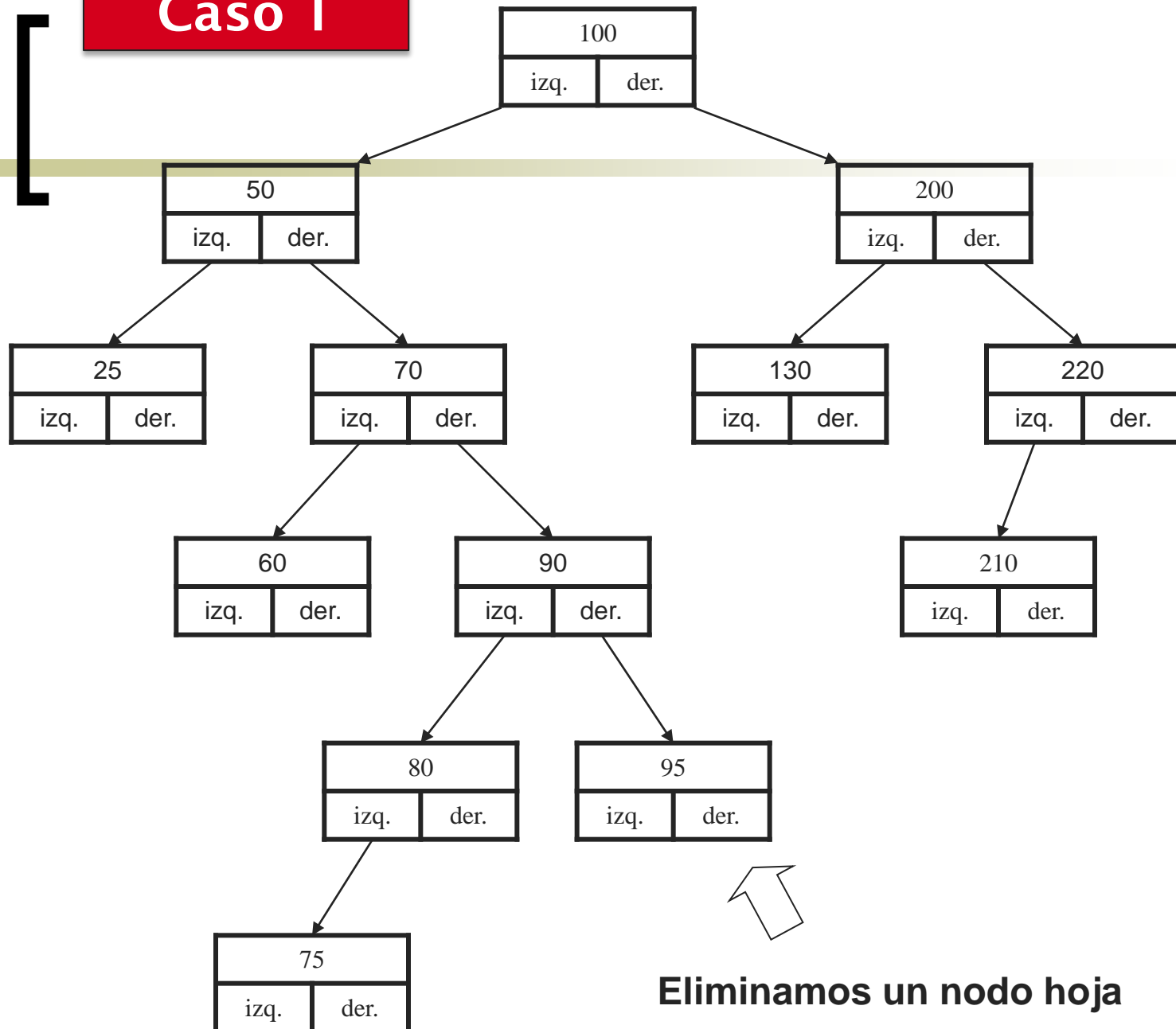
Función Eliminar

- La función de eliminar es más **complicada** ya que el árbol debe de quedar ordenado
- Habrá tres casos:

Nº	Caso	Solución
1	El nodo sea hoja	Se borra directamente
2	El nodo tenga una rama vacía	Se borra y se deja la otra rama "colgando" de donde lo hace ahora el nodo
3	El nodo tenga nodos en ambas ramas	Se busca el mayor de los menores para sustituirlo por el que se quiere eliminar.

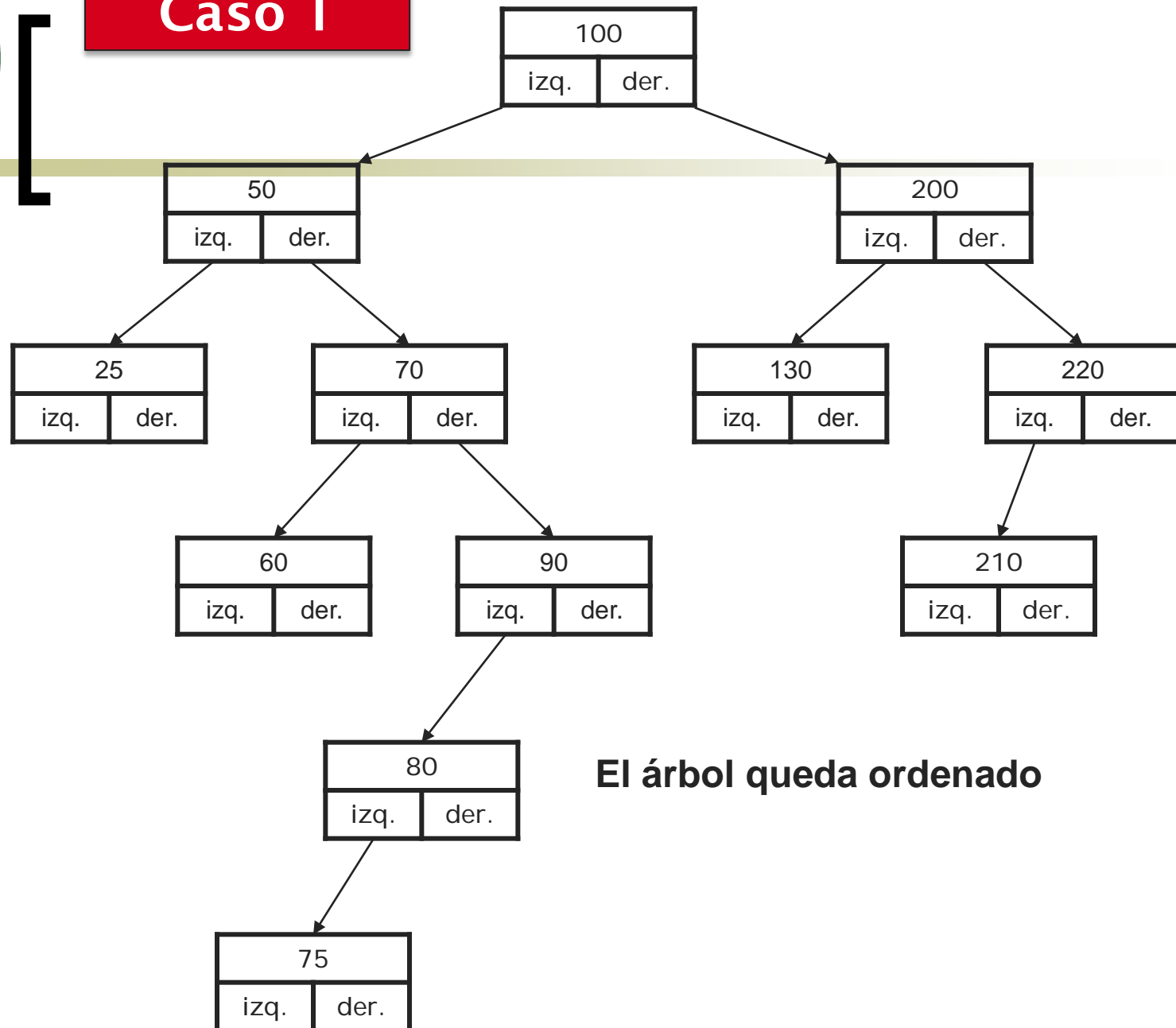


Caso 1



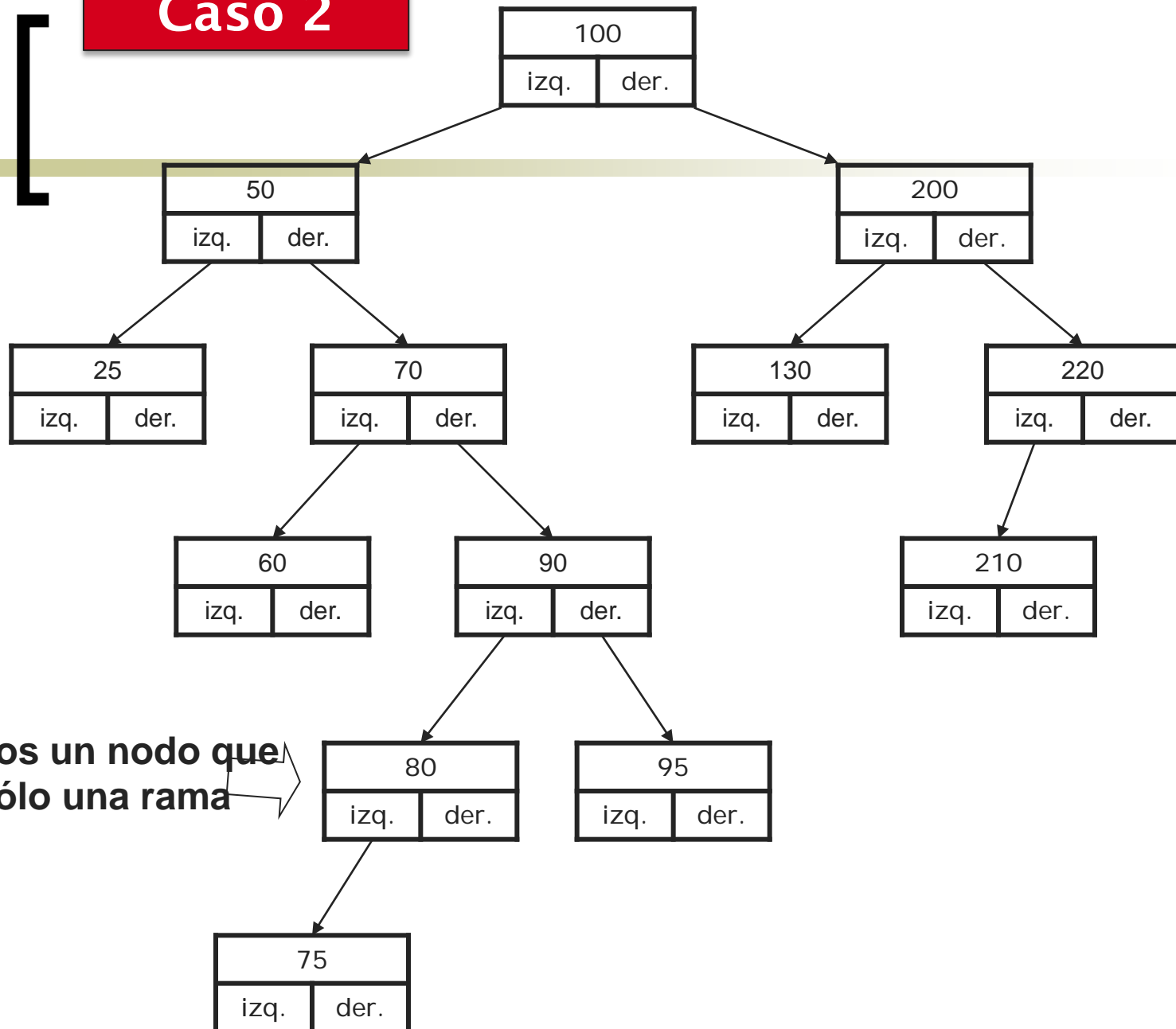


Caso 1



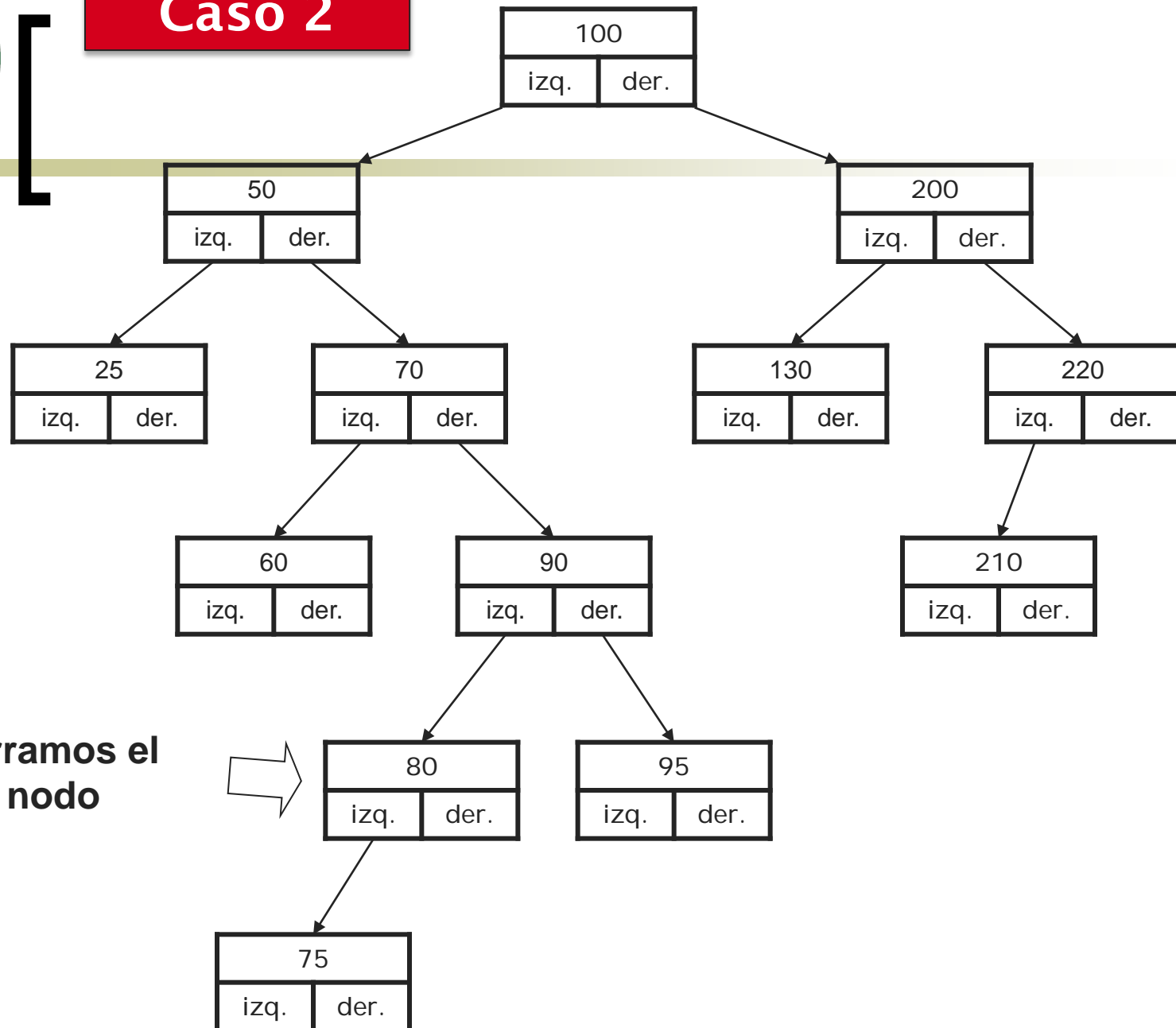


Caso 2



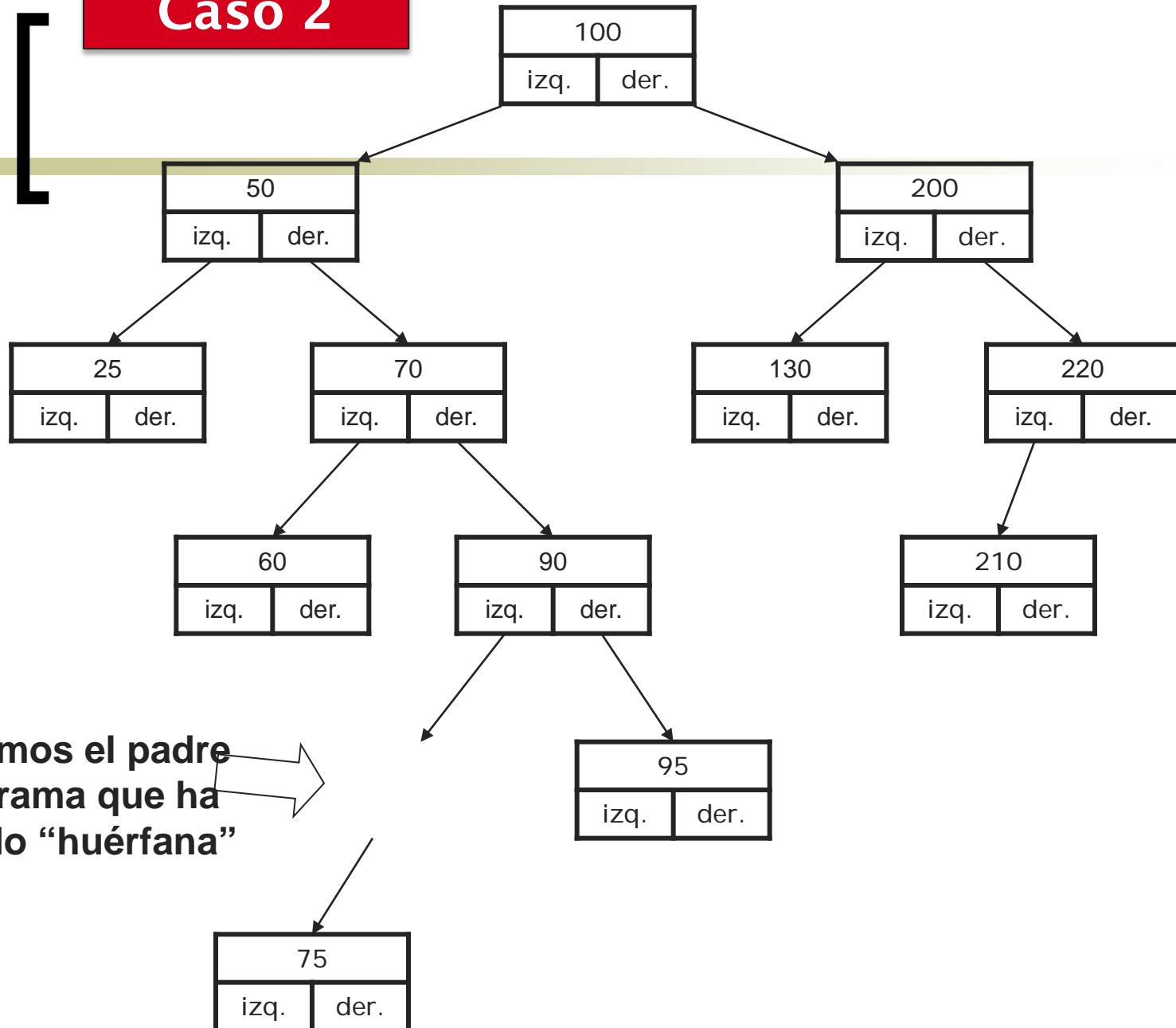


Caso 2



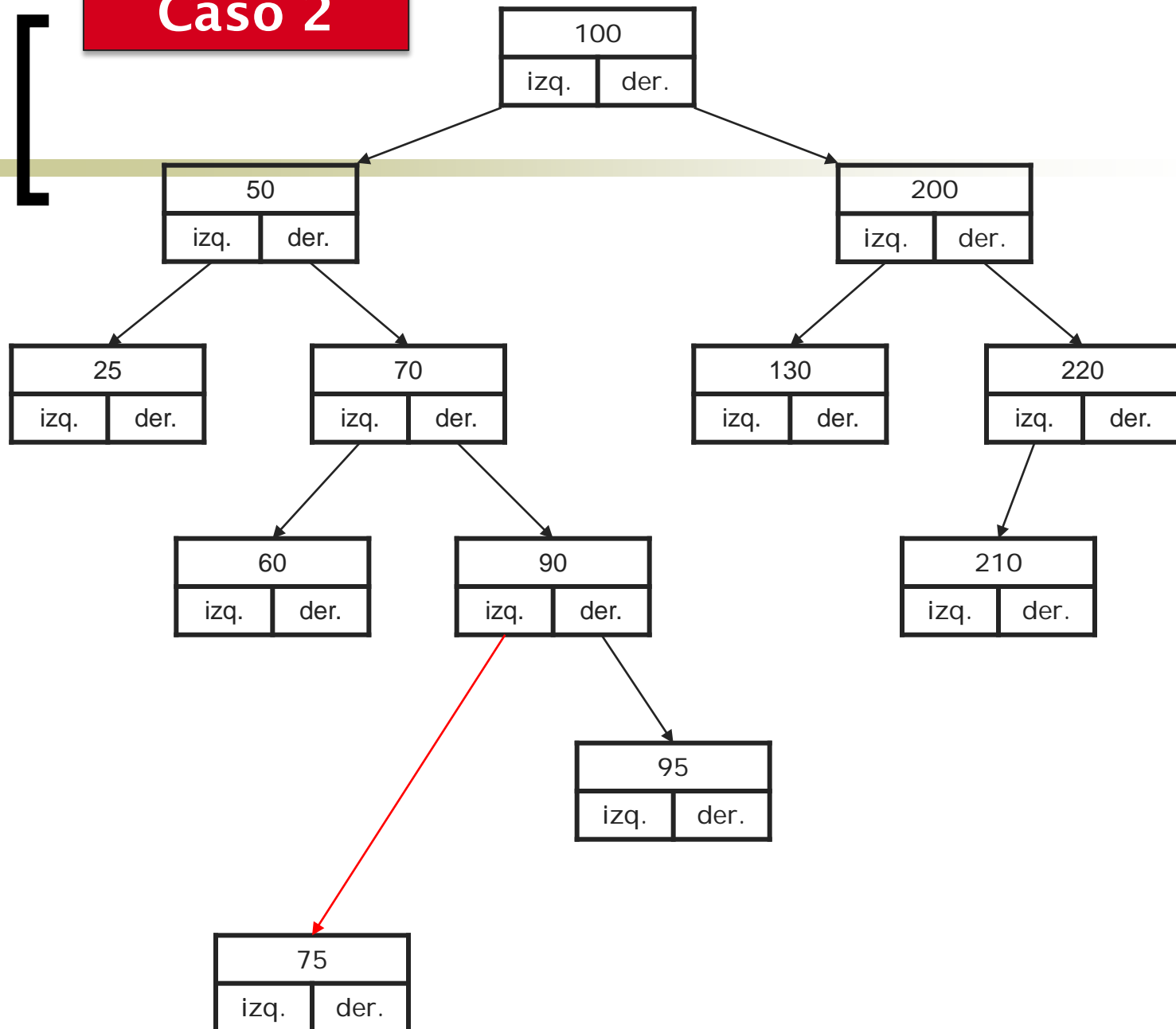


Caso 2



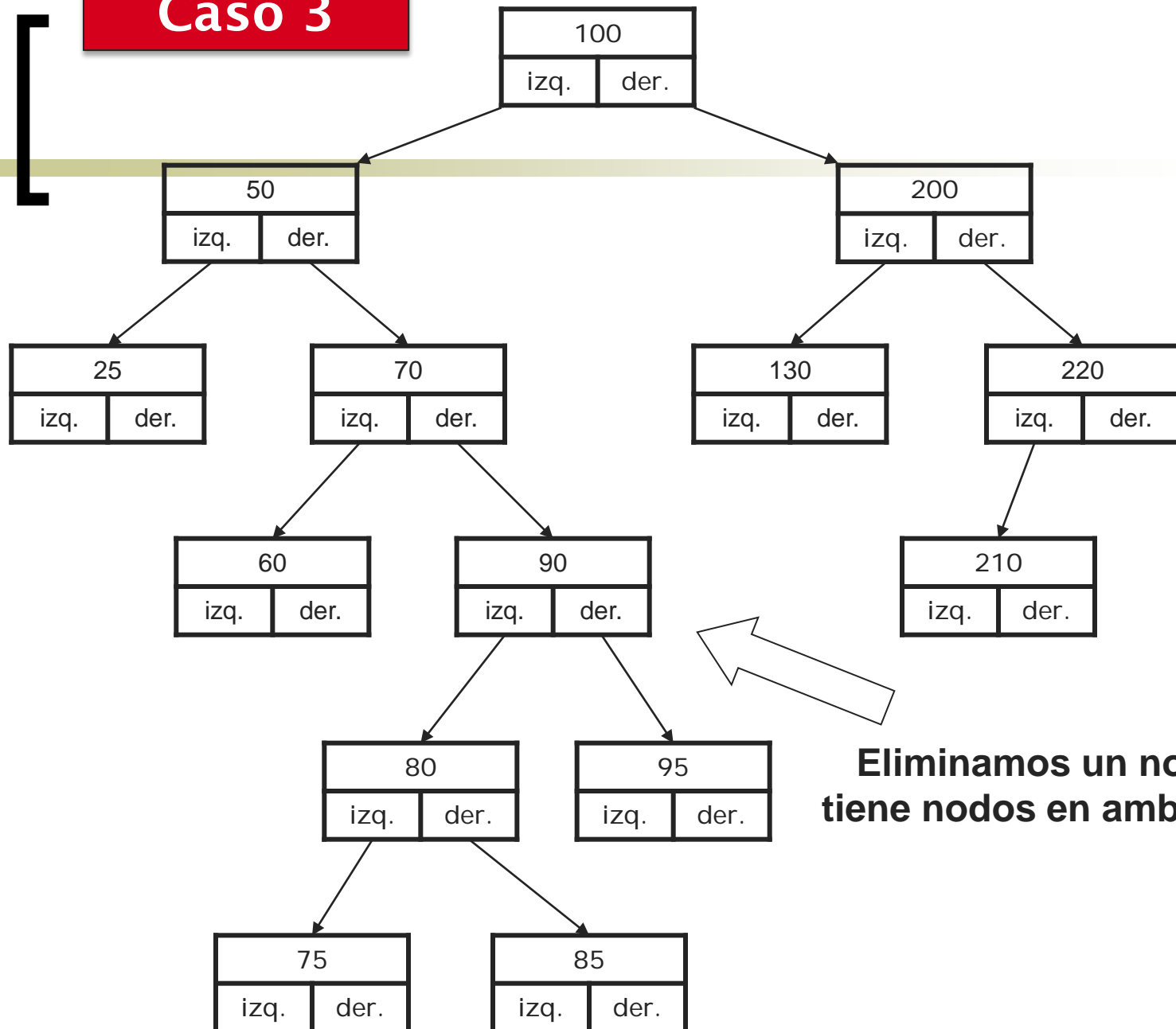


Caso 2





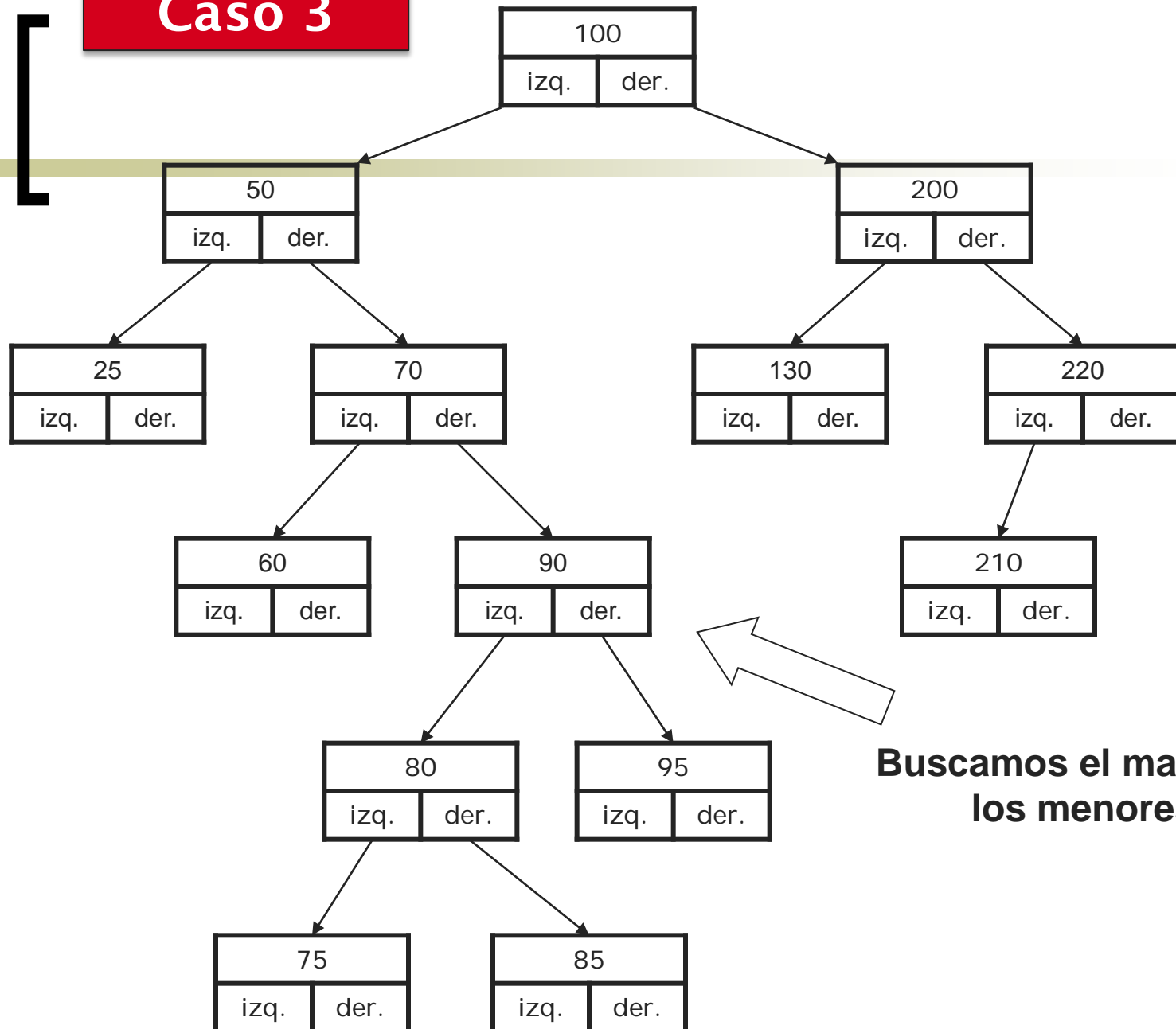
Caso 3



**Eliminamos un nodo que
tiene nodos en ambas ramas**

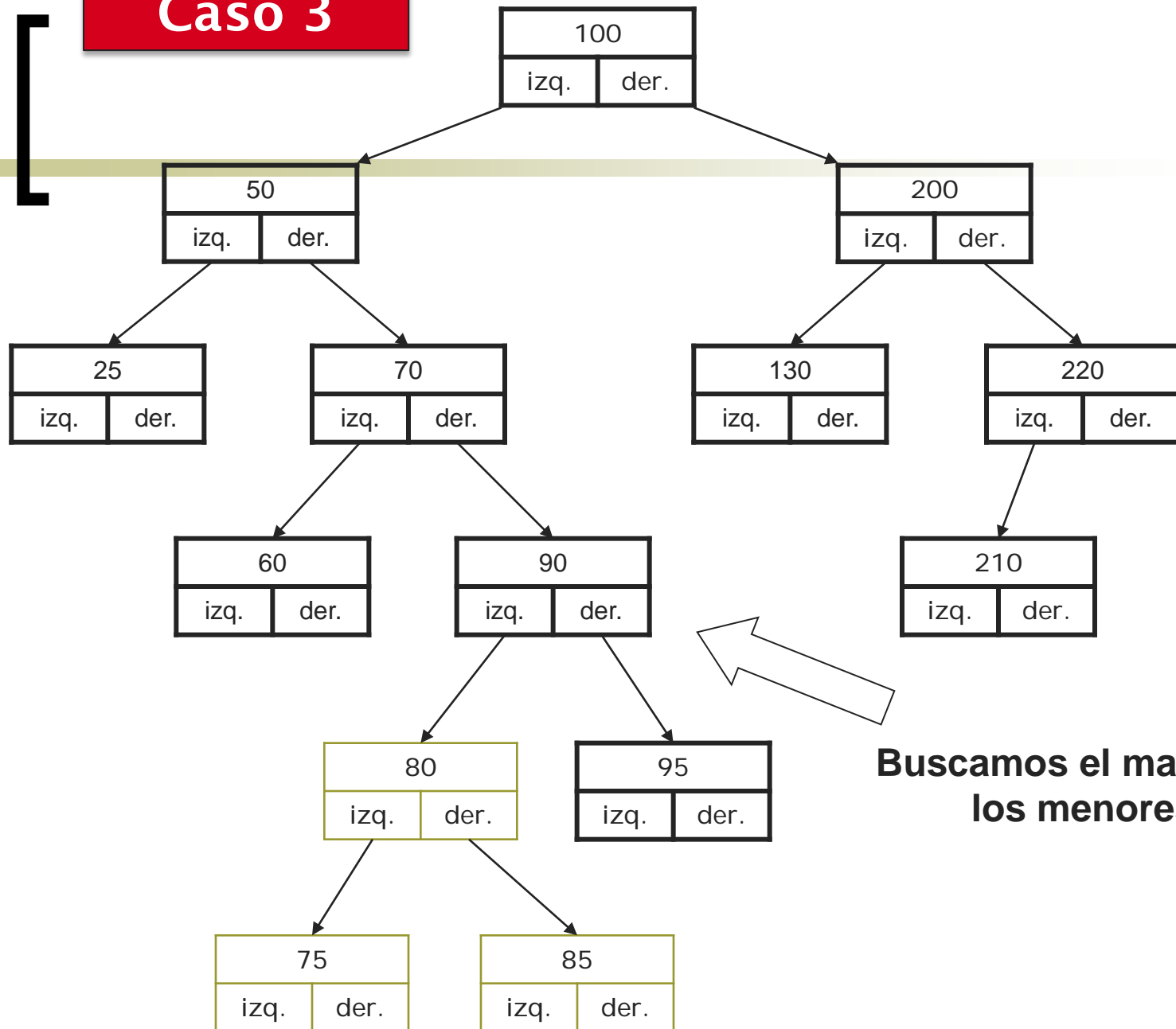


Caso 3



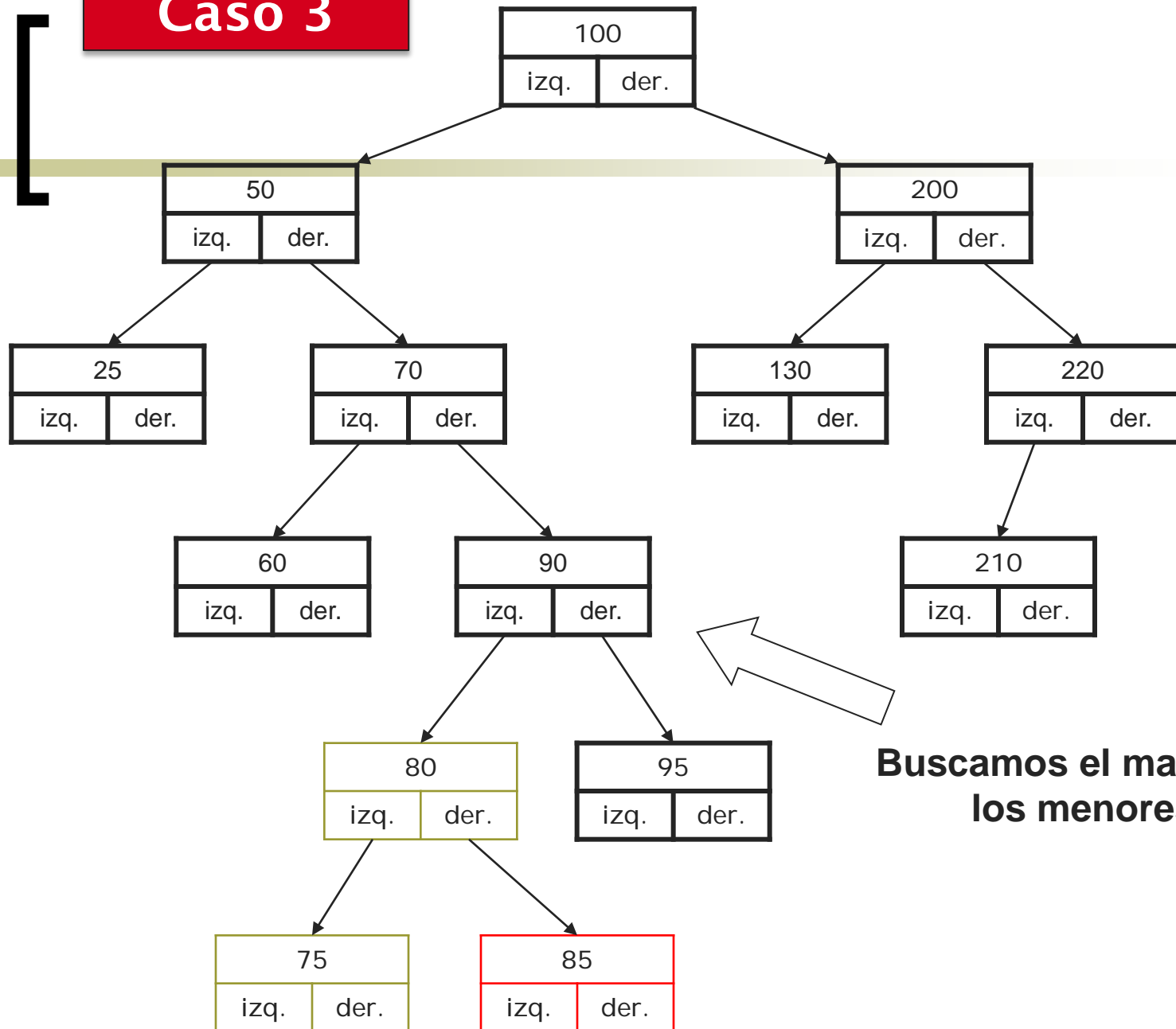


Caso 3



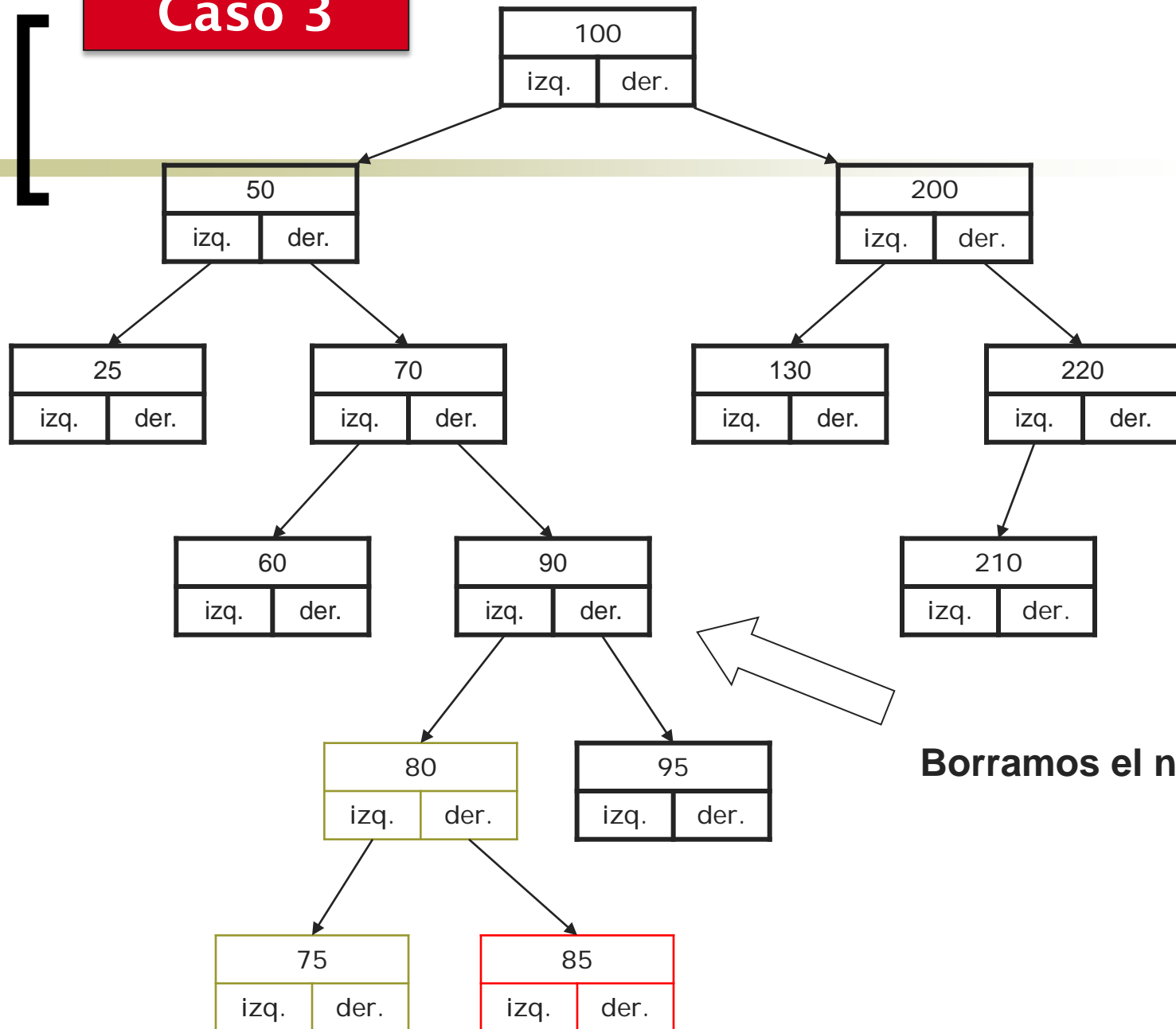


Caso 3



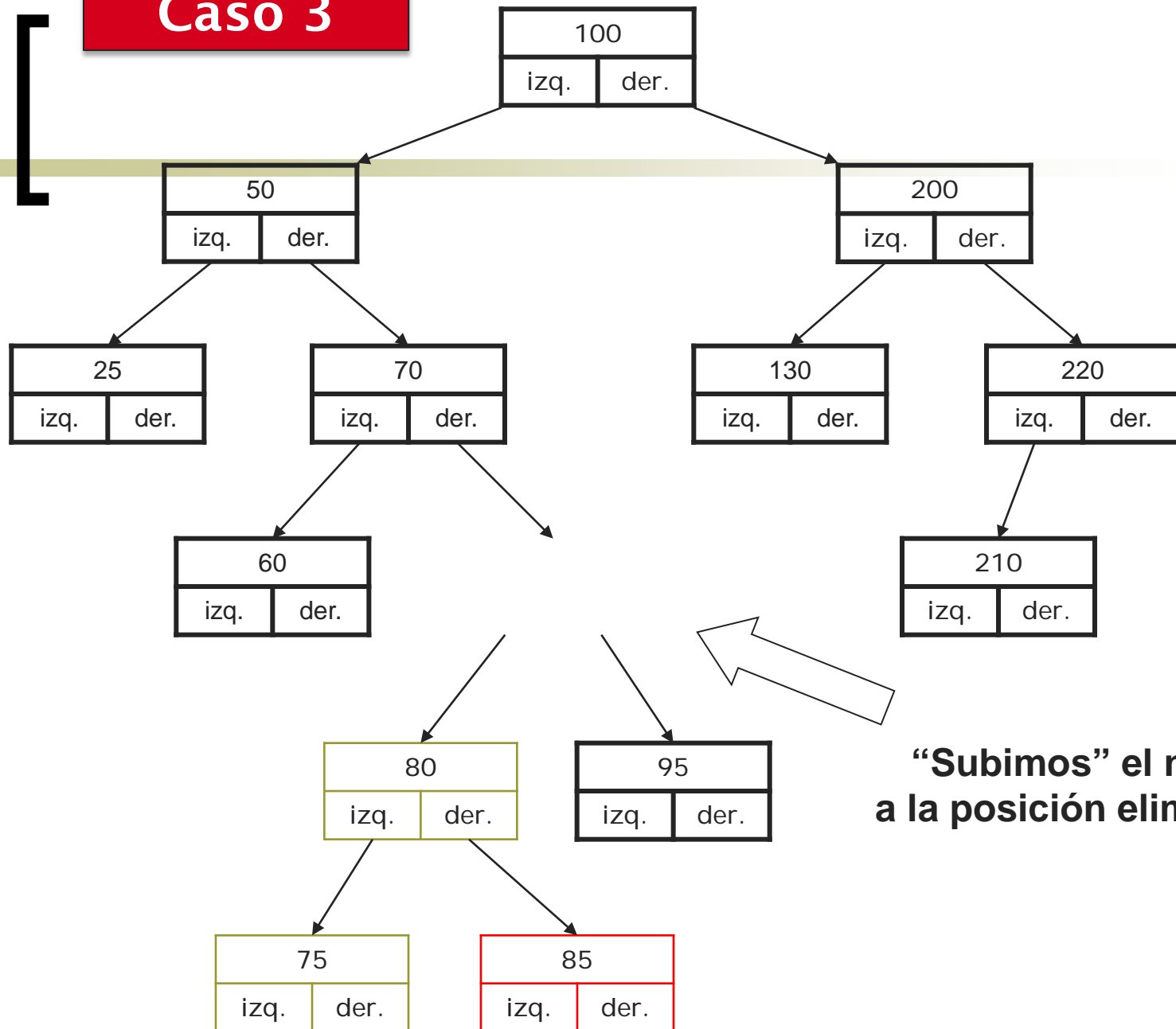


Caso 3



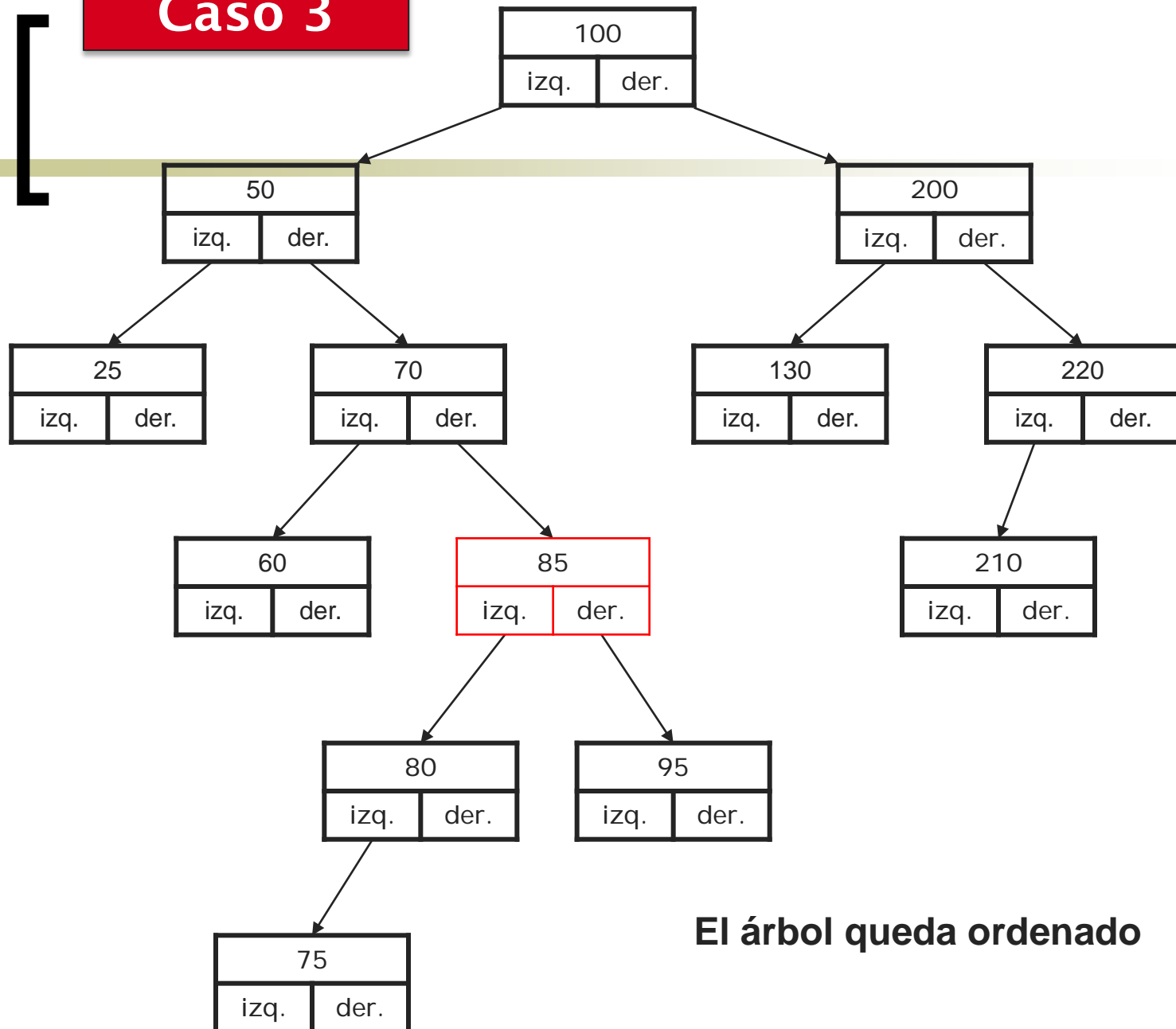


Caso 3





Caso 3



El árbol queda ordenado

```

P_NODO_ARBOL eliminar (P_NODO_ARBOL a, int i){
    P_NODO_LISTA q,p;
    if (a == NULL) return a;
    if (a->valor > i) a-> izq = eliminar (a->izq, i);
    else if (a->valor < i) a-> der = eliminar (a->der, i);
    else{
        if ((a->der == NULL)&&(a->izq == NULL)) //caso 1
            return borraNode(a);
        //Caso 2
        if (a->der == NULL){
            q = a->izq;
            borraNode (a);
            return q;
        }
        if (a->izq == NULL){
            q = a->der;
            borraNode (a);
            return q;
        }
        //Caso 3
        for(p=a,q=a->izq; q->der != NULL; q = q->der)
            p=q;
        if (p!=a)
            p-> der = q->izq;
        else
            p->izq =q->izq;
        a->valor =q->valor;
        borrarNode(q);
    }
    return a;
}

```

Bibliografía

- King, K.N. **C Programming. A modern approach.** 2ªed. Ed. W.W. Norton & Company. Inc. 2008.
- Khamtane Ashok. **Programming in C.** Ed. Pearson. 2012.
- Ferraris Llanos, R. D. **Fundamentos de la Informática y Programación en C.** Ed. Paraninfo. 2010.