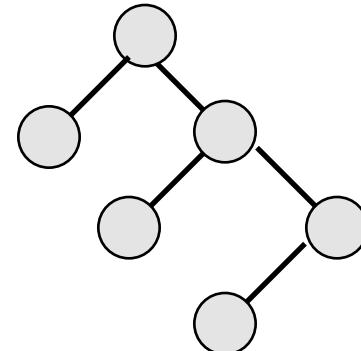
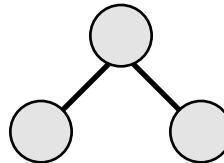
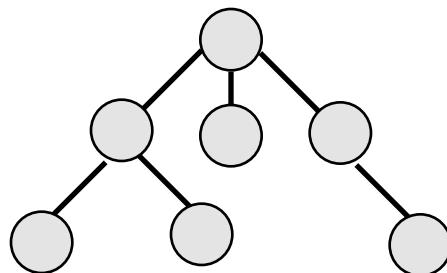


Árboles Binarios de Búsqueda

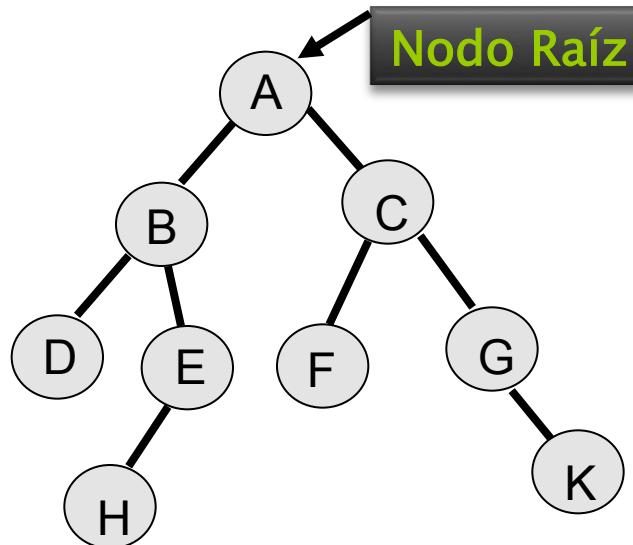
¿Qué es un Árbol?

- ▶ Es una estructura de datos jerárquica.
- ▶ La relación entre los elementos es de uno a muchos.



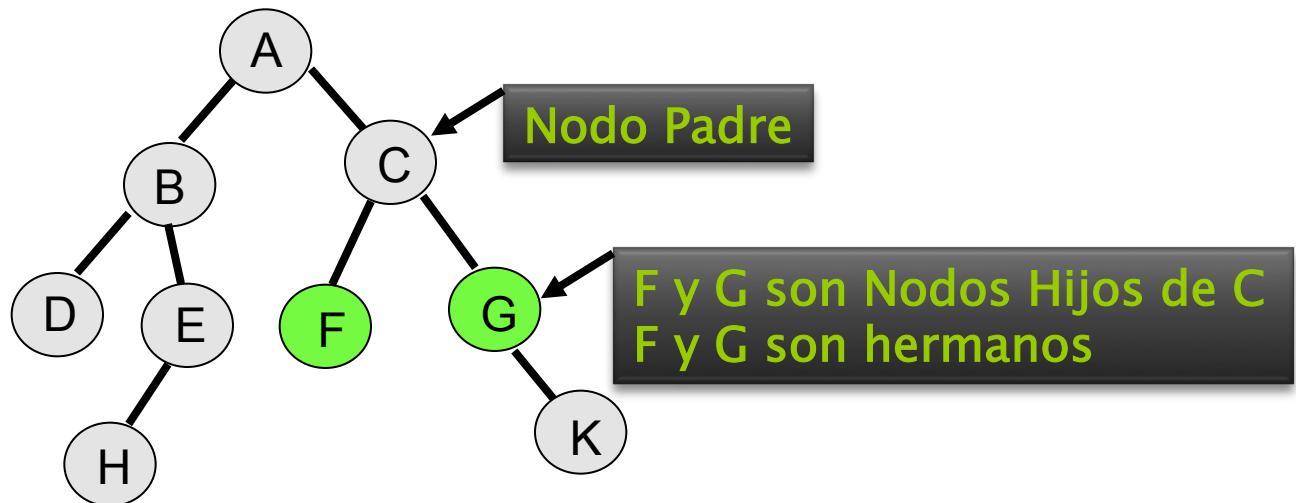
Terminología

- ▶ Nodo: Cada elemento en un árbol.
- ▶ Nodo Raíz: Primer elemento agregado al árbol.



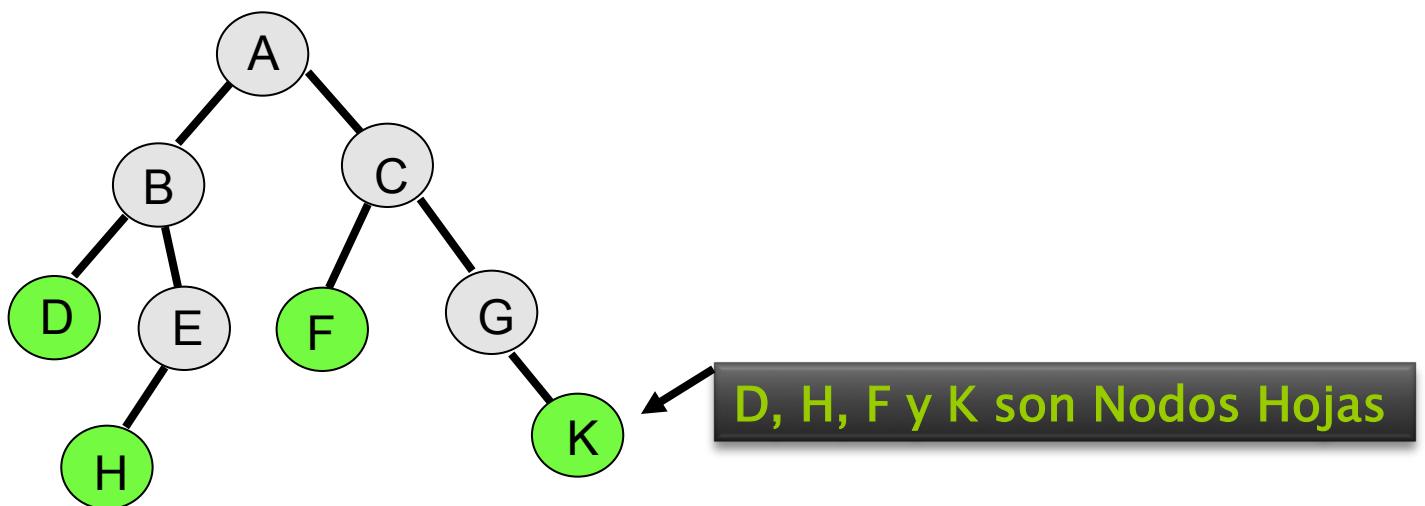
Más terminología

- ▶ Nodo Padre: Se le llama así al nodo predecesor de un elemento.
- ▶ Nodo Hijo: Es el nodo sucesor de un elemento.
- ▶ Hermanos: Nodos que tienen el mismo nodo padre.



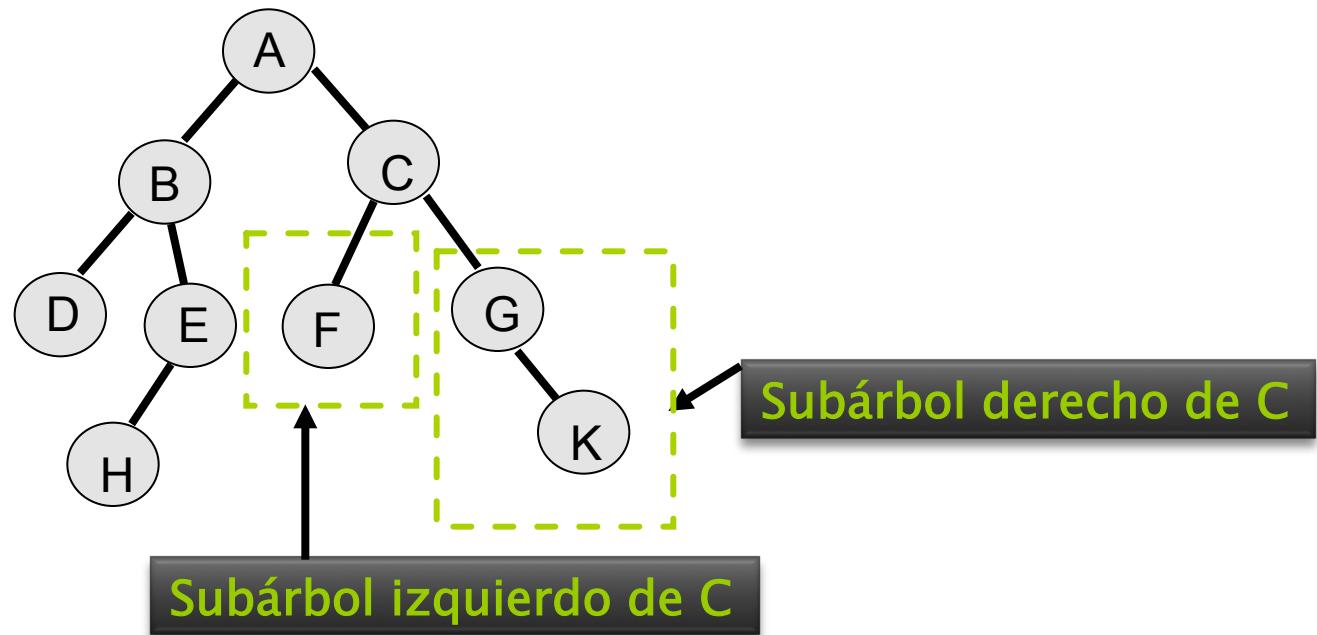
Más terminología

- ▶ Nodo Hoja: Aquel nodo que no tiene hijos.

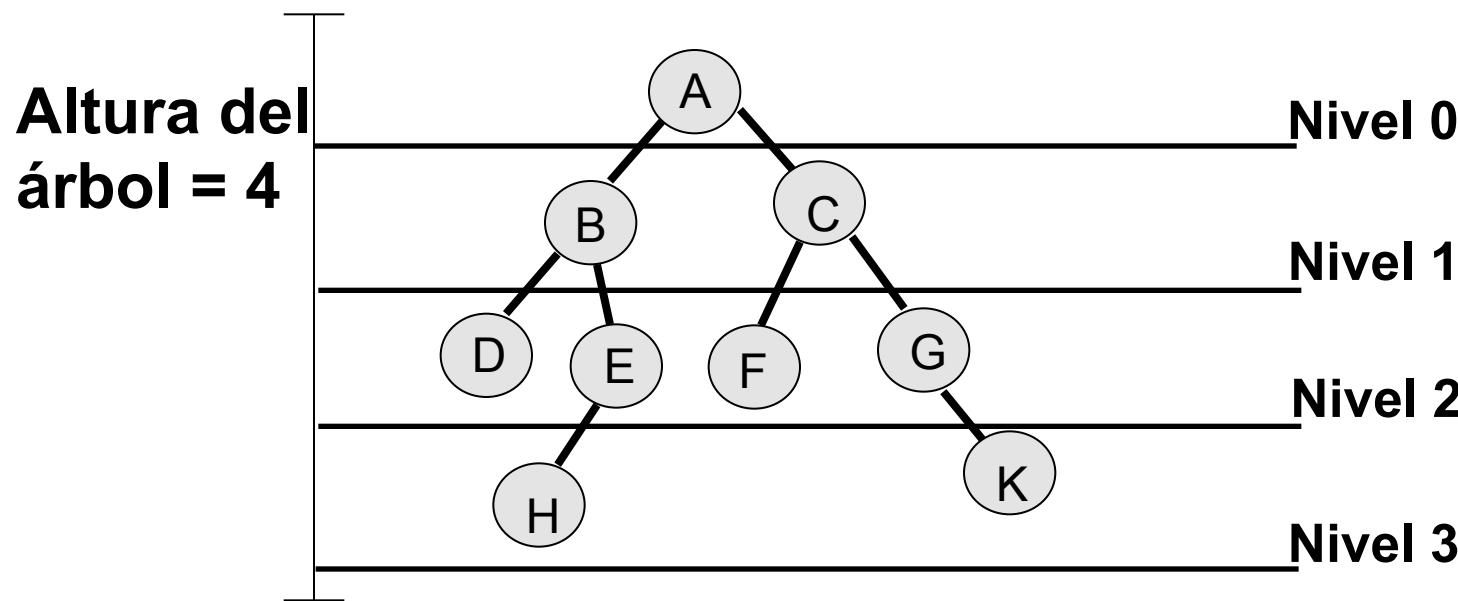


Más terminología

- ▶ **Subárbol**: Todos los nodos descendientes por la izquierda o derecha de un nodo.



Altura y Niveles

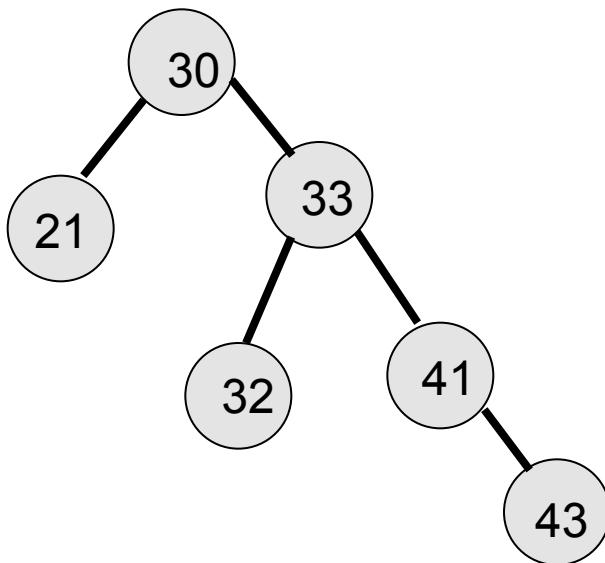
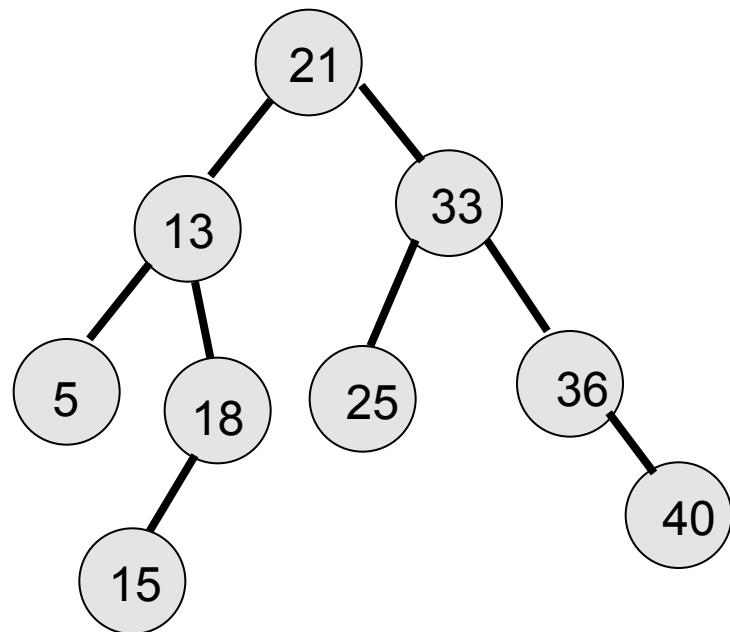


La Altura es la cantidad de niveles.

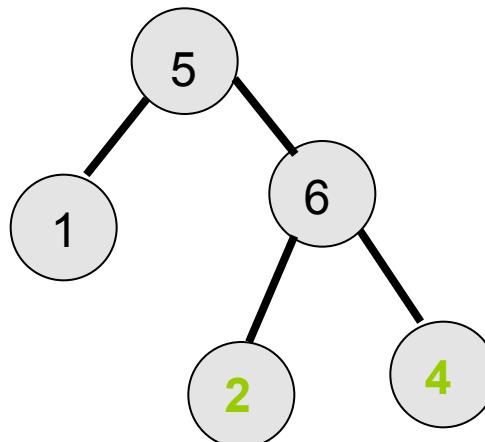
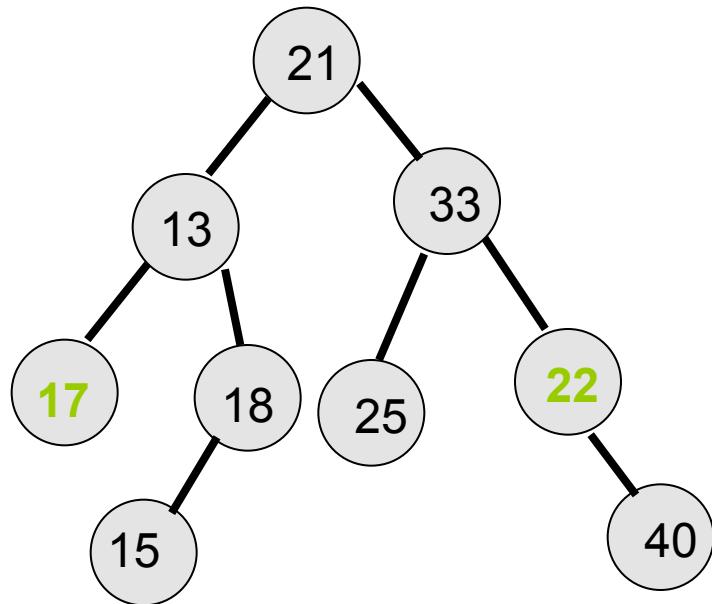
Árbol Binario de Búsqueda (ABB)

- ▶ Este tipo de árbol permite almacenar información ordenada.
- ▶ Reglas a cumplir:
 - Cada nodo del árbol puede tener 0, 1 ó 2 hijos.
 - Los descendientes **izquierdos** deben tener un valor **menor al padre**.
 - Los descendientes **derechos** deben tener un valor **mayor al padre**.

Ejemplos de ABB...



¿Por qué no son ABB?



Clase Nodo

```
class NodoArbol
{
public:
    int iInfo;
    NodoArbol *plzq, *pDer;
    NodoArbol( ){ plzq = pDer = NULL; }
    NodoArbol(int iDato)
        { iInfo = iDato; plzq = pDer = NULL; }
};
```

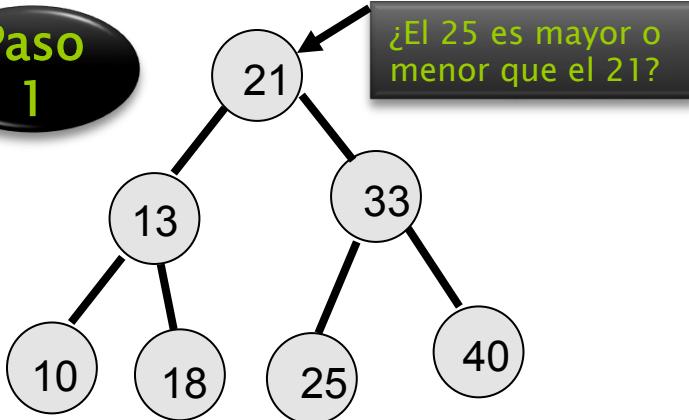
Clase Árbol Binario de Búsqueda

```
class ABB
{
    private:
        NodoArbol *pRaiz;
    public:
        //Constructor
        ABB() { pRaiz = NULL; }
        //Destructor
        ~ABB( ) { }
        //Otros métodos de la clase
};
```

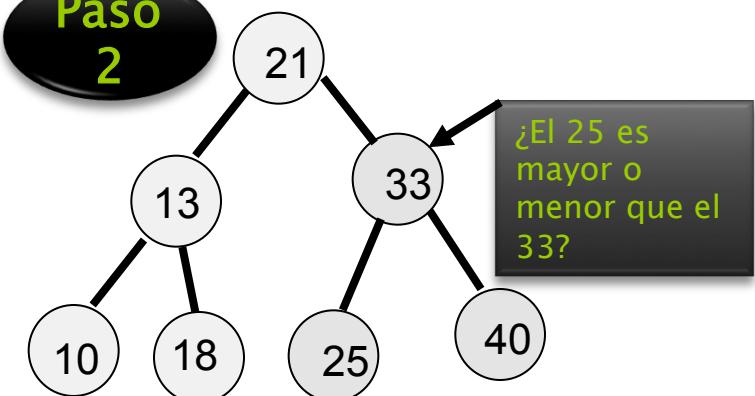
Proceso para buscar un nodo...

Buscar el 25

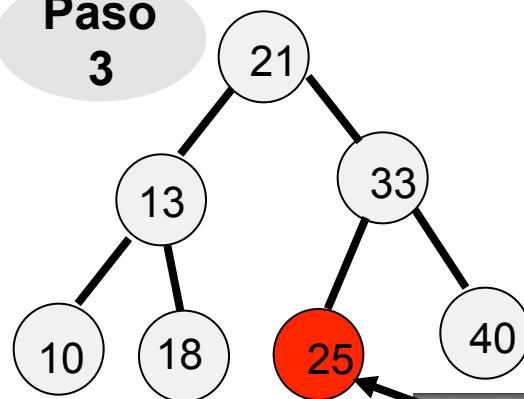
Paso
1



Paso
2



Paso
3



Encontrado

Implementación de la búsqueda

```
bool busca (int iValor)
{
    NodoArbol * pP = pRaiz;
    while (pP != NULL)
    {
        if (pP->iInfo == iValor)
            return true;
        else
            pP=(pP->iInfo > iValor? pP->pIzq: pP->pDer);
    }
    return false;
}
```

pP contiene la dirección del nodo que tiene el valor buscado

Equivalente a:

```
if ( pP -> iInfo > iValor )
    pP = pP -> pIzq;
else pP = pP -> pDer;
```

No se encontró el valor por lo que se regresa false

Proceso para agregar nodos...

Reglas:

- El valor a insertar no existe en el árbol.
- El nuevo nodo será un Nodo Hoja del árbol.

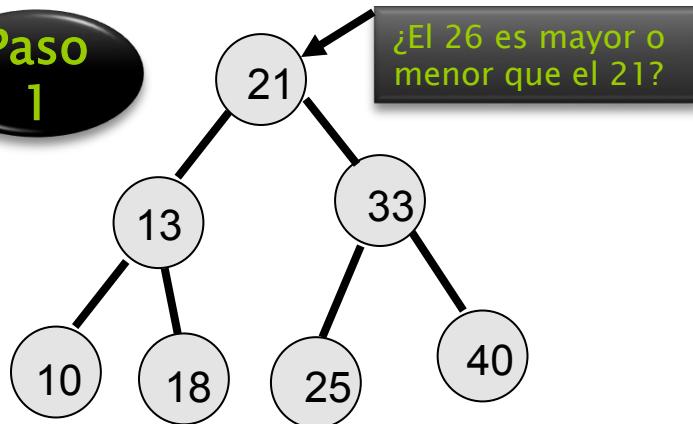
Procedimiento

1. Buscar el Nodo Padre del nodo a agregar.
2. Agregar el nodo.

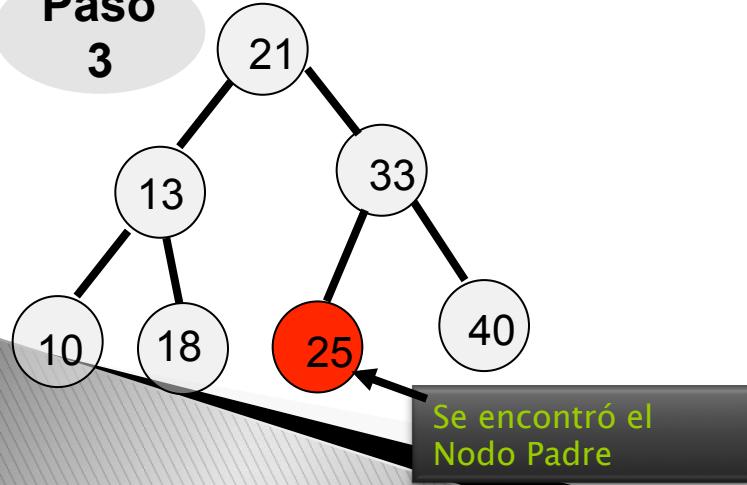
Ejemplo

Agregar el valor 26

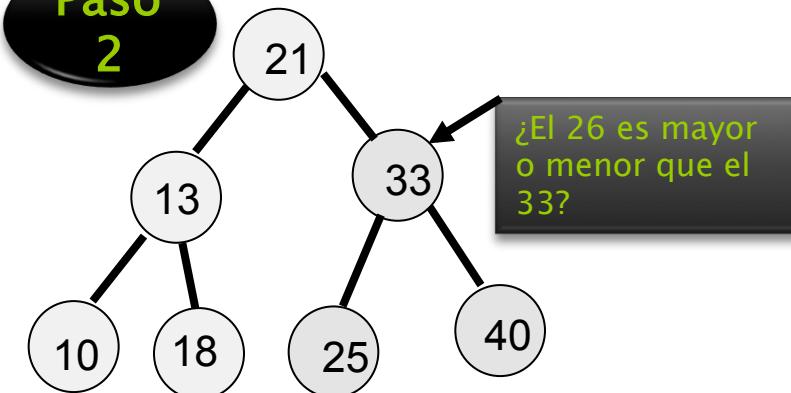
Paso
1



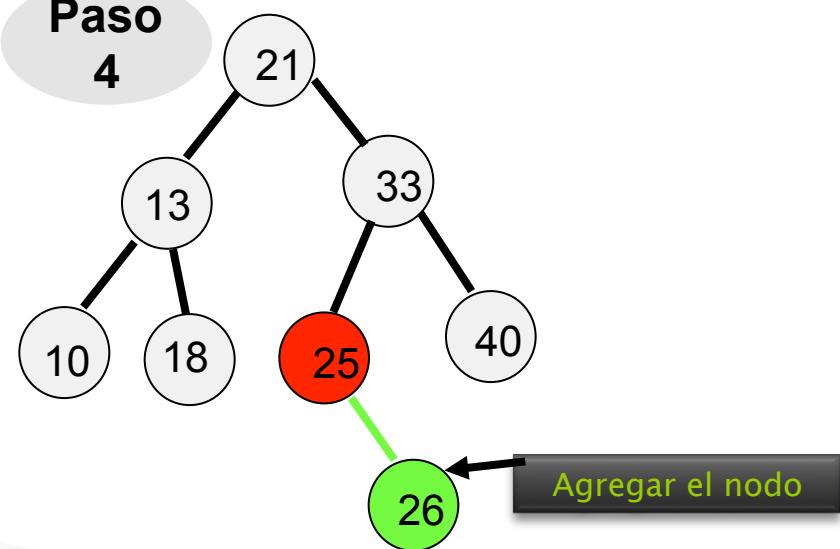
Paso
3



Paso
2

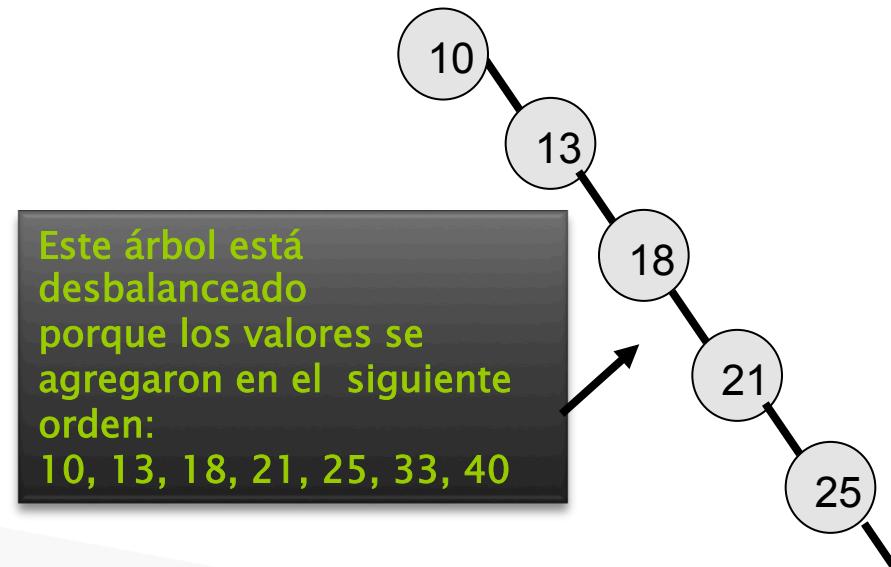


Paso
4



Comentarios importantes....

- ▶ El orden de inserción de los datos, determina la forma del ABB.
- ▶ ¿Qué pasará si se insertan los datos en forma ordenada?
- ▶ La forma del ABB determina la eficiencia del proceso de búsqueda.
- ▶ Entre menos altura tenga el ABB, más balanceado estará, y más eficiente será.



Agregar un valor al árbol

```
NodoArbol* encuentraPadre (int iValor)
```

```
{  
    NodoArbol *pPadre, *pHijo;  
    pHijo = pRaiz;  
    pPadre = NULL;
```

```
    while (pHijo != NULL && pHijo->iInfo != iValor)
```

```
{  
    pPadre = pHijo;  
    pHijo = (pHijo->iInfo > iValor ? pHijo->pIzq : pHijo->pDer);  
}
```

```
    return pPadre;  
}
```

```
void meterABB (int iValor)
```

```
{  
    NodoArbol *pPadre = encuentraPadre(iValor);  
    NodoArbol *pNuevo = new NodoArbol (iValor);  
    if( pPadre == NULL) //Agrega el primer nodo del árbol  
        pRaiz = pNuevo;  
    else  
    { //Agrega un nodo hoja  
        if ( pPadre->iInfo > iValor )  
            pPadre->pIzq = pNuevo;  
        else  
            pPadre->pDer = pNuevo;  
    }  
}
```

Proceso para eliminar un nodo

- ▶ Si el nodo a eliminar es un:

- **Nodo hoja**

- Buscar el Nodo Padre del nodo a borrar.
 - Desconectarlo.
 - Liberar el nodo.

- **Nodo con un hijo**

- Buscar el Nodo Padre del nodo a borrar.
 - Conectar el hijo con el padre del nodo a borrar.
 - Liberar el nodo.

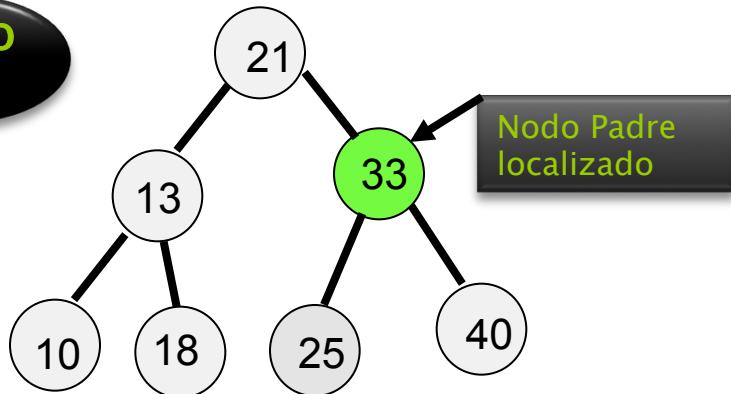
- **Nodo con dos hijos**

- Localizar el nodo predecesor o sucesor del nodo a borrar.
 - Copiar la información.
 - Eliminar el predecesor o sucesor según sea el caso.

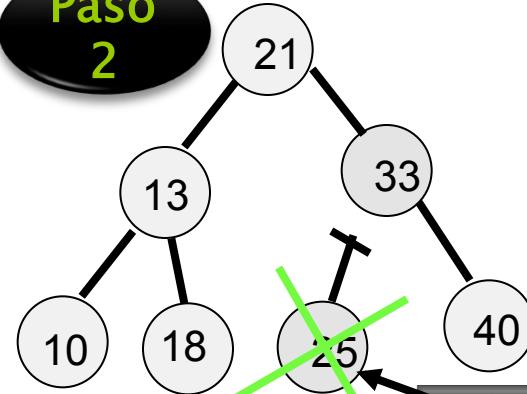
Caso: Eliminar Nodo hoja

Eliminar el valor 25

Paso
1



Paso
2

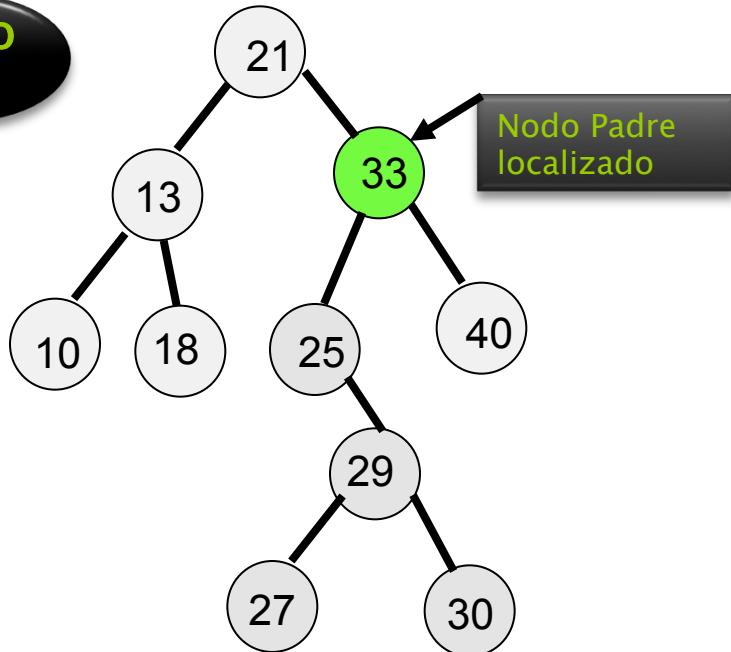


Desconectarlo y
liberar el nodo

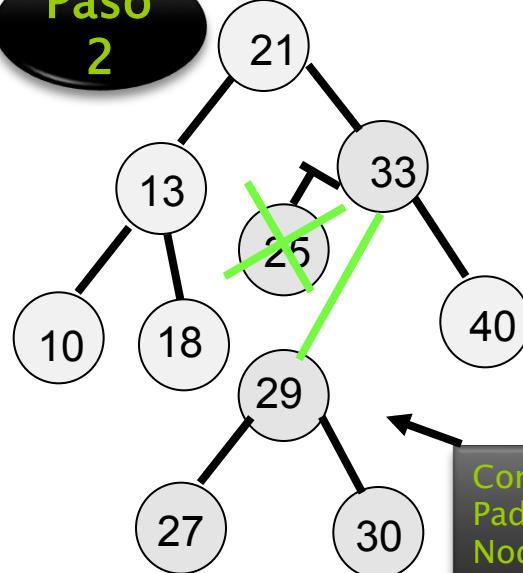
Caso: Eliminar Nodo con un hijo

Eliminar el valor 25

Paso
1



Paso
2

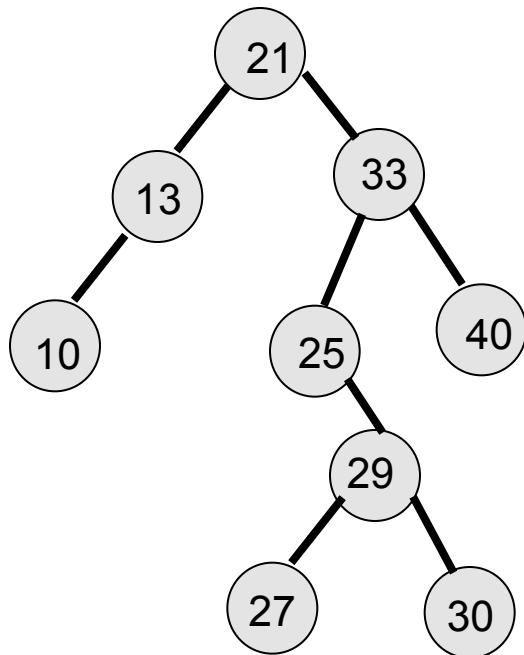


Caso: Eliminar nodo con dos hijos

1. Localizar el nodo predecesor o sucesor del nodo a borrar.
 - El PREDECESOR es “el Mayor de los Menores”.
 - El SUCESOR es “el Menor de los Mayores”.
 - Para la implementación es igual de eficiente programar la búsqueda del predecesor que del sucesor.
2. El valor del predecedor (o sucesor) se copia al nodo a borrar.
3. Eliminar el nodo del predecesor (o sucesor según sea el caso).

Predecesor

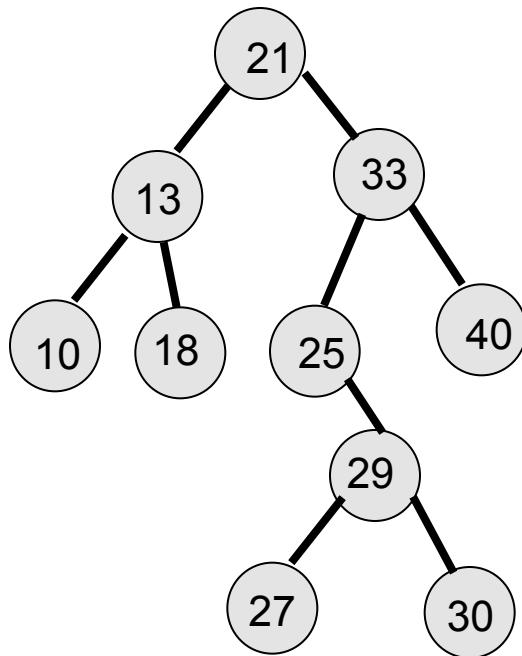
Uno a la IZQUIERDA y todo a la DERECHA



| El predecesor de: | Es: |
|-------------------|-----|
| 33 | 30 |
| 21 | 13 |
| 29 | 27 |

Sucesor

Uno a la DERECHA y todo a la IZQUIERDA



| El sucesor de: | Es: |
|----------------|-----|
| 21 | 25 |
| 33 | 40 |
| 29 | 30 |

Implementación del....

PREDECESOR

pActual apunta al nodo a borrar

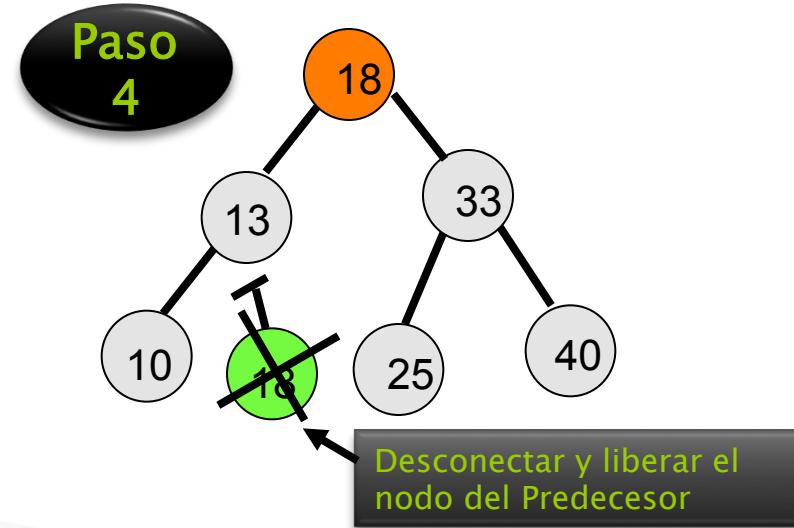
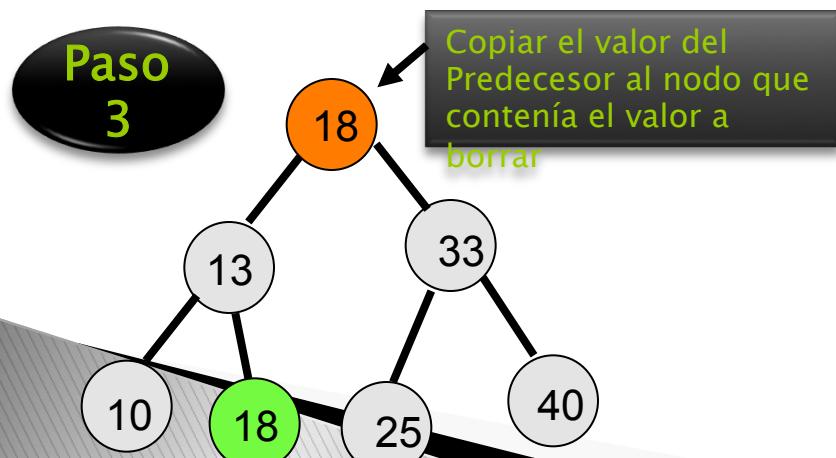
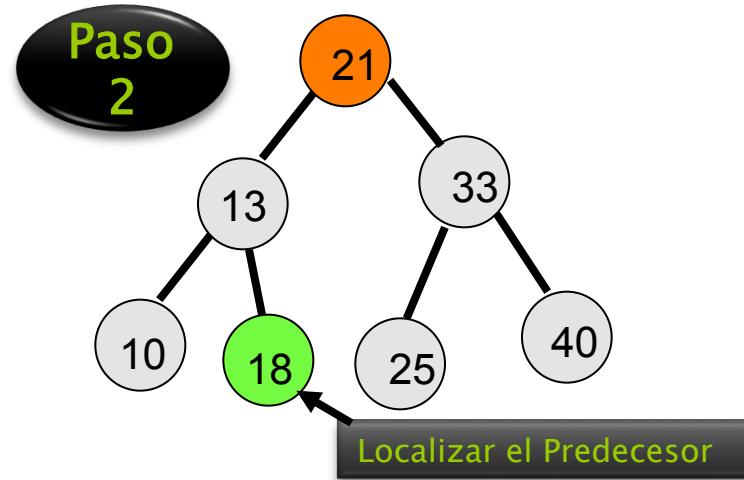
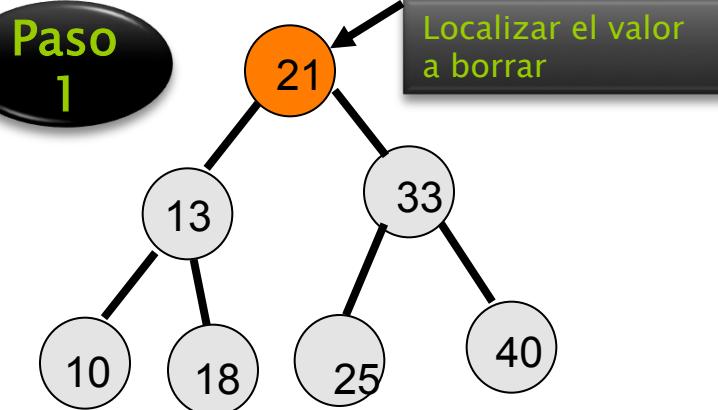
```
NodoArbol* predecesor (NodoArbol *pActual)
{
    NodoArbol *pP = pActual->pIzq;
    while (pP->pDer != NULL )
        pP = pP->pDer;
    return pP;
}
```

SUCESOR

```
NodoArbol* sucesor (NodoArbol *pActual)
{
    NodoArbol *pP = pActual->pDer;
    while (pP->pIzq != NULL )
        pP = pP->pIzq;
    return pP;
}
```

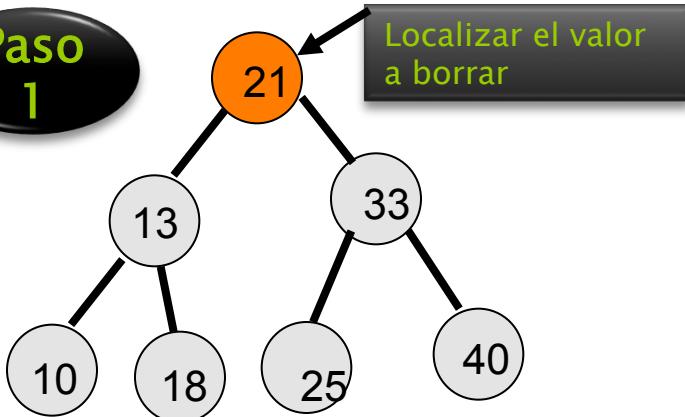
Caso: Eliminar Nodo con dos hijos

Eliminar el valor 21
utilizando el **predecesor**

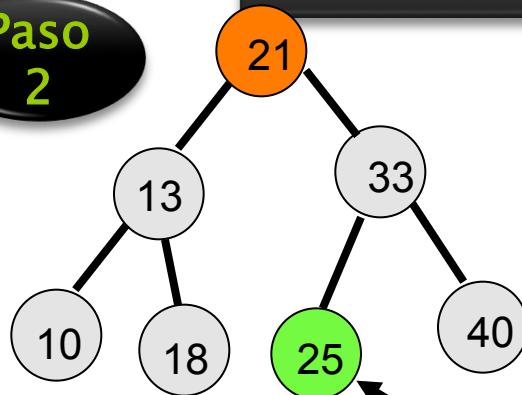


Caso: Eliminar Nodo con dos hijos

Paso 1

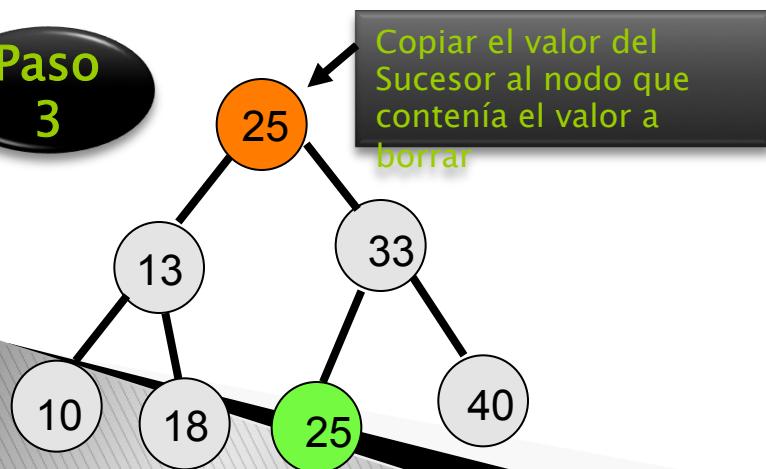


Paso 2

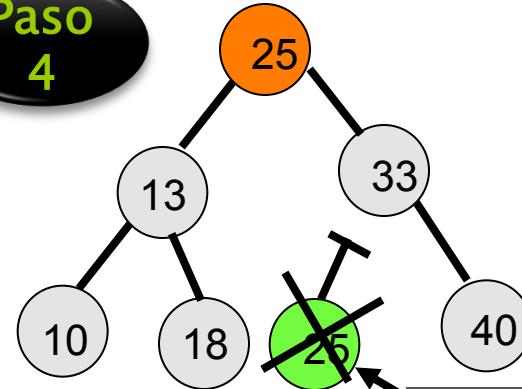


Eliminar el valor 21
utilizando el Sucesor

Paso 3



Paso 4



Desconectar y liberar el
nodo del Sucesor

Elimina un valor del árbol

```
void sacarABB(int iValor) {
    NodoArbol *pPadre = encuentraPadre(iValor);
    NodoArbol *pNodoABorrar;
    if (pPadre == NULL) pNodoABorrar = pRaiz;
    else
        pNodoABorrar=(pPadre->iInfo > iValor? pPadre->pIzq : pPadre->pDer);

    //Encontrar substituto cuando el nodo a borrar tiene 2 hijos
    if (pNodoABorrar->pIzq != NULL && pNodoABorrar->pDer != NULL)
    {
        NodoArbol *pSubstituto = predecesor(pNodoABorrar);
        int iNuevovalor = pSubstituto->iInfo;
        sacarABB (pSubstituto->iInfo);
        pNodoABorrar->iInfo = iNuevovalor;
    }
    else if (pPadre == NULL)
    { //Borra nodo raíz el cual solo tiene un hijo
        if (pNodoABorrar->pDer == NULL)
            pRaiz = pNodoABorrar->pIzq;
        else pRaiz = pNodoABorrar->pDer;
    }
    else if (pPadre->iInfo > iValor)      //Borra nodo con 0 o 1 hijo
        if (pNodoABorrar->pIzq == NULL)
            pPadre->pIzq = pNodoABorrar->pDer;
        else pPadre->pIzq = pNodoABorrar->pIzq;
    else if (pNodoABorrar->pDer == NULL)
        pPadre->pDer = pNodoABorrar->pIzq;
        else pPadre->pDer = pNodoABorrar->pDer;
    }
}
```