



UNIVERSIDAD DE LOS LAGOS

ÁRBOL BINARIO

ANALISIS Y DISEÑO DE ALGORITMOS

JUAN JOSÉ RAMÍREZ LAMA
JUAN.RAMIREZ@ULAGOS.CL

DEPARTAMENTO DE CIENCIAS DE LA INGENIERÍA
INGENIERÍA CIVIL EN INFORMÁTICA

Campus Osorno
Av. Fuchelocer 1305
Teléfono +56 64 2333 000
Fax +56 64 2333 774
Osorno, Chile

Campus Puerto Montt
Camino a Chonchihue Km 6
Teléfono +56 65 2322 536
Puerto Montt, Chile

Sede Santiago
República 517
Barrio Universitario
Teléfono +56 02 2675 3057
Santiago, Chile

Sede Chiloé
Ubaldo Mansilla Barrientos 131
Teléfono 56 65 2322 409
Castro, Chile
Eleuterio Ramírez 348
Teléfono +56 65 2322 476
Ancud, Chile

5 UNIVERSIDAD ACREDITADA
Comisión Nacional de Acreditación
Sistema de Evaluación, Desarrollo e Innovación
SISTEMA AVANZADA

www.ulagos.cl

TABLA DE CONTENIDO

1 Árboles Binarios

2 Construcción

3 Árbol Binario de Búsqueda

- Recorridos
- Inserción en un ABB
- Eliminar Nodos

4 Implementación de un Árbol Binario

SECCIÓN SIGUIENTE

1 Árboles Binarios

2 Construcción

3 Árbol Binario de Búsqueda

- Recorridos
- Inserción en un ABB
- Eliminar Nodos

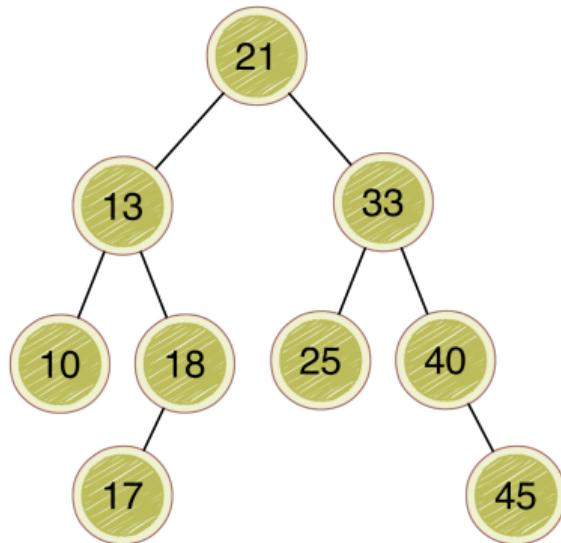
4 Implementación de un Árbol Binario

ÁRBOLES BÍNARIOS

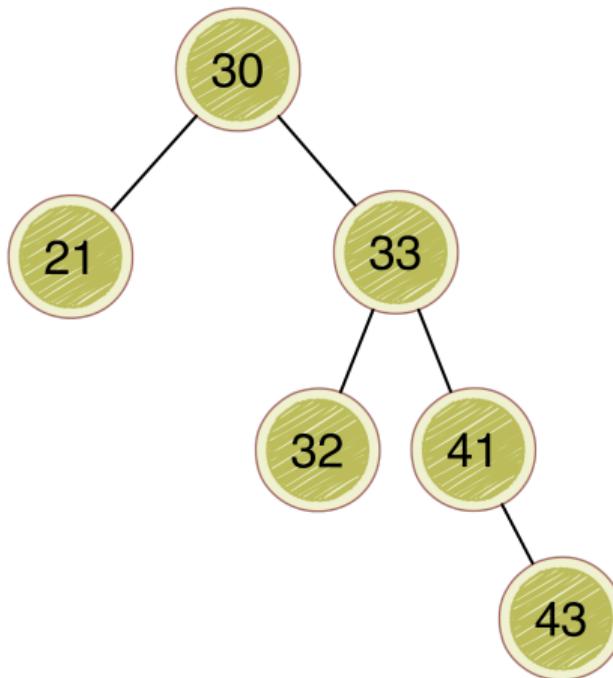
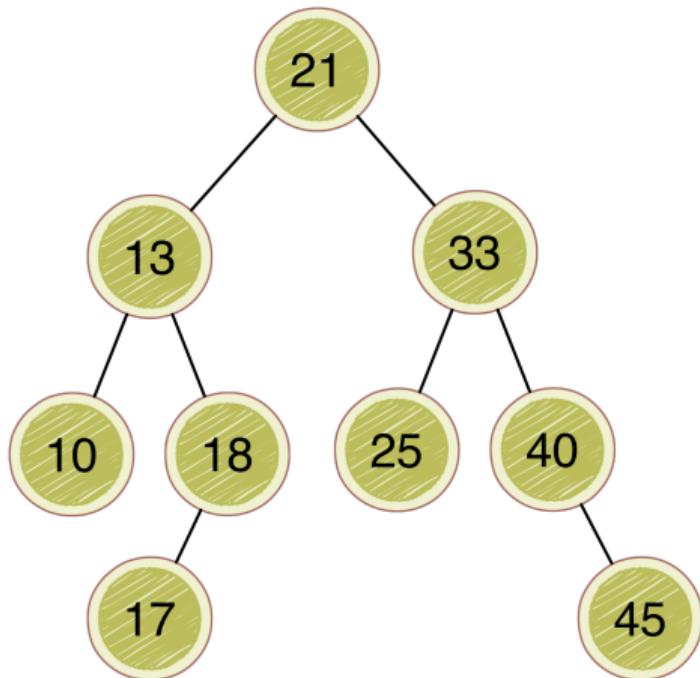
- ▶ Este tipo de árbol permite almacenar información ordenada.
 - ▶ Son el soporte para los árboles binarios de búsqueda (árboles binarios ordenados).
- ▶ Reglas a cumplir:
 - ▶ Cada nodo del árbol puede tener 0, 1 ó 2 hijos.
 - ▶ Los descendientes izquierdos deben tener un valor menor al padre.
 - ▶ Los descendientes derechos deben tener un valor mayor al padre.
- ▶ Un uso de los árboles binarios lo constituyen los árboles de expresión, que son la estructura de datos fundamental en el diseño de compiladores.
- ▶ Otro uso de los árboles binarios es el árbol de codificación de Huffman que se usa para implementar un algoritmo simple, pero eficiente de compresión de datos.

DEFINICIÓN DE ÁRBOL BINARIO

- ▶ Conjunto de elementos del mismo tipo donde:
 - ▶ Puede ser un conjunto vacío, o bien **árbol binario vacío**
 - ▶ Puede ser un conjunto no vacío, donde existe un elemento llamado raíz, y el resto de los elementos se distribuyen en **DOS** subconjuntos disjuntos, cada uno de los cuales es un árbol binario, llamados **subárbol izquierdo** y **subárbol derecho**



EJEMPLOS DE ÁRBOLES BÍNARIOS



DEFINICIÓN DE ÁRBOL BINARIO

Completo si cada nodo, salvo las hojas tienen 2 hijos.

Similares tienen la misma forma, pero distinta información.

Equivalentes tienen la misma forma y la misma información.

ESPECIFICACIÓN ALGEBRAICA

- ▶ **Tipo:** Árbol
- ▶ **Tipos Auxiliares:** Elementos, Booleano
- ▶ **Sintaxis:**
 - ▶ ÁrbolVacío: () → Árbol
 - ▶ EsVacío: Árbol → Booleano
 - ▶ Construir: Árbol x Elementos x Árbol → Árbol
 - ▶ Hijolzq: Árbol → Árbol
 - ▶ HijoDcho: Árbol → Árbol
 - ▶ Raíz: Árbol → Elementos

SECCIÓN SIGUIENTE

1 Árboles Binarios

2 Construcción

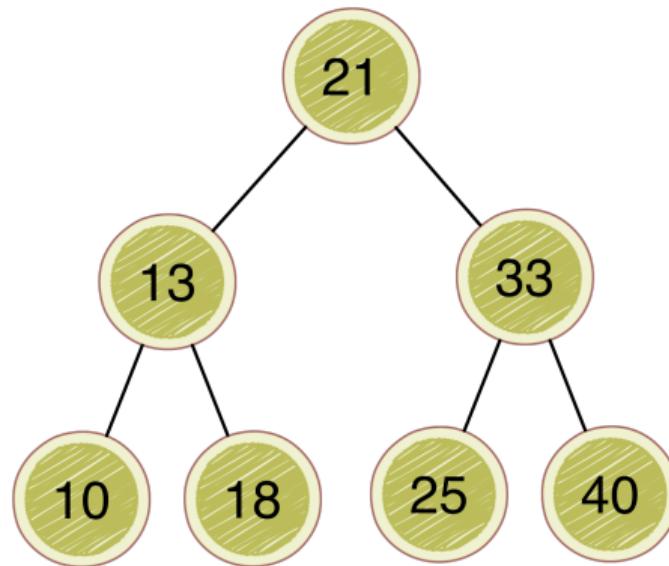
3 Árbol Binario de Búsqueda

- Recorridos
- Inserción en un ABB
- Eliminar Nodos

4 Implementación de un Árbol Binario

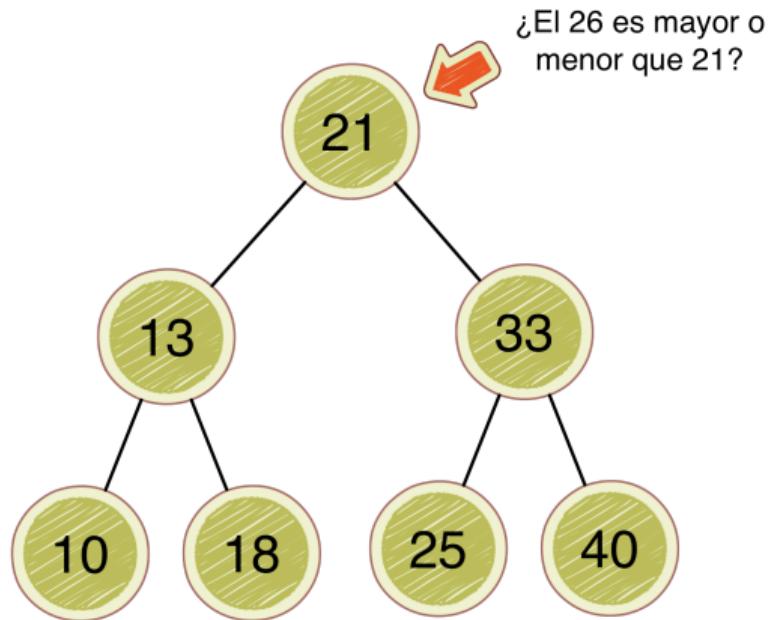
AGREGAR UN ELEMENTO

- Agregar el valor 26



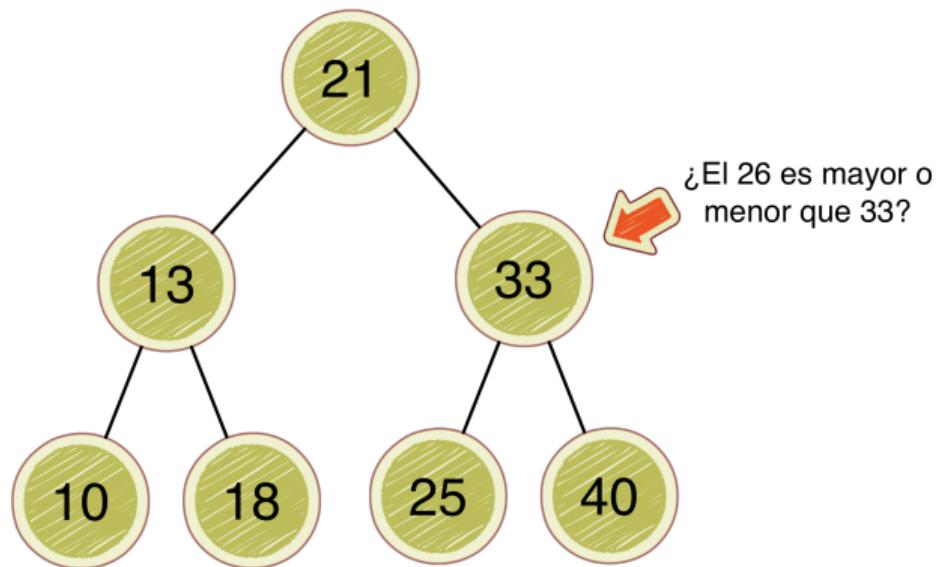
AGREGAR UN ELEMENTO

- Agregar el valor **26**



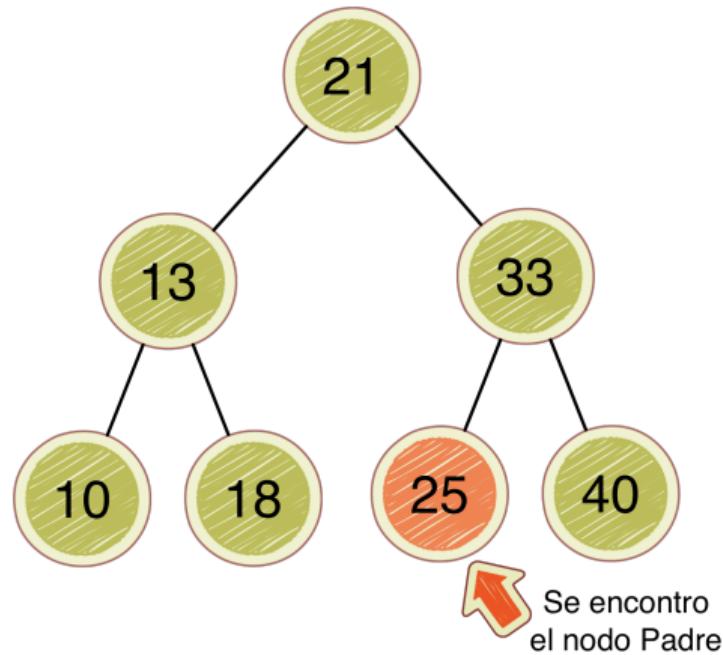
AGREGAR UN ELEMENTO

- Agregar el valor 26



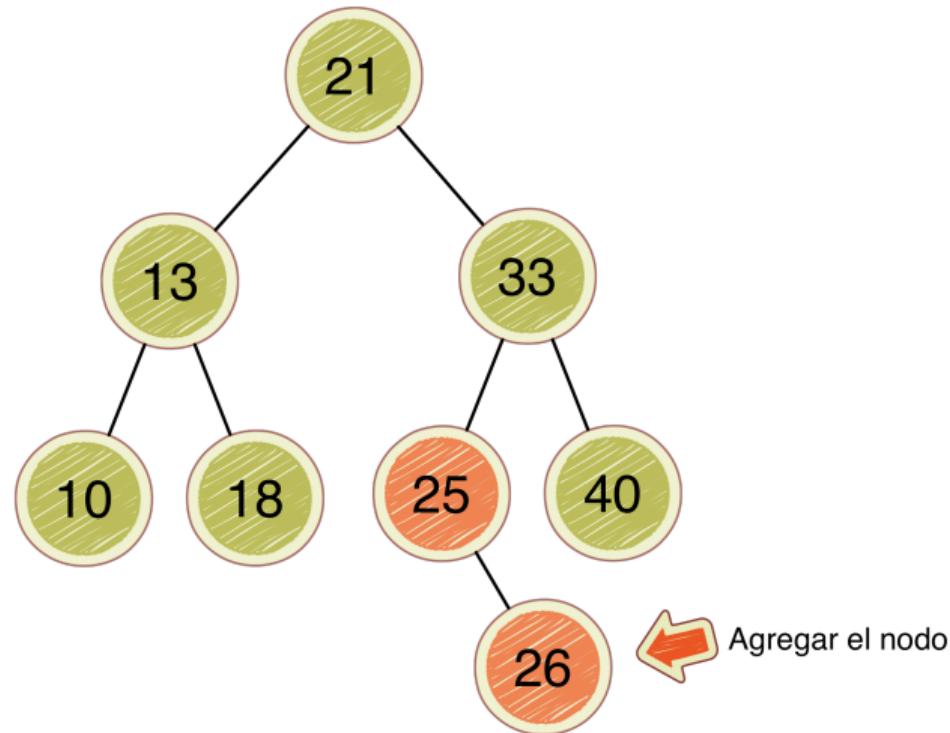
AGREGAR UN ELEMENTO

- Agregar el valor 26



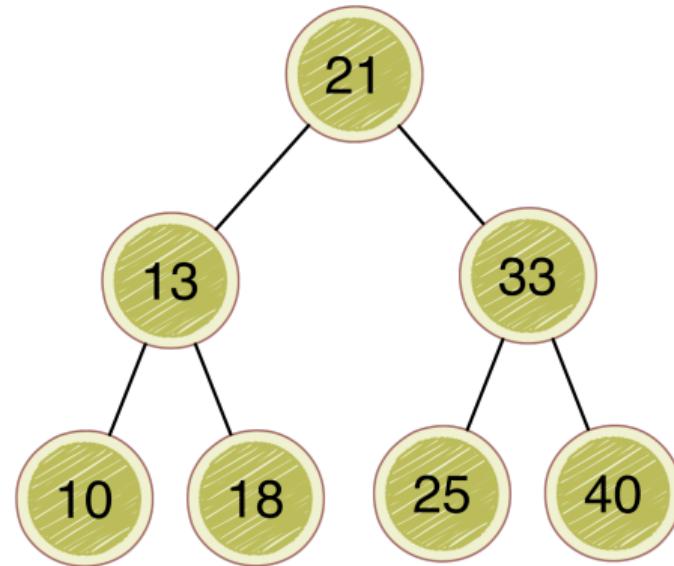
AGREGAR UN ELEMENTO

- Agregar el valor 26



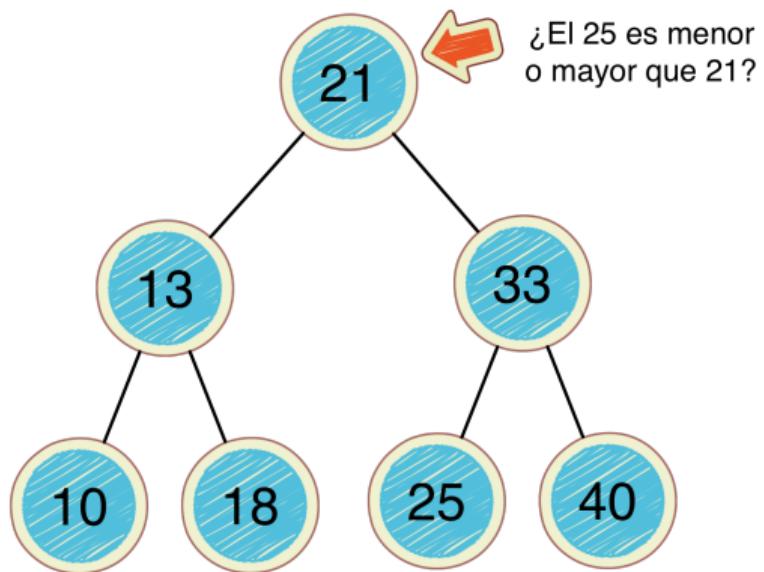
BUSCAR UN ELEMENTO

- ▶ Buscar el valor 25



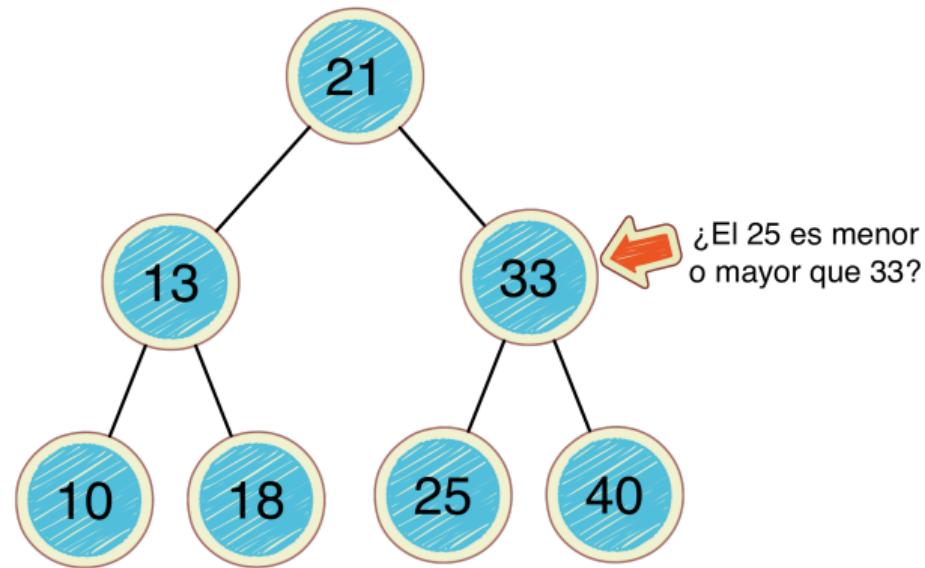
BUSCAR UN ELEMENTO

- ▶ Buscar el valor 25



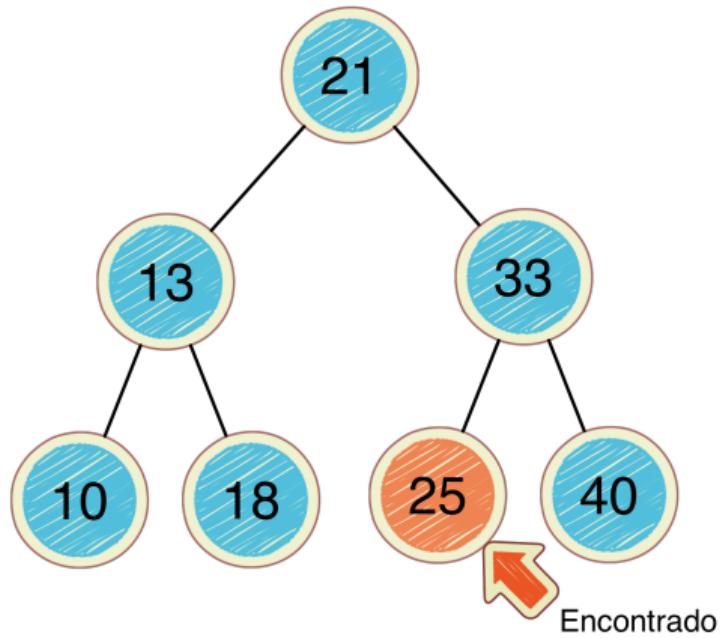
BUSCAR UN ELEMENTO

- ▶ Buscar el valor **25**



BUSCAR UN ELEMENTO

- ▶ Buscar el valor 25



SECCIÓN SIGUIENTE

1 Árboles Binarios

2 Construcción

3 Árbol Binario de Búsqueda

- Recorridos
- Inserción en un ABB
- Eliminar Nodos

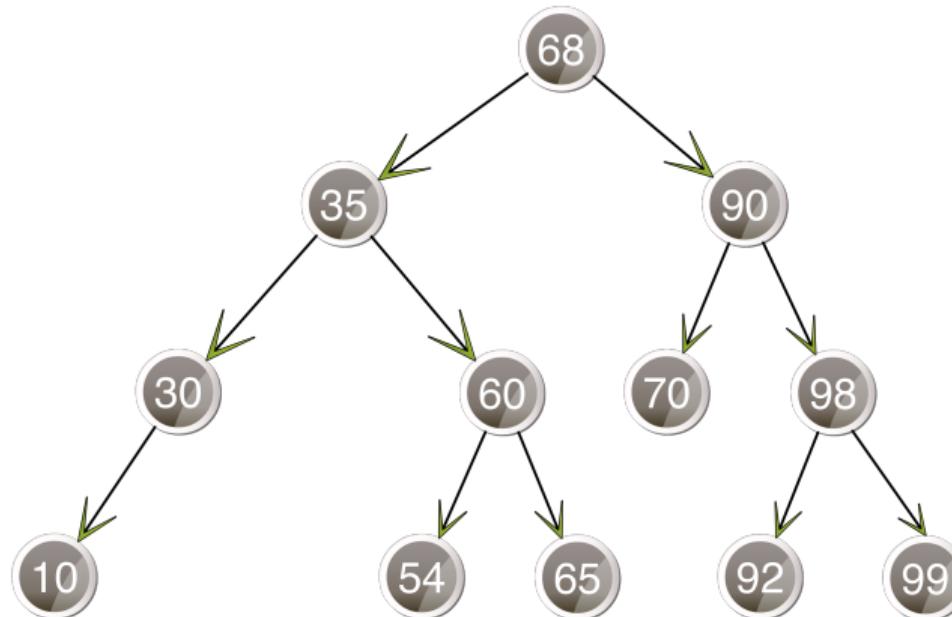
4 Implementación de un Árbol Binario

ÁRBOL BINARIO DE BÚSQUEDA

- ▶ Un árbol es un ABB si éste:
- ▶ Es binario
- ▶ Sus nodos son subárboles de búsqueda binarios y contienen información ordenada de tal que todos los elementos a la izquierda de la raíz son menores a la raíz y todos los elementos a la derecha de la raíz son mayores a la raíz.

CARACTERÍSTICAS DE UN ABB

- ▶ Todos los nodos a la izquierda son menores al padre.
- ▶ Todos los nodos a la derecha son mayores al padre.
- ▶ Y solo pueden tener 2 hijos a lo mucho.



RECORRIDO

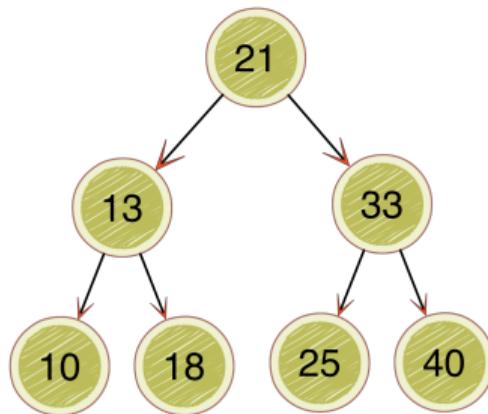
- ▶ Recorrer el árbol es “pasar por” o “visitar” todos los nodos del mismo.
- ▶ Recorridos típicos:
 - ▶ **Preorden** (Raíz-Izquierda-Derecha)
 - ▶ **Inorden** (Izquierda-Raíz-Derecha)
 - ▶ **Postorden** (Izquierda-Derecha-Raíz)
 - ▶ **Anchura (o por niveles)**: procesa los nodos comenzando por la raíz y avanzando de forma descendente y de izquierda a derecha

RECORRIDO EN PREORDEN

▶ Proceso:

- ▶ Visita el nodo raíz del árbol.
- ▶ Recorre el preorden el subárbol izquierdo del nodo raíz.
- ▶ Recorre el preorden el subárbol derecho del nodo raíz.

▶ Aplicación: Generar una réplica del árbol.



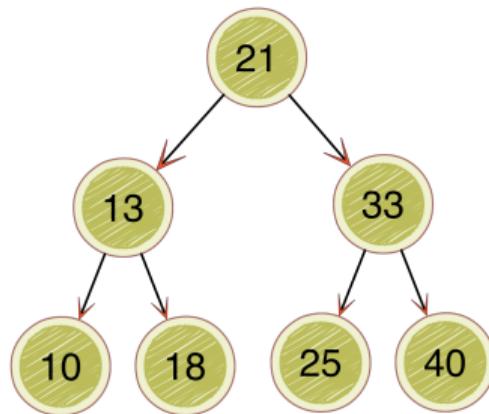
Recorrido en PreOrden: 21-13-10-18-33-25-40

RECORRIDO EN INORDEN

▶ Proceso:

- ▶ Recorre en inorder el subárbol izquierdo.
- ▶ Visita la raíz del árbol.
- ▶ Recorre en inorder el subárbol derecho.

▶ **Aplicación:** Desplegar en orden creciente los elementos del árbol si este es un ABB.



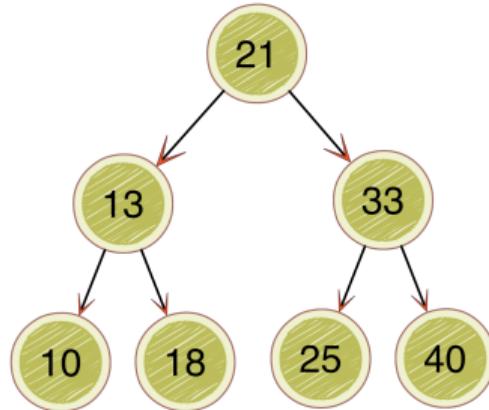
Recorrido en InOrden: 10-13-18-21-25-33-40

RECORRIDO EN POSTORDEN

▶ Proceso:

- ▶ Recorre en postorden el subárbol izquierdo.
- ▶ Recorre en postorden el subárbol derecho.
- ▶ Visita la raíz del árbol.

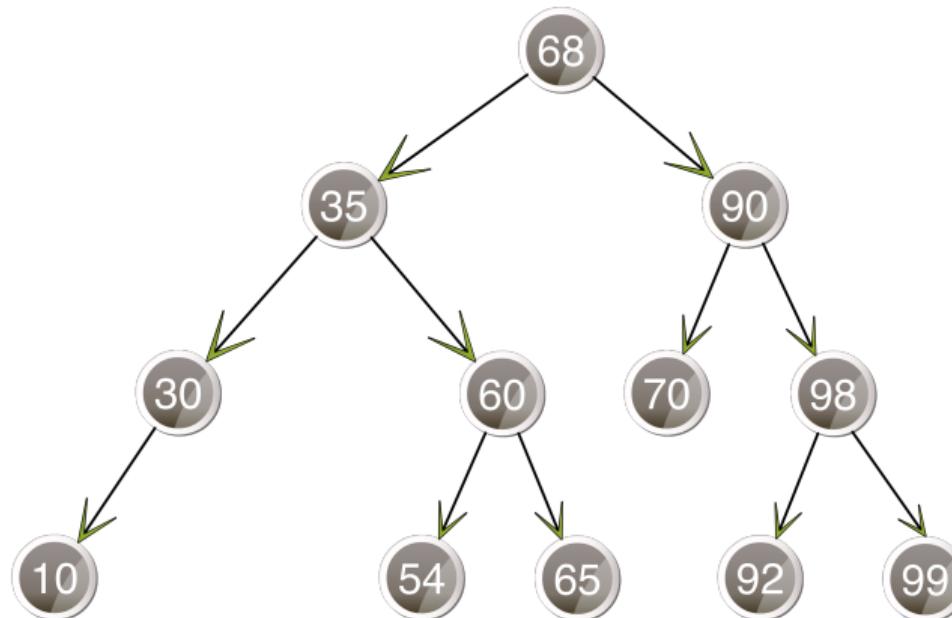
▶ Aplicación: Liberar los nodos de un árbol.



Recorrido en PostOrden: 10-18-13-25-40-33-21

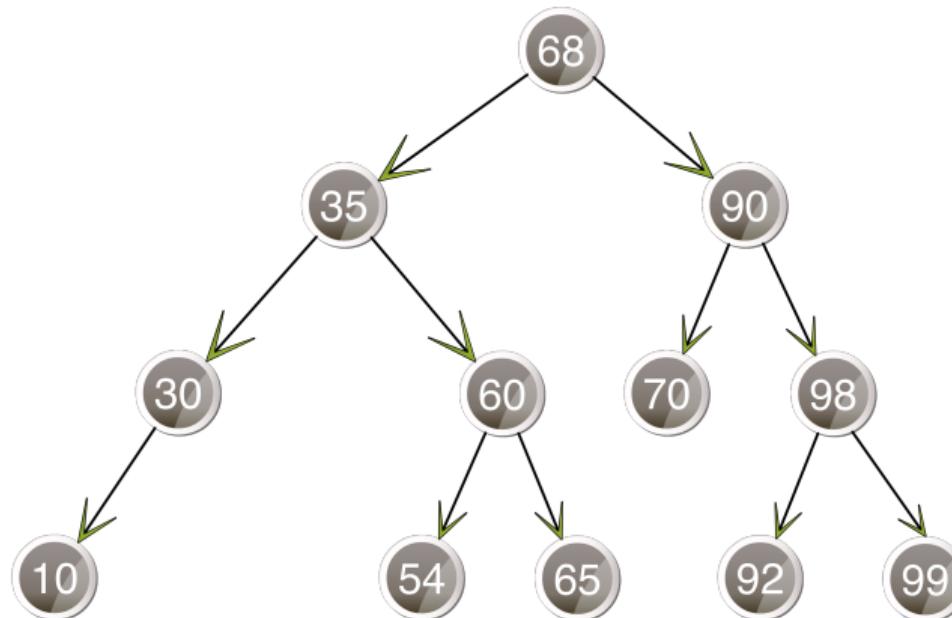
RECORRIDOS DE UN ABB

- ▶ PreOrden (R-I-D): 68-35-30-10-60-54-65-90-70-98-92-99
- ▶ InOrden (I-R-D): 10-30-35-54-60-65-68-70-90-92-98-99
- ▶ PostOrden (I-D-R): 10-30-54-65-60-35-70-92-99-98-90-68



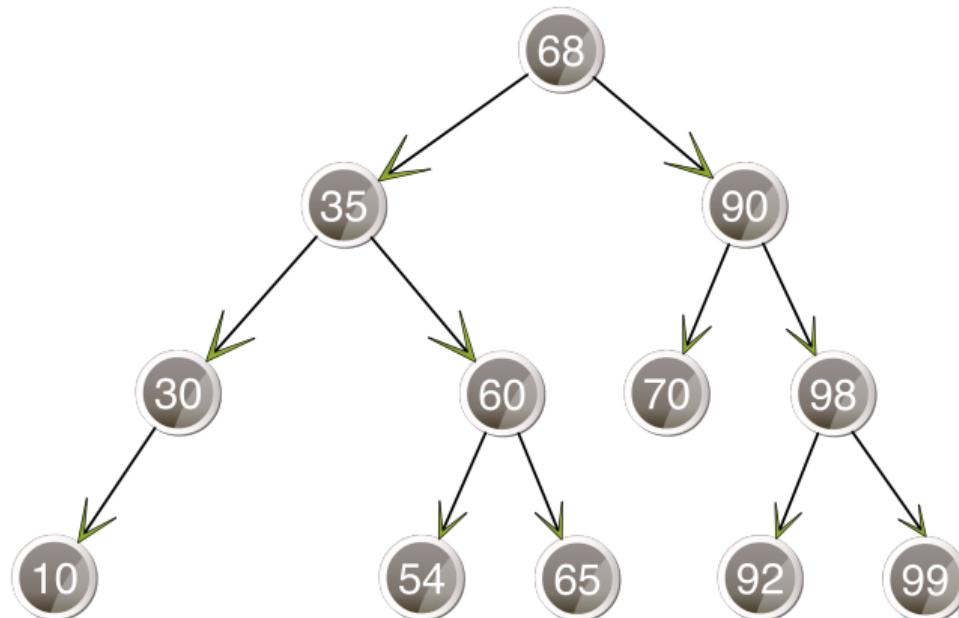
RECORRIDOS DE UN ABB

- ▶ PreOrden (R-I-D): 68-35-30-10-60-54-65-90-70-98-92-99
- ▶ InOrden (I-R-D): 10-30-35-54-60-65-68-70-90-92-98-99
- ▶ PostOrden (I-D-R): 10-30-54-65-60-35-70-92-99-98-90-68



RECORRIDOS DE UN ABB

- ▶ PreOrden (R-I-D): 68-35-30-10-60-54-65-90-70-98-92-99
- ▶ InOrden (I-R-D): 10-30-35-54-60-65-68-70-90-92-98-99
- ▶ PostOrden (I-D-R): 10-30-54-65-60-35-70-92-99-98-90-68



INSERCIÓN EN UN ABB

- ① Debe compararse el valor o dato a insertar con la raíz del árbol.
- ② Si es mayor, debe avanzarse hacia el subárbol derecho.
- ③ Si es menor, debe avanzarse hacia el subárbol izquierdo.
- ④ Repetir sucesivamente el paso 1 hasta que se cumpla alguna de las siguientes condiciones:
 - ▶ El subárbol derecho es igual a vacío, o el subárbol izquierdo es igual a vacío; en cuyo caso se procederá a insertar el elemento en el lugar que le corresponde.
 - ▶ El valor o dato que quiere insertarse es igual a la raíz del árbol; en cuyo caso no se realiza la inserción.

INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

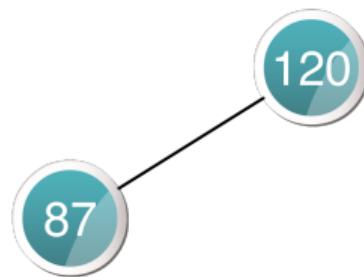
120 - 87 - 43 - 65 - 140 - 99 - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

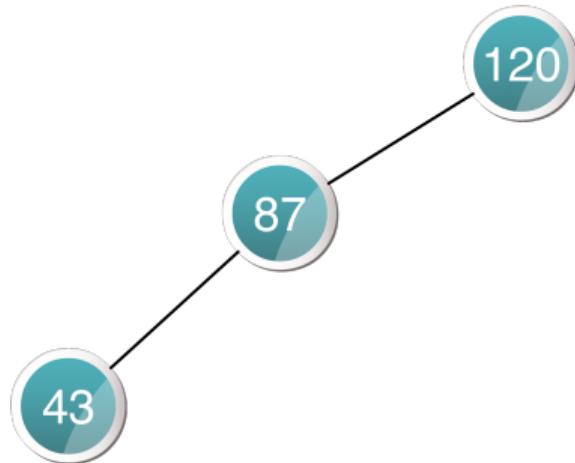
120 - **87** - 43 - 65 - 140 - 99 - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

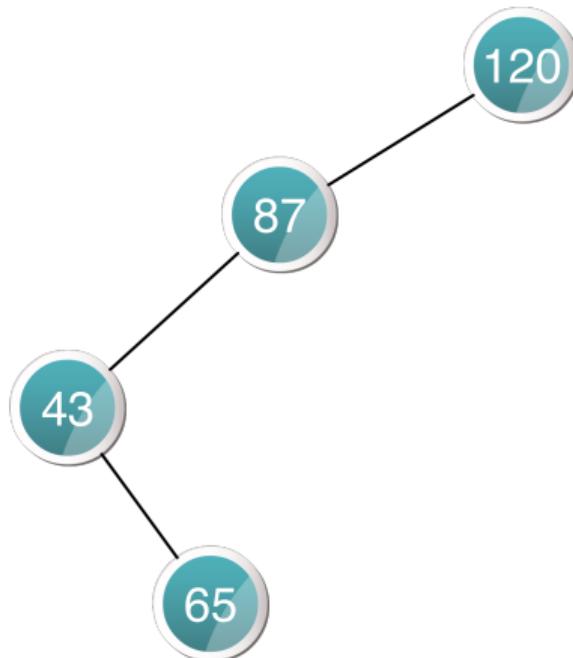
120 - 87 - **43** - 65 - 140 - 99 - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

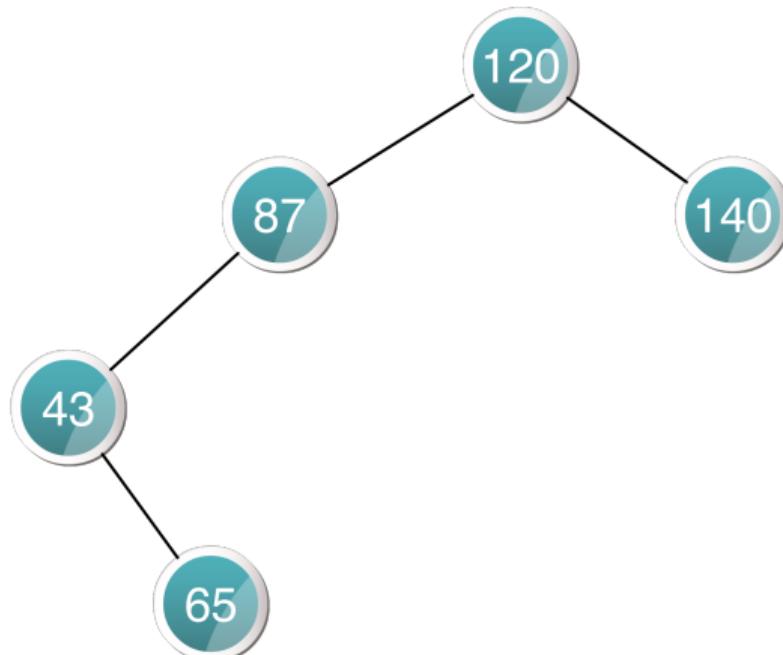
120 - 87 - 43 - **65** - 140 - 99 - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

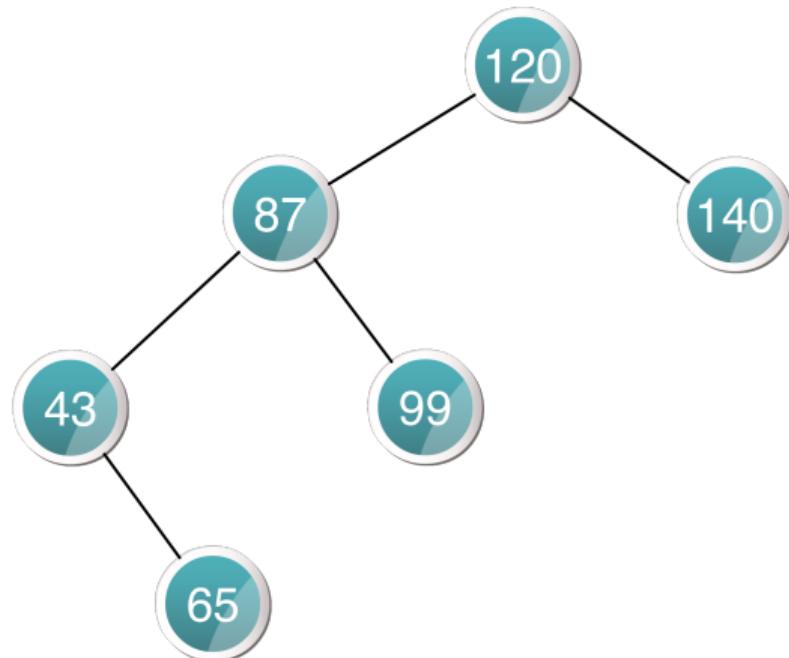
120 - 87 - 43 - 65 - **140** - 99 - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

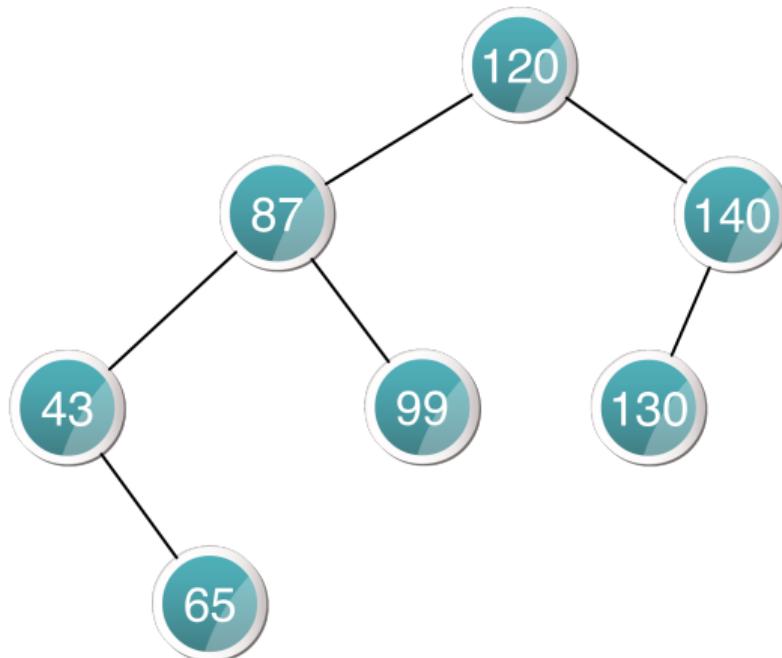
120 - 87 - 43 - 65 - 140 - **99** - 130 - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

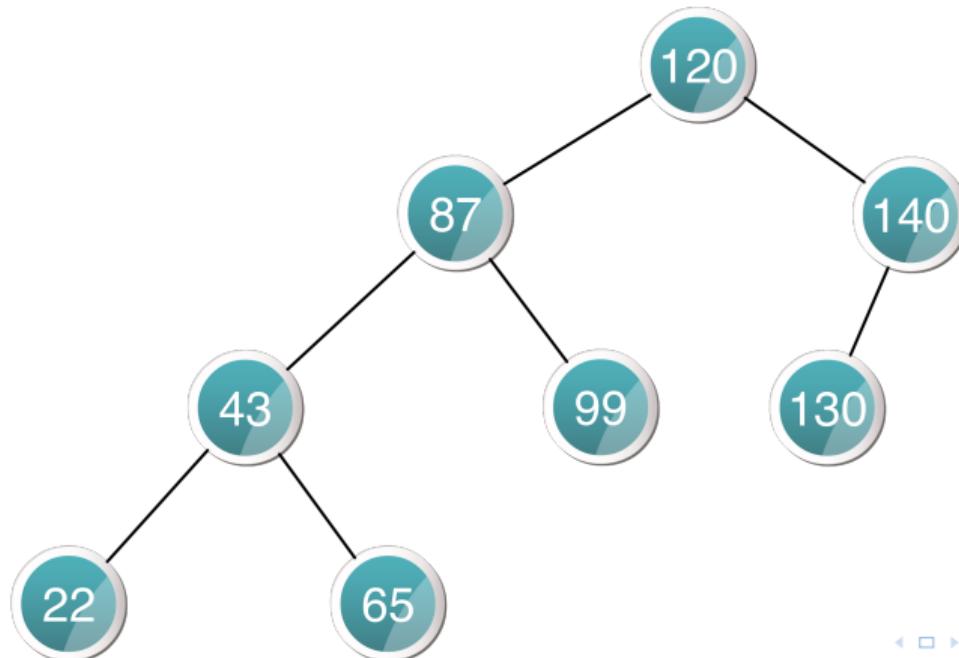
120 - 87 - 43 - 65 - 140 - 99 - **130** - 22 - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

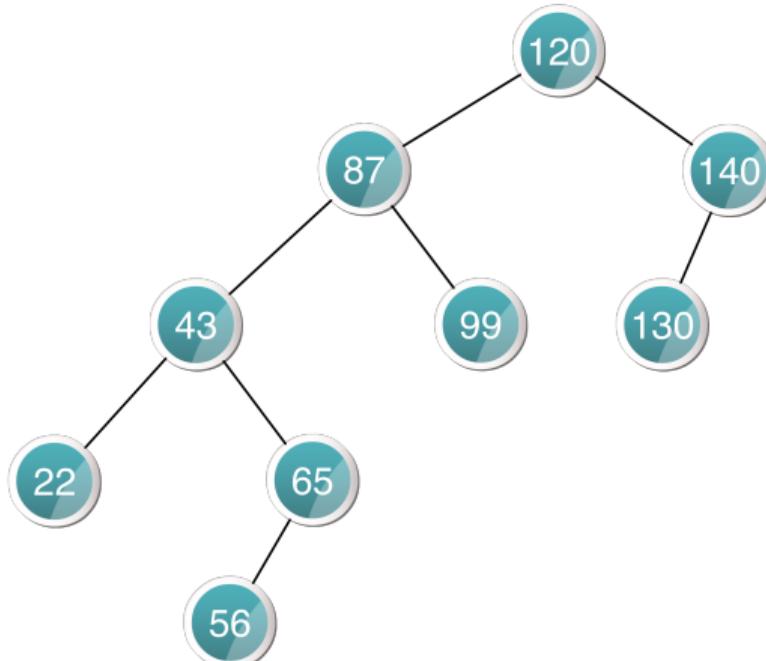
120 - 87 - 43 - 65 - 140 - 99 - 130 - **22** - 56



INSERCIÓN EN UN ABB

- ▶ Suponga que quieren insertarse los siguientes datos en un árbol binario de búsqueda que se encuentra vacío.

120 - 87 - 43 - 65 - 140 - 99 - 130 - 22 - 56



PROCESO PARA INSERTAR NODOS...

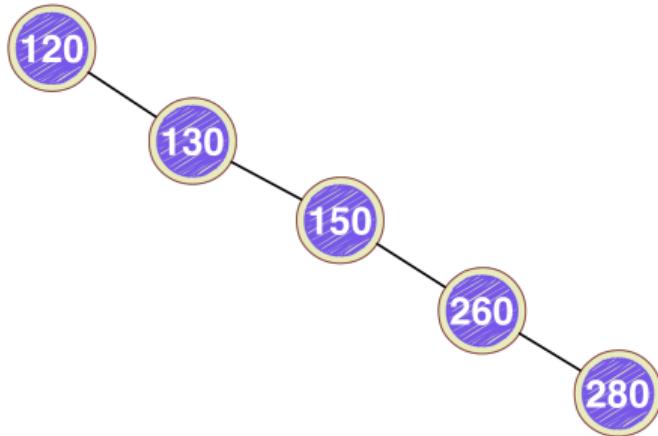
- ▶ Reglas:
 - ▶ El valor a insertar no existe en el árbol.
 - ▶ El nuevo nodo será un Nodo Hoja del árbol.
- ▶ Procedimiento
 - 1 Buscar el Nodo Padre del nodo a agregar.
 - 2 Agregar el nodo hoja.

COMENTARIOS

- ▶ 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.

ABB DESBALANCEADO

- Este siguiente árbol está desbalanceado porque los valores se agregaron en el siguiente orden:
120, 130, 150, 260, 280



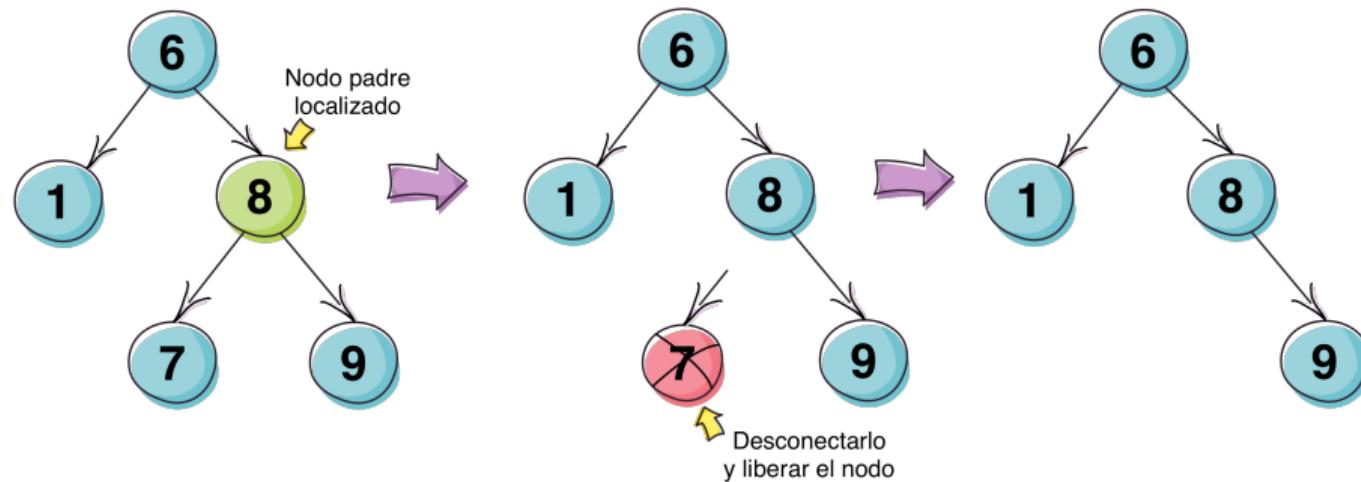
ELIMINAR UN NODO

- ▶ Para eliminar un nodo existen los siguientes casos:
 - ① Si el elemento a borrar es Terminal (hoja),
 - ② Si el elemento a borrar tiene un solo hijo,
 - ③ Si el elemento a borrar tiene los dos hijos.

CASO 1: ELIMINAR Hoja

- ▶ Si el elemento a borrar es terminal (hoja), simplemente se elimina.
- ▶ aux = aux.izq = null

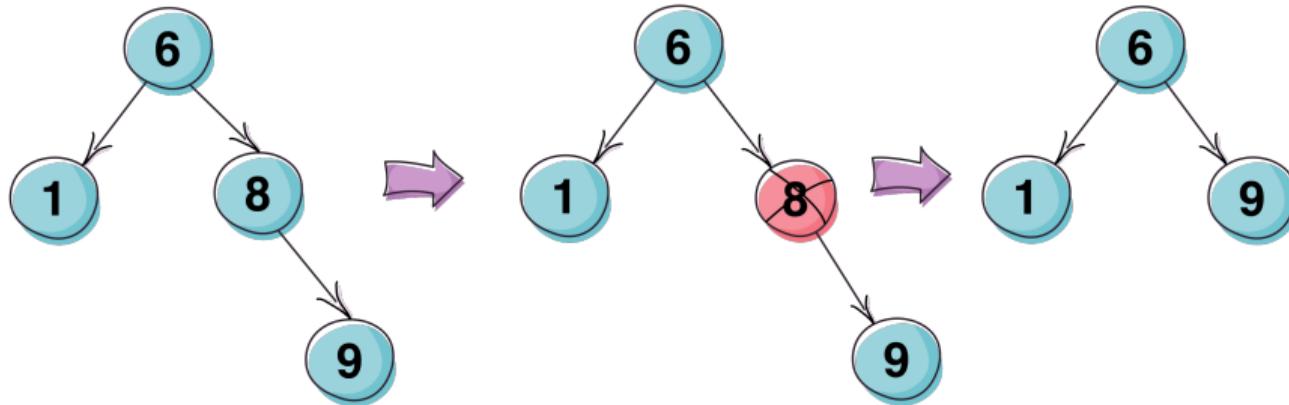
Ej: Eliminar el valor 7.



CASO 2: ELIMINAR NODO CON UN HIJO

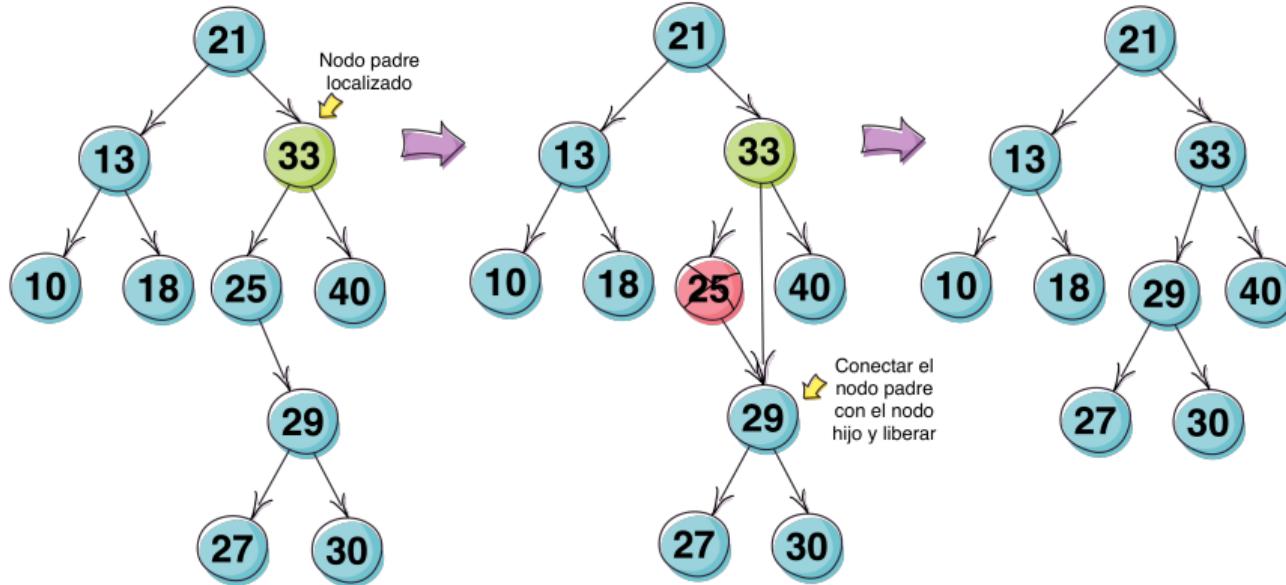
- ▶ Si el elemento a borrar tiene un solo hijo, entonces tiene que sustituirlo por el hijo.

Ej: Eliminar el valor **8**.



CASO 2: ELIMINAR NODO CON UN HIJO

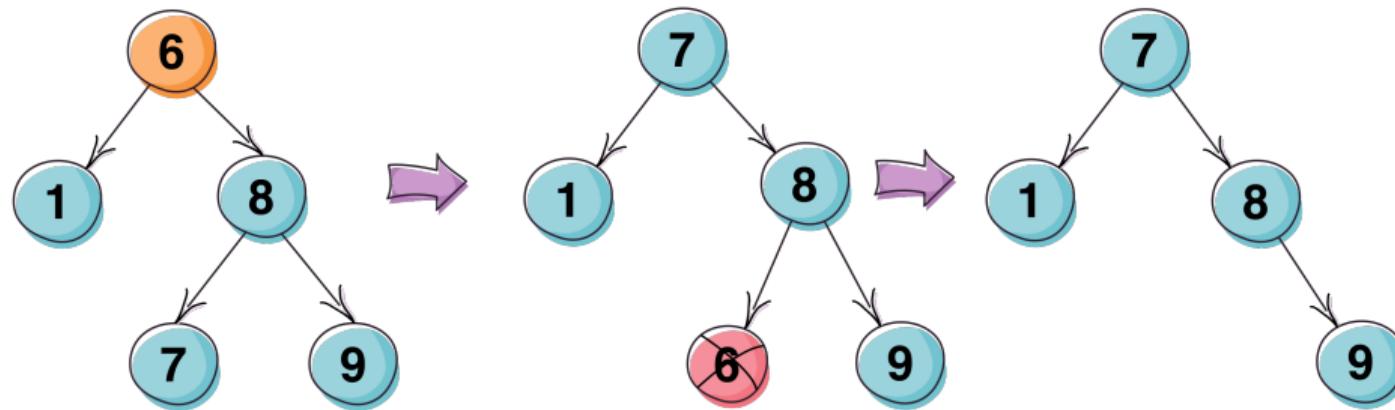
Ej: Eliminar el valor 25.



CASO 3: ELIMINAR NODO CON DOS HIJO

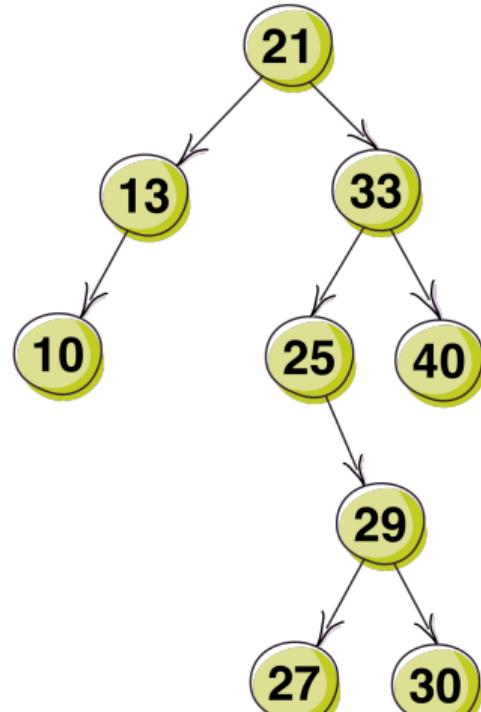
- ▶ Si el elemento a borrar tiene los dos hijos, entonces se tienen que sustituir por sucesor o predecesor inorden.

Ej: Eliminar el valor 6.



CASO 3: ELIMINAR NODO CON DOS HIJO

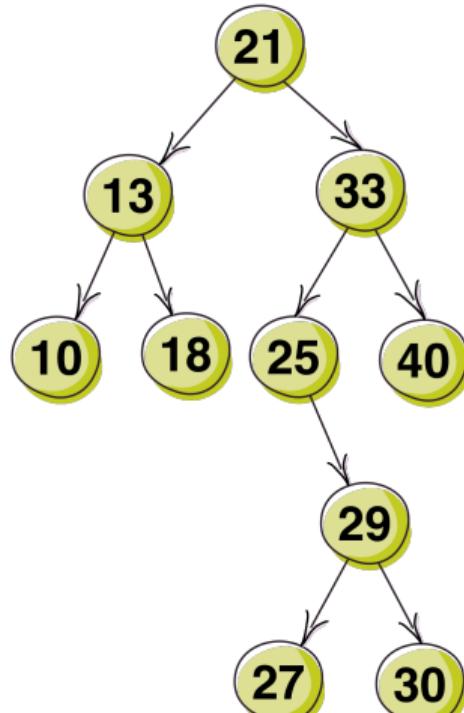
- ▶ Predecesor
- ▶ Uno a la IZQUIERDA y todo a la DERECHA



El Predecesor de:	Es:
33	30
21	13
29	27

CASO 3: ELIMINAR NODO CON DOS HIJO

- ▶ Sucesor
- ▶ Uno a la DERECHA y todo a la IZQUIERDA



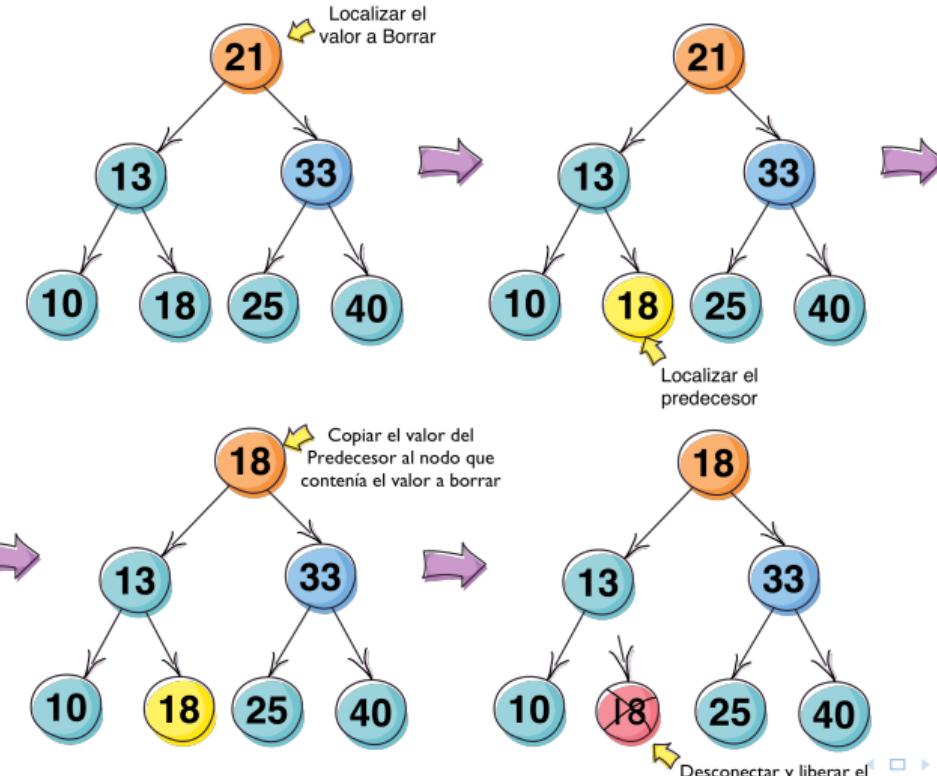
El Sucesor de:	Es:
21	25
33	40
29	30

CASO 3: ELIMINAR NODO CON DOS HIJO

- 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).

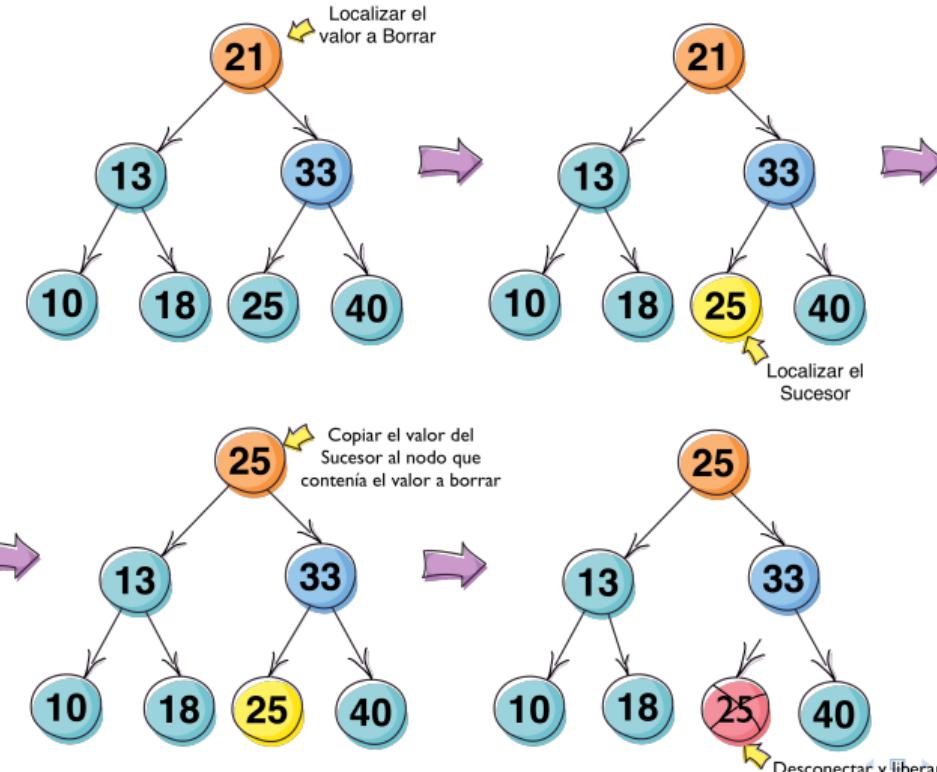
CASO 3: ELIMINAR NODO CON DOS HIJO

Ej: Eliminar el valor 21 utilizando el **predecesor**



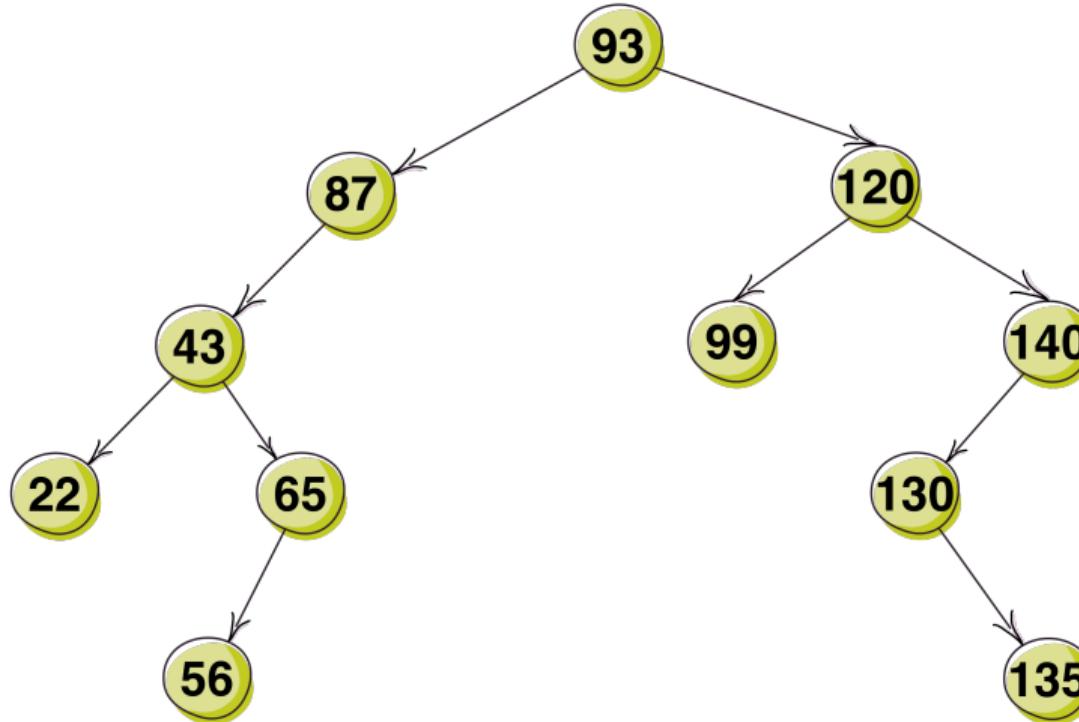
CASO 3: ELIMINAR NODO CON DOS HIJO

Ej: Eliminar el valor 21 utilizando el **sucesor**



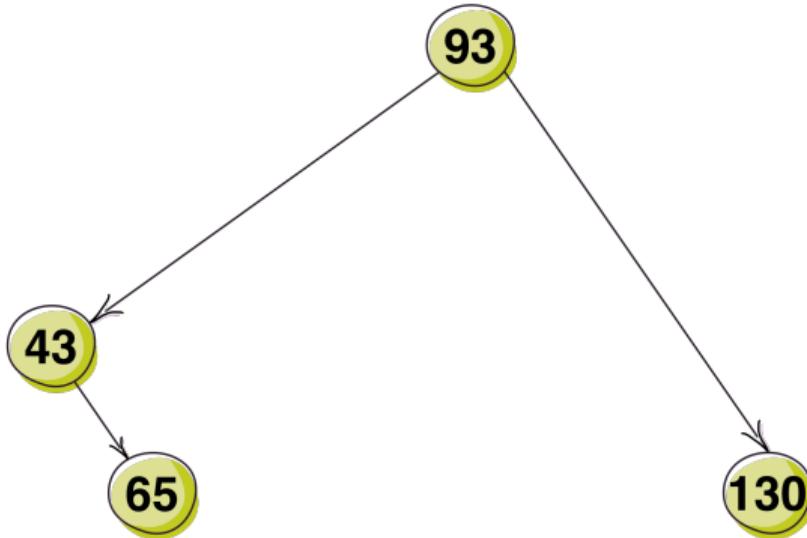
EJERCICIO 1

- Eliminar, del ABB, las claves: 22, 99, 87, 120, 140, 135, 56 en forma sucesiva.



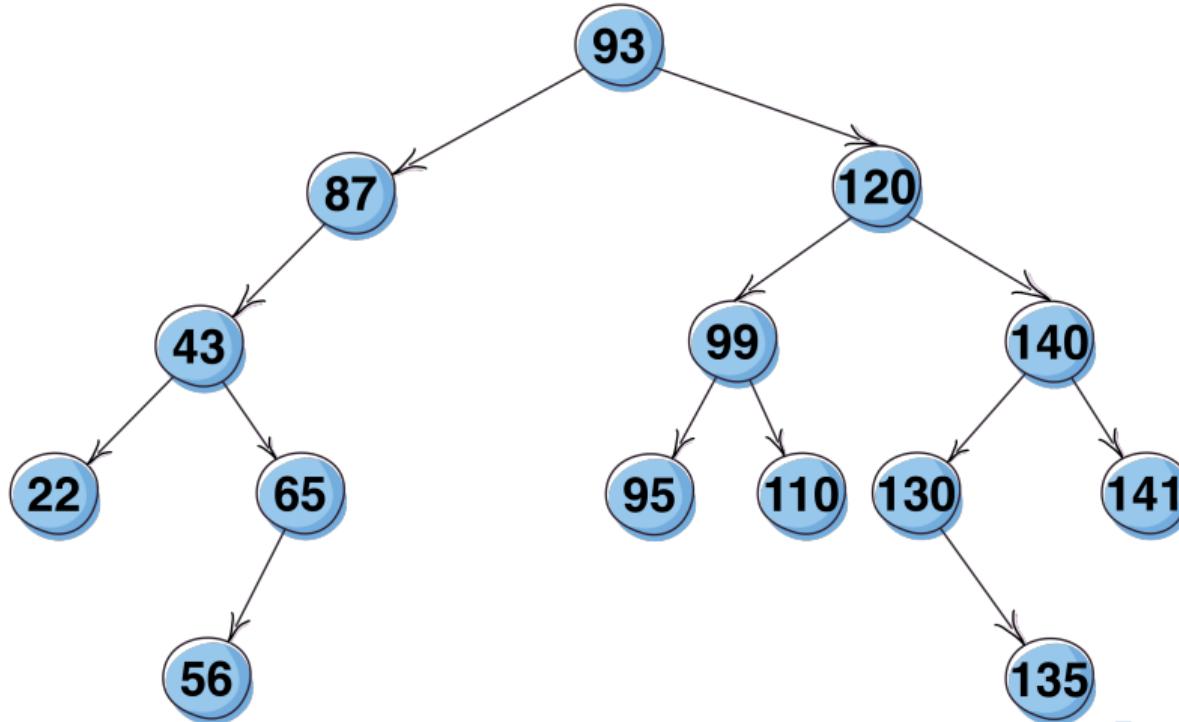
EJERCICIO 1: SOLUCIÓN

- ▶ Eliminar, del ABB, las claves: 22, 99, 87, 120, 140, 135, 56 en forma sucesiva.
- ▶ Solución:



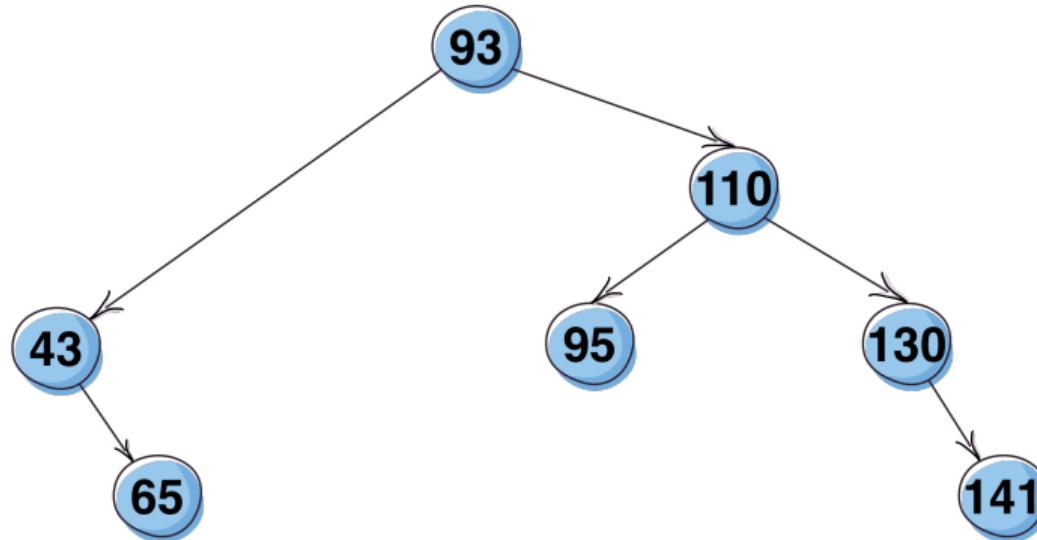
EJERCICIO 2

- ▶ Eliminar, del ABB, las claves: 22, 99, 87, 120, 140, 135, 56 en forma sucesiva.



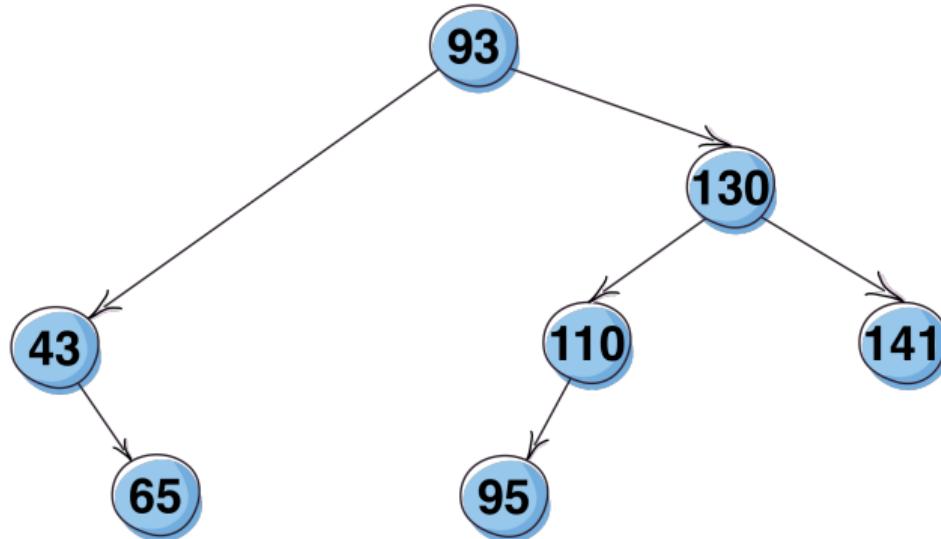
EJERCICIO 2: SOLUCIÓN

- ▶ Eliminar, del ABB, las claves: 22, 99, 87, 120, 140, 135, 56 en forma sucesiva.
- ▶ Solución usando predecesor:



EJERCICIO 2: SOLUCIÓN

- ▶ Eliminar, del ABB, las claves: 22, 99, 87, 120, 140, 135, 56 en forma sucesiva.
- ▶ Solución usando sucesor:



SECCIÓN SIGUIENTE

1 Árboles Binarios

2 Construcción

3 Árbol Binario de Búsqueda

- Recorridos
- Inserción en un ABB
- Eliminar Nodos

4 Implementación de un Árbol Binario

CÓDIGO ESTRUCTURA ÁRBOL BINARIO

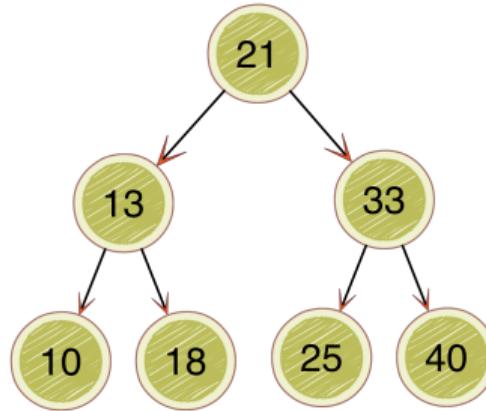
- ▶ Se definirá el árbol con una clave de tipo entero (puede ser cualquier otro tipo de datos) y dos hijos: izquierdo (izq) y derecho (der).
- ▶ Para representar dinámicamente los enlaces con los hijos se utilizan punteros. El árbol vacío se representará con un puntero nulo.
- ▶ Un árbol binario puede declararse de la siguiente manera:

```
struct nodoarbol{  
    int clave;  
    struct nodoarbol *izq;  
    struct nodoarbol *der;  
};  
typedef struct nodoarbol NODO; //definicion de tipo nodo  
NODO *creaArbol(NODO *r);
```

RECORRIDO PREORDEN

- ▶ Visita el nodo actual, y después visita el subárbol izquierdo y una vez visitado, visitar el subárbol derecho en PreOrden.
- ▶ Es un proceso recursivo por naturaleza.

```
void preorden(NODO *a) {
    if (a != NULL) {
        visitar(a);
        preorden(a->izq);
        preorden(a->der);
    }
}
```

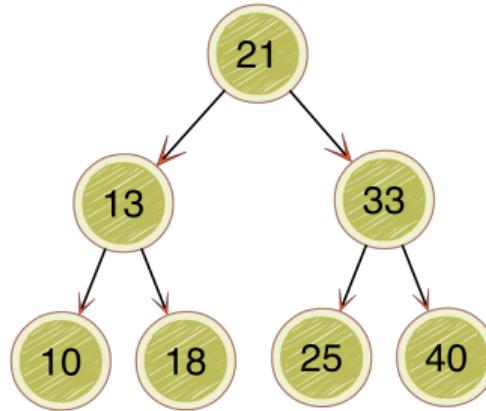


21-13-10-18-33-25-40

RECORRIDO INORDEN

- ▶ Visita el subárbol izquierdo en InOrden, después visita el nodo actual, y una vez visitado, visita el subárbol derecho en InOrden.
- ▶ Es un proceso recursivo por naturaleza.

```
void  inorden(NODO *a) {
    if (a != NULL) {
        inorden(a->izq);
        visitar(a);
        inorden(a->der);
    }
}
```

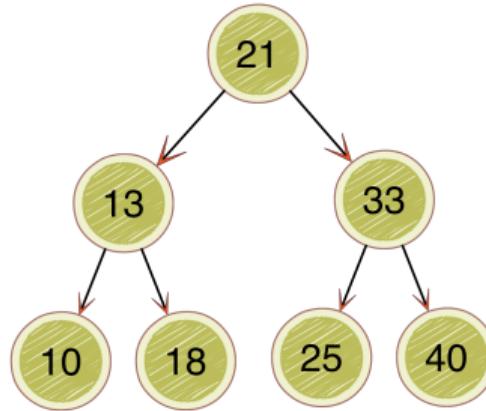


10-13-18-21-25-33-40

RECORRIDO POSTORDEN

- ▶ Visita el subárbol izquierdo en PostOrden, después visita el subárbol derecho en PostOrden, y una vez visitados, visita el nodo actual.
- ▶ Es un proceso recursivo por naturaleza.

```
void postorden (NODO *a) {
    if (a != NULL) {
        postorden(a->izq);
        postorden(a->der);
        visitar(a);
    }
}
```



10-18-13-25-40-33-21

RECORRIDO EN AMPLITUD

- ▶ Consiste en ir visitando el árbol por niveles.
- ▶ Primero se visitan los nodos de nivel 1 (raíz), después los nodos de nivel 2, así hasta que ya no queden más.
- ▶ El recorrido no es recursivo sino iterativo, utilizando una cola como estructura de datos auxiliar.
- ▶ El procedimiento consiste en encolar (si no están vacíos) los subárboles izquierdo y derecho del nodo extraído de la cola, y seguir desencolando y encolando hasta que la cola esté vacía.

RECORRIDO EN AMPLITUD

```
void amplitud(NODO *a) {  
    tCola cola; //las claves de la cola seran de tipo arbol binario  
    arbol *aux;  
    if (a != NULL) {  
        CrearCola(cola);  
        encolar(cola, a);  
        while (!colavacia(cola)) {  
            desencolar(cola, aux);  
            visitar(aux);  
            if (aux->izq != NULL)  
                encolar(cola, aux->izq);  
            if (aux->der != NULL)  
                encolar(cola, aux->der);  
        }  
    }  
}
```

EJERCICIOS PROPUESTOS

- ① Desarrollar versiones No Recursivas para la implementación de recorridos PreOrden, InOrden y PostOrden de un AB en sus versiones dinámicas y estáticas.
- ② Considérese la sustitución de la cola por una pila en el recorrido en amplitud. ¿Qué tipo de recorrido se obtiene?

EJEMPLO DE IMPLEMENTACIONES

CUENTA HOJAS DE UN AB

```
int cuentaHojas(NODO *a) {
    if (a != NULL) {
        if ((a->izq ==NULL) && (a->der ==NULL))
            return(1);
        return(cuentaHojas(a->izq )+cuentaHojas(a->der));
    }
    else return(0);
}
```

CUENTA NODOS INTERNOS DE UN AB

```
int nodoInt(NODO *a) {
    if (a != NULL) {
        return(nodoInt(a->izq )+nodoInt(a->der));
        if ((a->izq !=NULL) || (a->der !=NULL))
            return(1);
    }
    else return(0);
}
```

EJEMPLO DE IMPLEMENTACIONES

CONTAR LOS NODOS DE UN ÁRBOL

```
int nodos(NODO *a) {
    if (a == null)
        return 0;
    else
        return (1+nodos(a->izq)+nodos(a->der));
}
```

SUMAR LOS NODOS QUE HAY EN EL ÁRBOL

```
int sumaNodo(NODO *a) {
    if (a== null)
        return 0;
    else
        return (sumaNodo(a->der)+sumaNodo(a->izq));
}
```

EJEMPLO DE IMPLEMENTACIONES

CALCULAR PROFUNDIDAD DEL ÁRBOL

```
int profundidad(NODO *a) {  
    if (a== null)  
        return (0);  
    if (profundidad (a->der) > profundidad (a->izq))  
        return (profundidad (a->der) + 1);  
    else  
        return ( profundidad(a->izq) + 1);  
}
```

IMPLEMENTACIÓN ITERATIVA DE BUSCAR

```
data Buscar_ABB(abb t, clave k){  
    abb p;          dato e;          e=NULL;          p=t;  
    if (!estaVacio(p)){  
        while (!estaVacio(p) && (p->k!=k) ) {  
            if (k < p->k){ p=p->l ;}  
            if (p->k < k){ p=p->r ;}  
        }  
        if (!estaVacio(p)&&(p->d!=NULL) ) {e=copiaDato (p->d) ;}  
    }  
    return e;  
}  
int buscar(NODO *r, int elem){  
    if (r == NULL)  
        return 0;  
    else if (r->clave < elem)  
        return buscar(r->hDerecho, elem);  
    else if (r->clave > elem)  
        return buscar(r->hlzquierdo, elem);  
    else    return 1;  
}
```

IMPLEMENTACIÓN ITERATIVA DE INSERTAR

```
void insertar(NODO **a, int elem) {
    if (*a == NULL) {
        *a = (NODO *) malloc(sizeof(tArbol));
        (*a)->clave = elem;
        (*a)->hIzquierdo = NULL;
        (*a)->hDerecho = NULL;
    }
    else if ((*a)->clave < elem)
        insertar(&(*a)->hDerecho, elem);
    else if ((*a)->clave > elem)
        insertar(&(*a)->hIzquierdo, elem);
}
```

IMPLEMENTACIÓN ITERATIVA DE BORRAR

```
void borrar(NODO **a, int elem){  
    NODO *aux;  
    if (*a == NULL) return;  
    if ((*a)->clave < elem) borrar(&(*a)->hDerecho, elem);  
    else if ((*a)->clave > elem) borrar(&(*a)->hIzquierdo, elem);  
    else if ((*a)->clave == elem) {  
        aux = *a;  
        if ((*a)->hIzquierdo == NULL) *a = (*a)->hDerecho;  
        else if ((*a)->hDerecho == NULL) *a = (*a)->hIzquierdo;  
        else reemplazar(&(*a)->hIzquierdo, &aux);  
        free(aux);  
    }  
}  
  
void reemplazar(NODO **a, NODO **aux){  
    if ((*a)->hDerecho == NULL){  
        (*aux)->clave = (*a)->clave;  
        *aux = *a;  
        *a = (*a)->hIzquierdo;  
    }  
    else  
        reemplazar(&(*a)->hDerecho, aux);  
}
```

IMPLEMENTACIÓN ITERATIVA DE BORRAR

- ▶ Los árboles son estructuras de datos eficientes.
- ▶ Todos los árboles poseen un hijo izquierdo y un hijo derecho.
- ▶ Existen formas para recorrer un árbol.
 - ▶ Preorden
 - ▶ Inorden
 - ▶ Postorden
 - ▶ Anchura
- ▶ Una mejora son los árboles AVL, árboles Rojinegros y árboles AA.