DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Análisis de Algoritmos y Estructura de Datos

TDA árbol, TDA árbol binario y TDA árbol binario de búsqueda

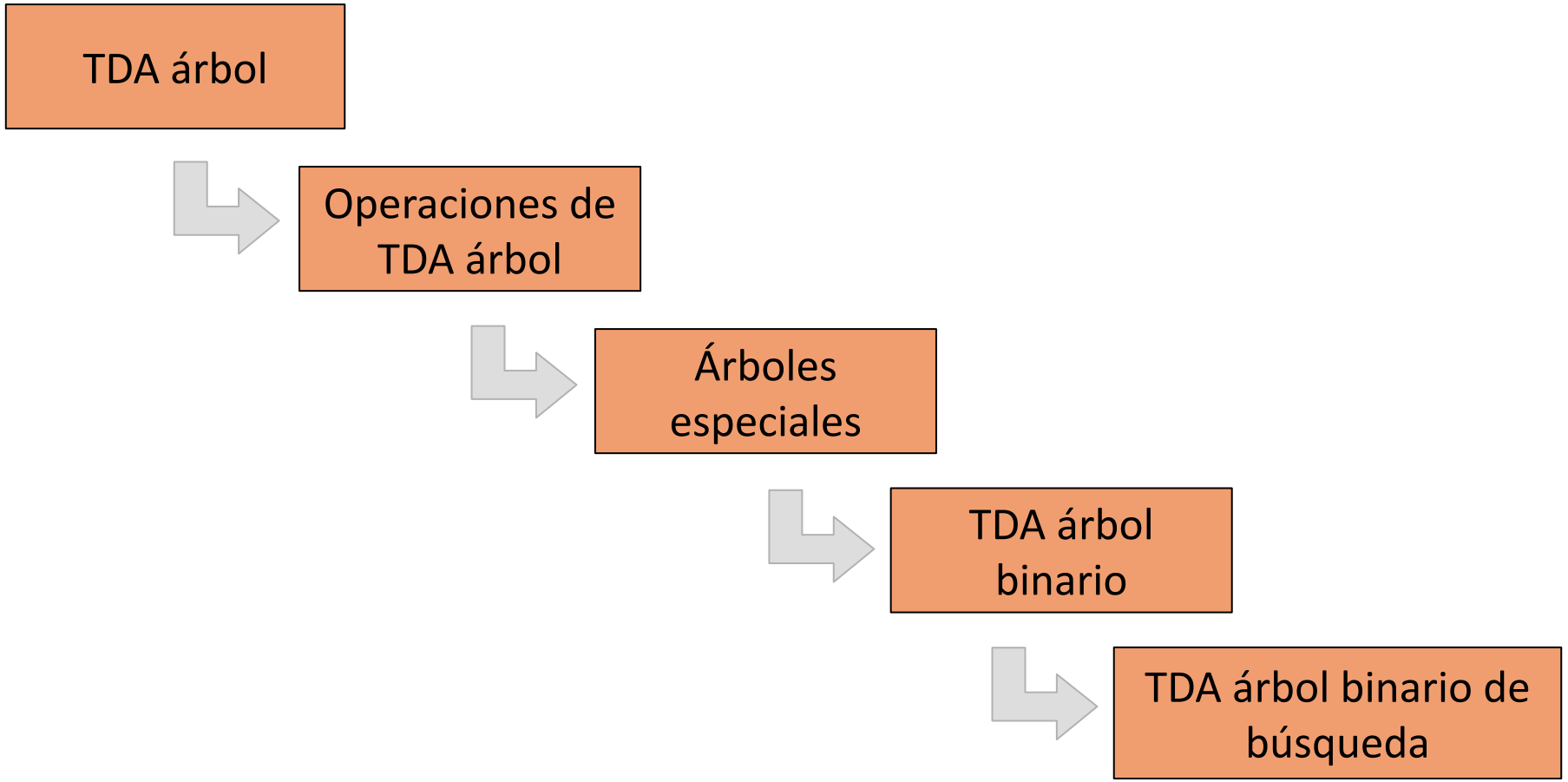
Prof. Violeta Chang C

Semestre 2 – 2023

TDA árbol

- **Contenidos**:
  - Especificación de TDA árbol
  - Operaciones con árboles
  - Árboles binarios y de búsqueda
- **Objetivos**:
  - Entender y explicar estructura de datos de TDA árbol
  - Comprender funcionamiento de operaciones con árboles y determinar su complejidad
  - Entender algoritmos de recorrido de árboles
  - Conocer tipos de árboles especiales: binarios y de búsqueda

Ruta de la sesión

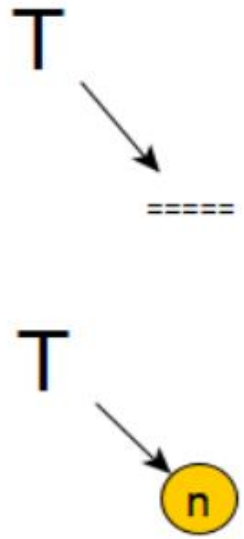


DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

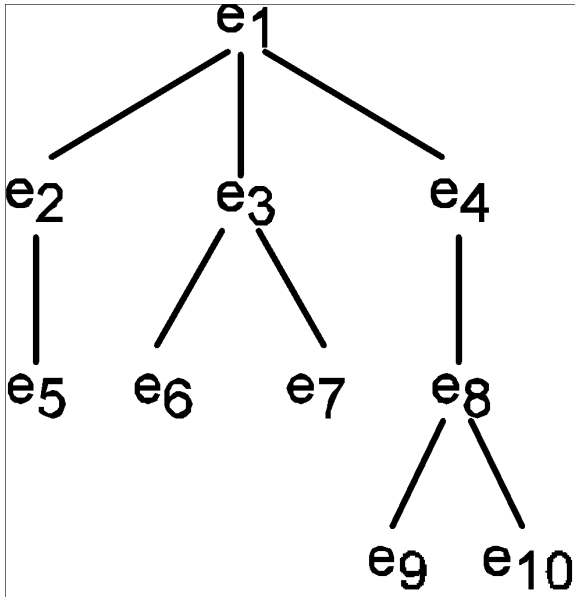
TDA árbol

Especificación de TDA Árbol

- **Estructura de datos**
  - Un árbol (o tree) es una colección no lineal con componentes homogéneos y capacidad ilimitada
  - Un árbol se considera como un grafo acíclico
  - A un árbol con 0 nodos se le conoce como árbol vacío
  - Un único nodo es un árbol cuya raíz es el único nodo existente
  - Puntero externo: apunta a la raíz del árbol
  - Tipos de nodos:
    - Nodo raíz: nodo inicial del árbol
    - Nodo interno: nodo que posee antecesor y sucesor
    - Nodo hoja: nodo que no posee sucesores (hijos)



Especificación de TDA árbol



raíz: e_1
nodos internos: e_2, e_3, e_4, e_8
hojas: e_5, e_6, e_7, e_9, e_{10}

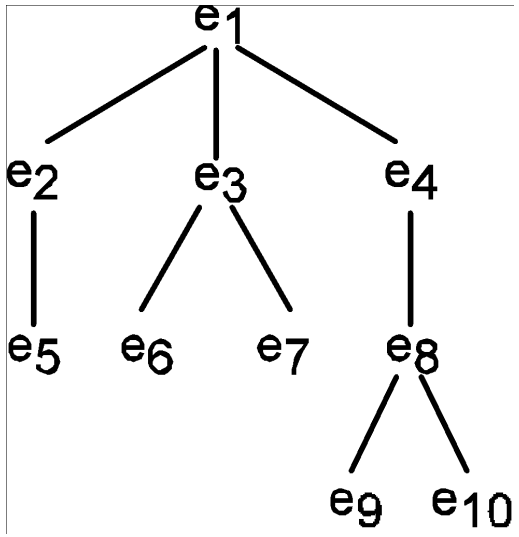
e_1 es el **padre** de e_2, e_3, e_4, entonces e_2, e_3, e_4 son **hermanos** y son los **hijos** de e_1.

El **grado** de e_1 es 3. El grado de e_3 es 2 y el grado de e_9 es 0.

La **aridad** del árbol es 3 → e_1 tiene 3 hijos y no hay otro nodo en el árbol que tenga más hijos.

La **altura** del árbol es 4 → el camino más largo desde la raíz a una hoja tiene 4 nodos.

Especificación de TDA árbol



Ruta

$e_1 - e_3$	numNodos: 2
$e_1 - e_3 - e_7$	numNodos: 3
$e_1 - e_4 - e_8 - e_{10}$	numNodos: 4

Altura de un nodo

e_1	altura: 4
e_3	altura: 2
e_{10}	altura: 1

Profundidad de un nodo (NIVEL)

e_1	profundidad: 1
e_2, e_3, e_4	profundidad: 2
e_5, e_6, e_7, e_8	profundidad: 3
e_9, e_{10}	profundidad: 4

Ancestros de un nodo

e_{10}: e_8, e_4, e_1

Descendientes de un nodo

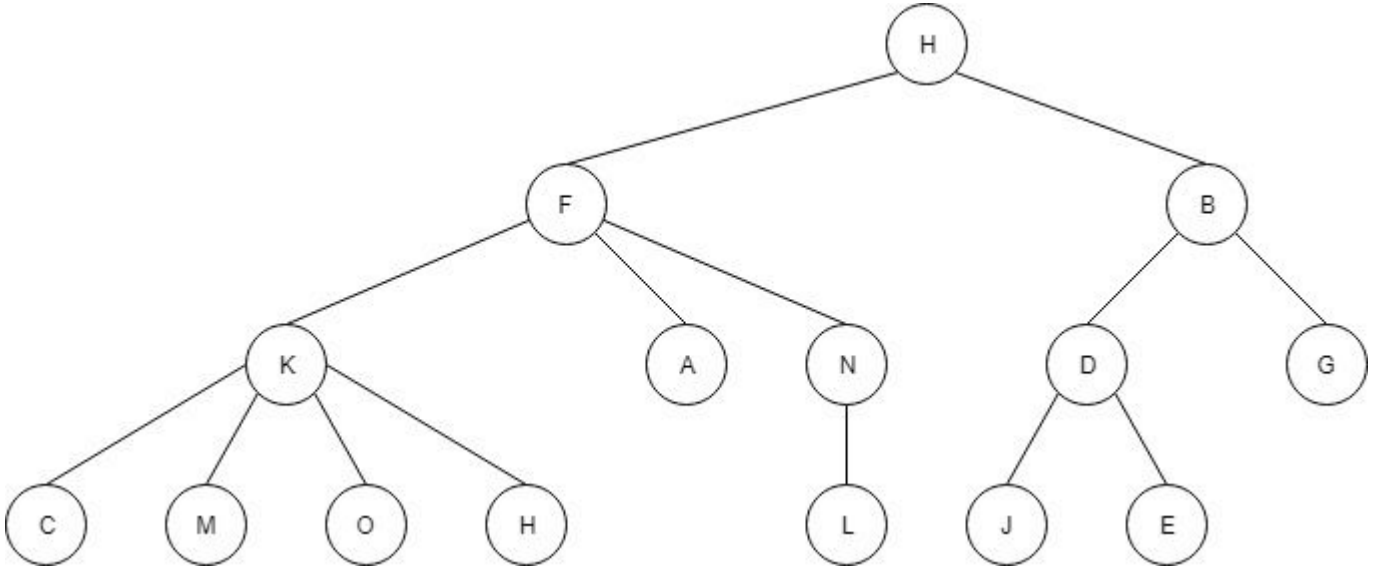
e_4: e_8, e_9, e_{10}

- **Definición recursiva para ancestro**

ancestros(*n*) =
- vacío, si *n* es la raíz del árbol — *caso base*
- padre de *n* + ancestros(padre de *n*), si *n* no es la raíz — *caso recursivo*

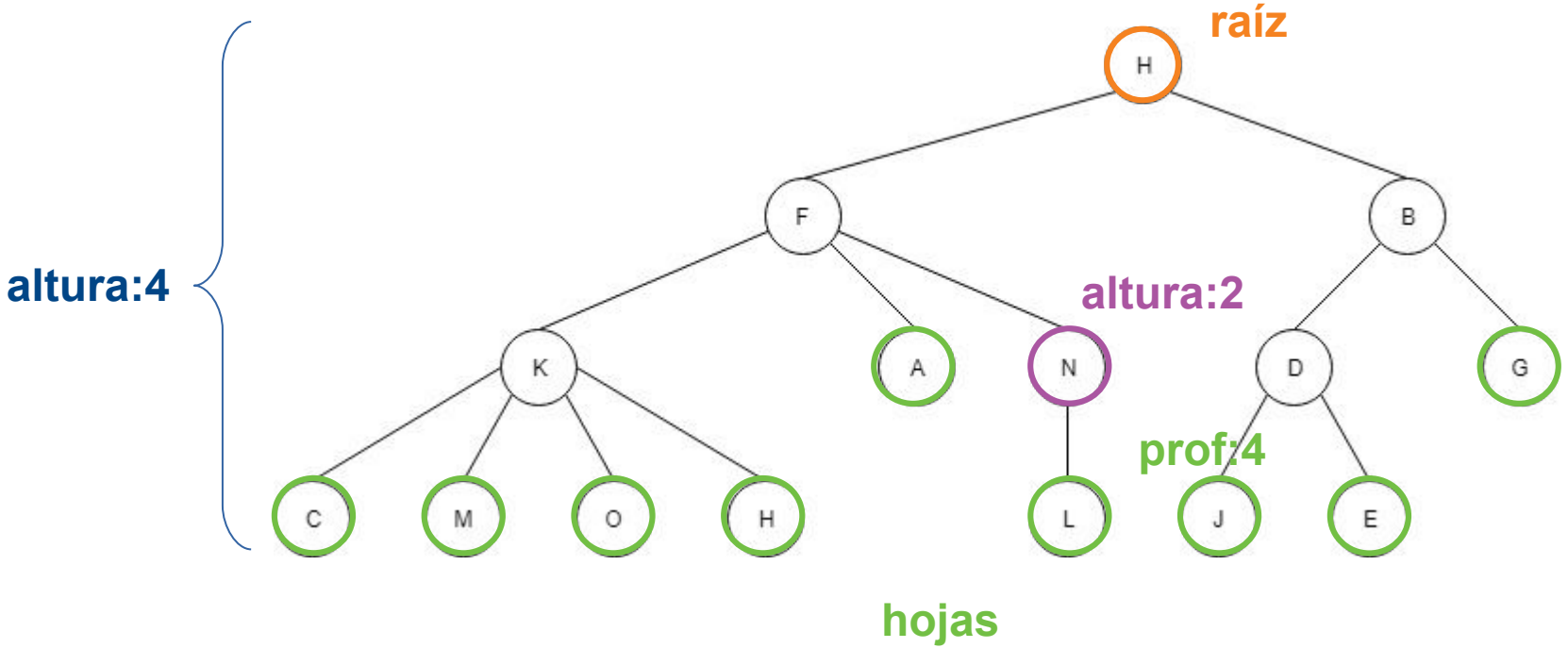
Especificación de TDA árbol

- Para el siguiente árbol, identificar su raíz, sus hojas, su altura, la profundidad del vértice J y la altura del vértice N.



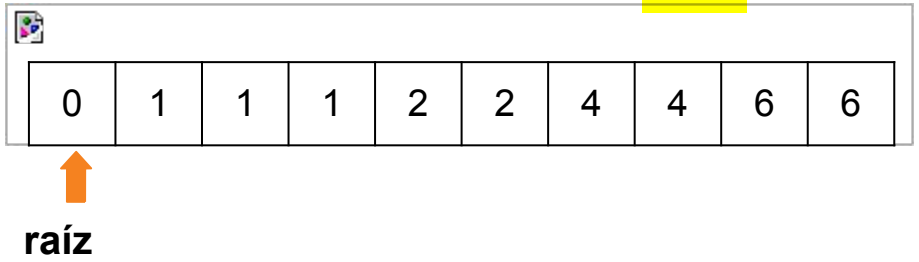
Especificación de TDA árbol

- Para el siguiente árbol, identificar su raíz, sus hojas, su altura, la profundidad del vértice J y la altura del vértice N.

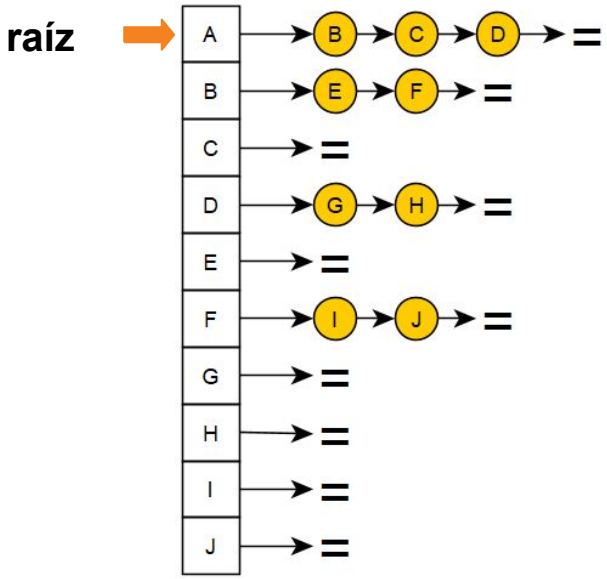


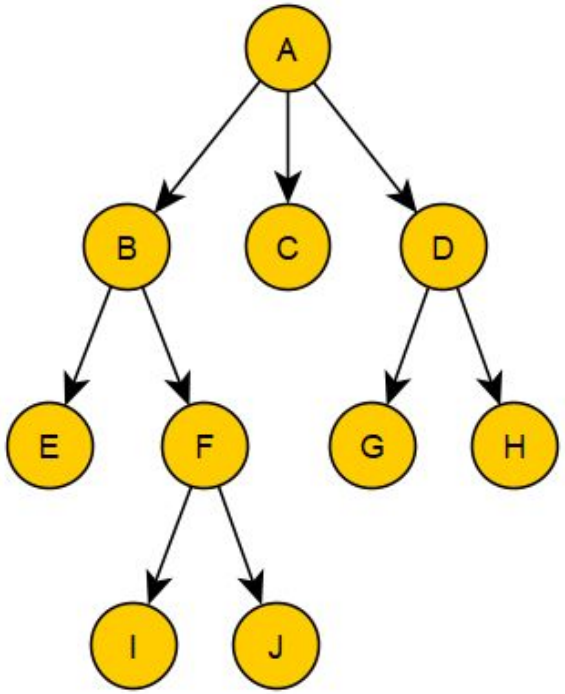
Representación de TDA árbol

- Usando un arreglo:



- **Usando listas enlazadas:**





Especificación de TDA árbol

- **Operaciones:**
  - **esArbolVacío(T)**: determina si árbol T está vacío o no
  - **raíz(T)**: retorna la raíz del árbol T
  - **padre(T,nodo)**: retorna el padre de nodo. Si T es la raíz retorna nulo
  - **esHoja(T,nodo)**: indica si nodo es o no una hoja
  - **insertarNodo(T,nodo)**: inserta nodo en árbol T
  - **eliminarNodo(T,nodo)**: elimina nodo de árbol T
  - **buscarDato(T,dato)**: busca nodo con dato en árbol T
  - **recorrerArbol(T)**: muestra contenido de cada nodo de árbol T

Tipos de árboles

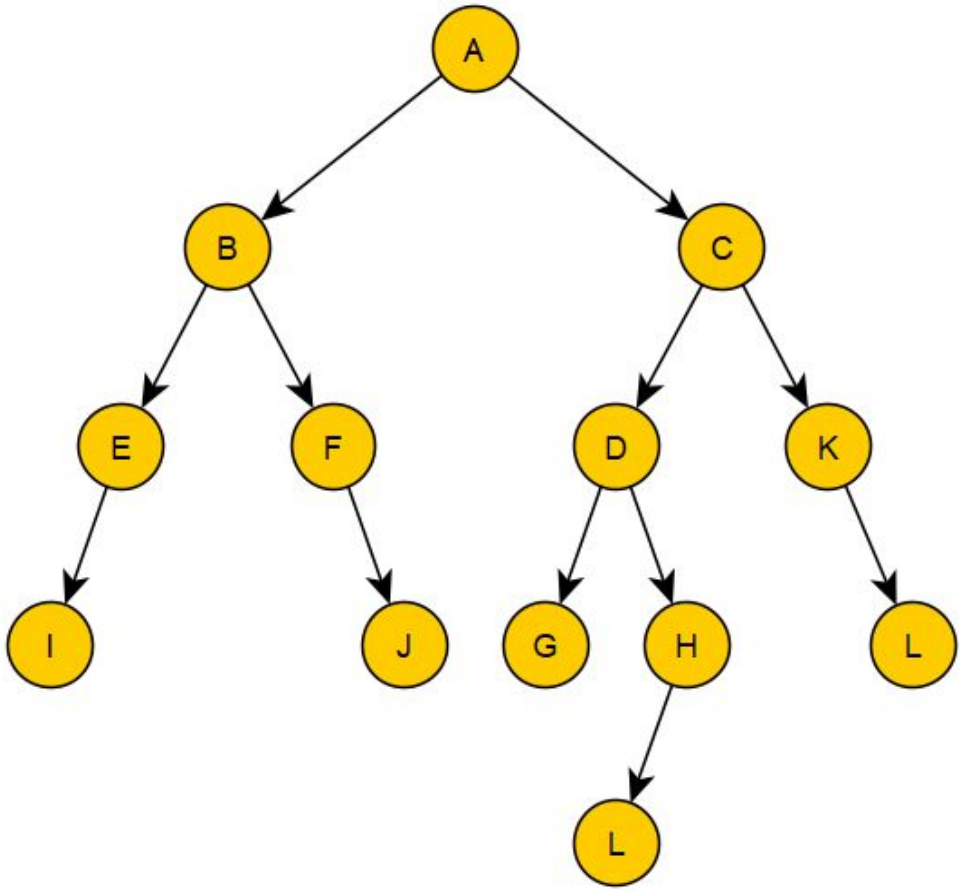
- Varios tipos de árboles → *restricciones* a **estructura de datos** y *restricciones* a algoritmos de **operaciones**
- Tipos de árboles
  - Árboles generales
  - **Árboles binarios** (AB)
  - **Árboles binarios de búsqueda** (ABB)
  - **Árboles balanceados** (AVL)
  - Árboles 2-3
  - Árboles 2-3-4
  - Árboles Red-Black

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

TDA árbol binario

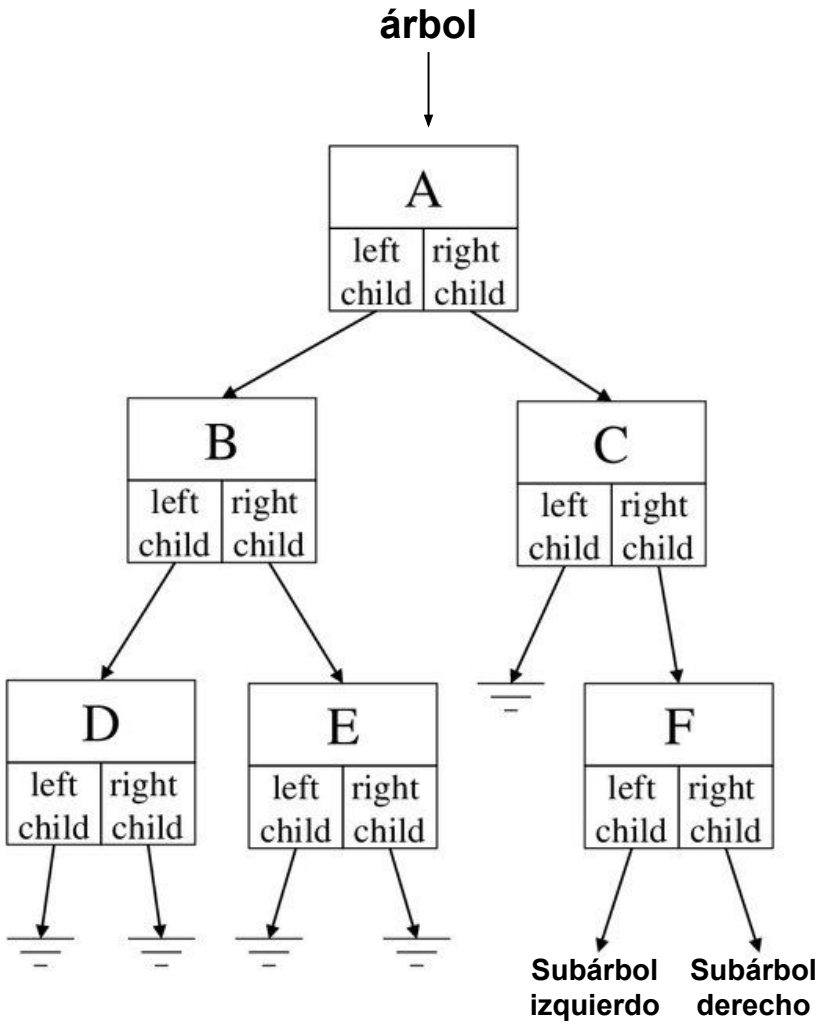
TDA árbol binario

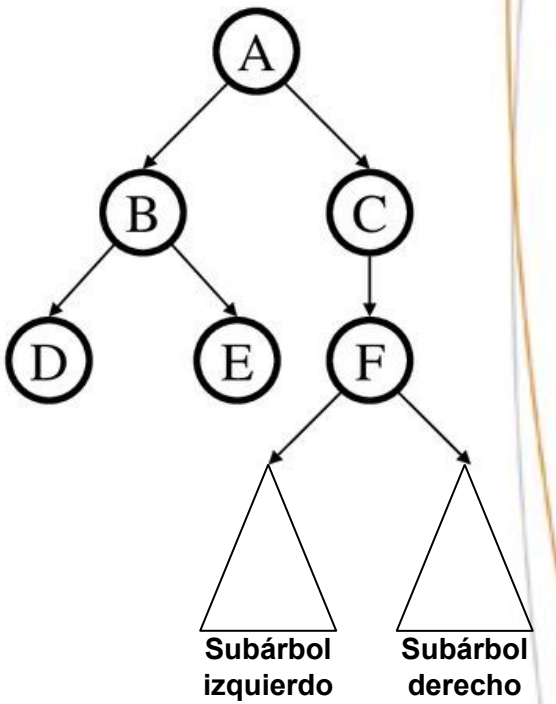
- **Estructura de datos**
  - Un árbol binario (o binary tree) es una colección no lineal con componentes homogéneos y capacidad ilimitada, con la restricción que **cada nodo puede tener a lo más 2 hijos**



TDA árbol binario

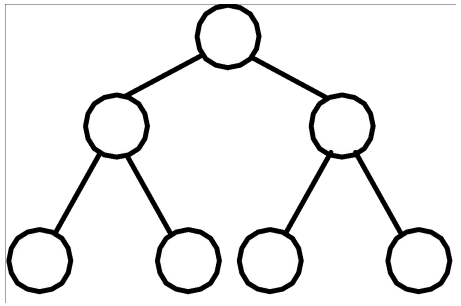
- **Estructura de datos**
  - **Puntero externo**: apunta la raíz del árbol
  - Cada nodo contiene:
    - dato
    - puntero a hijo izquierdo
    - puntero a hijo derecho





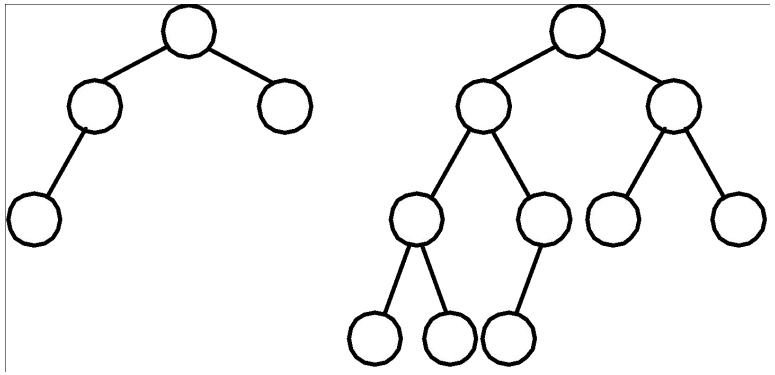
TDA árbol binario

- **Estructura de datos**
  - Un árbol binario es un **árbol con aridad 2**: un nodo puede tener 0, 1 o 2 hijos



(a) Un **AB lleno** de altura 3

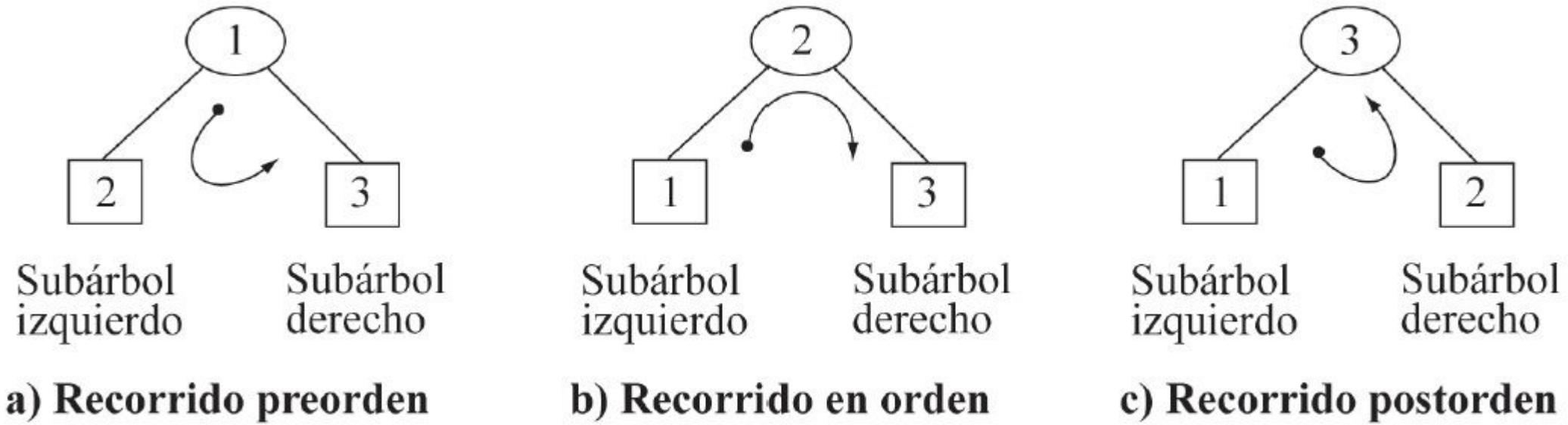
Un **árbol binario lleno** de altura *h* tiene todos los nodos desde el nivel 1 hasta el nivel *h* con dos hijos



(b) **AB completos** de altura 3 y 4.

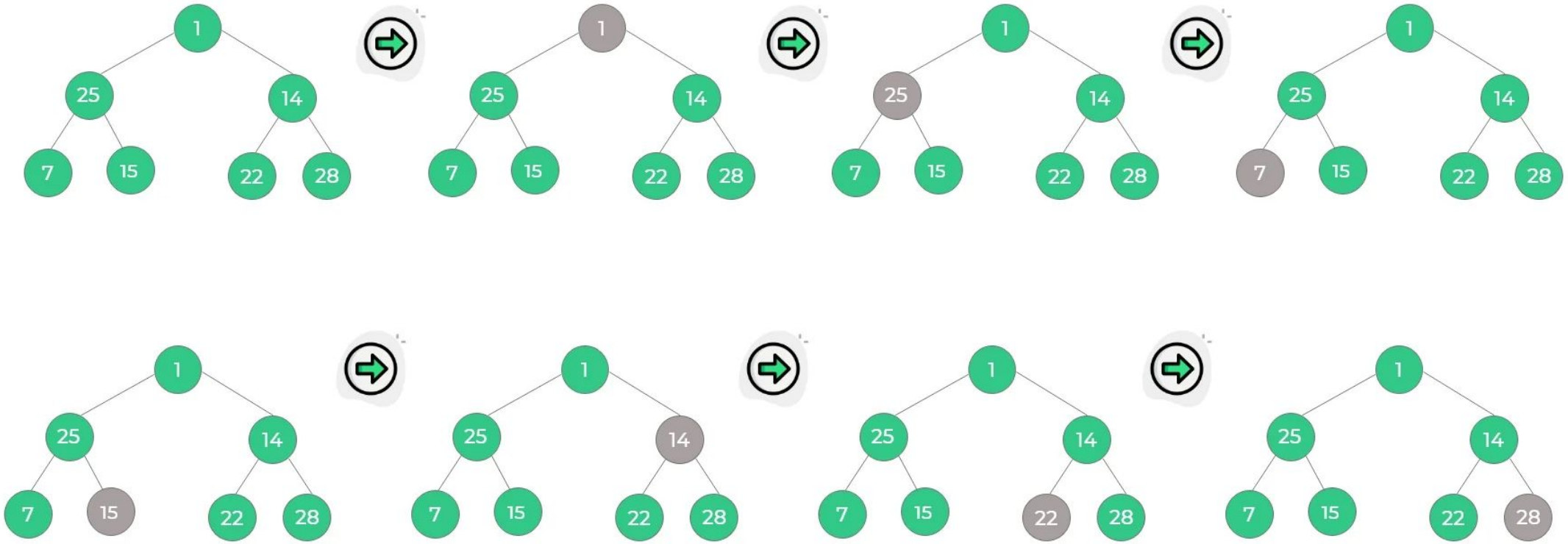
Un **árbol binario completo** de altura *h* tiene todos los nodos desde el nivel 1 hasta el nivel *h* – 1 con dos hijos, y todos los nodos del nivel *h* son contiguos y están en el lado izquierdo del árbol

Recorrido de árbol binario



¿Cuándo se visita la raíz?

Recorrido pre-orden de AB



1, 25, 7, 15, 14, 22, 28

Recorrido pre-orden de AB

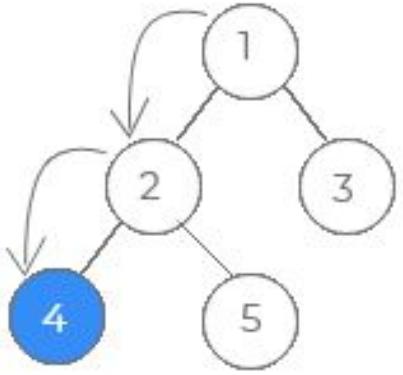
- Desde el puntero externo hacia la raíz del árbol, y nodo por nodo, se recorre cada uno de los elementos para procesarlos de cierto modo según: **raíz-hijolzq–hijoDer**

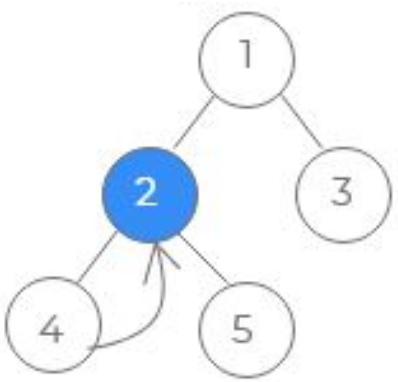
visualización, conteo, promedio, etc

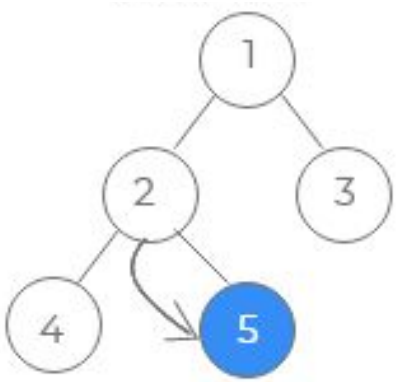
```
recorrerPreOrdenAB(arbol)
    si no(esArbolVacio(arbol)) entonces
        si esHoja(arbol) entonces
            procesar(arbol→dato)
        sino
            procesar(arbol→dato)
            recorrerPreOrdenAB(arbol→izq)
            recorrerPreOrdenAB(arbol→der)
```

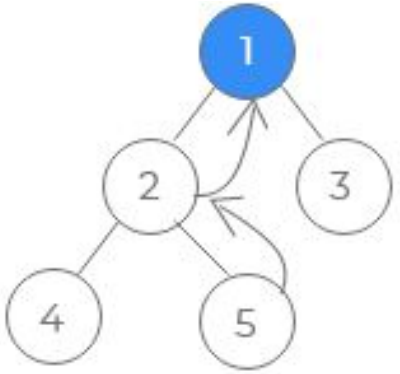
Orden de complejidad: O(n)

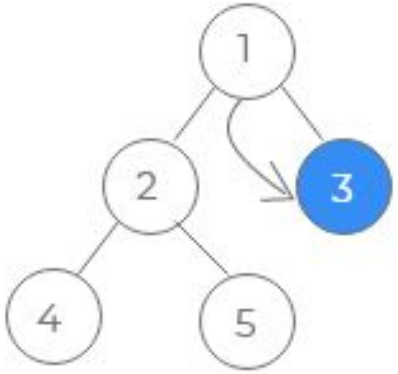
Recorrido en-orden de AB











Inorder - 4 2 5 1 3

PrepInsta

Recorrido en-orden de AB

- Desde el puntero externo hacia la raíz del árbol, y nodo por nodo, se recorre cada uno de los elementos para procesarlos de cierto modo según: **hijolzq–raíz-hijoDer**

visualización, conteo, promedio, etc

```
recorrerEnOrdenAB(arbol)
    si no(esArbolVacio(arbol)) entonces
        si esHoja(arbol) entonces
            procesar(arbol→dato)
        sino
            recorrerEnOrdenAB(arbol→izq)
            procesar(arbol→dato)
            recorrerEnOrdenAB(arbol→der)
```

Orden de complejidad: O(n)

Recorrido post-orden de AB

- Desde el puntero externo hacia la raíz del árbol, y nodo por nodo, se recorre cada uno de los elementos para procesarlos de cierto modo según: **hijoIzq–hijoDer-raíz**

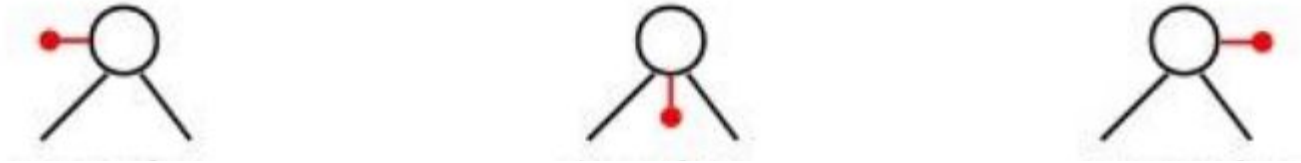
visualización, conteo, promedio, etc

```
recorrerPostOrdenAB(arbol)
    si no(esArbolVacio(arbol)) entonces
        si esHoja(arbol) entonces
            procesar(arbol→dato)
        sino
            recorrerPostOrdenAB(arbol→izq)
            recorrerPostOrdenAB(arbol→der)
            procesar(arbol→dato)
```

Orden de complejidad: O(n)

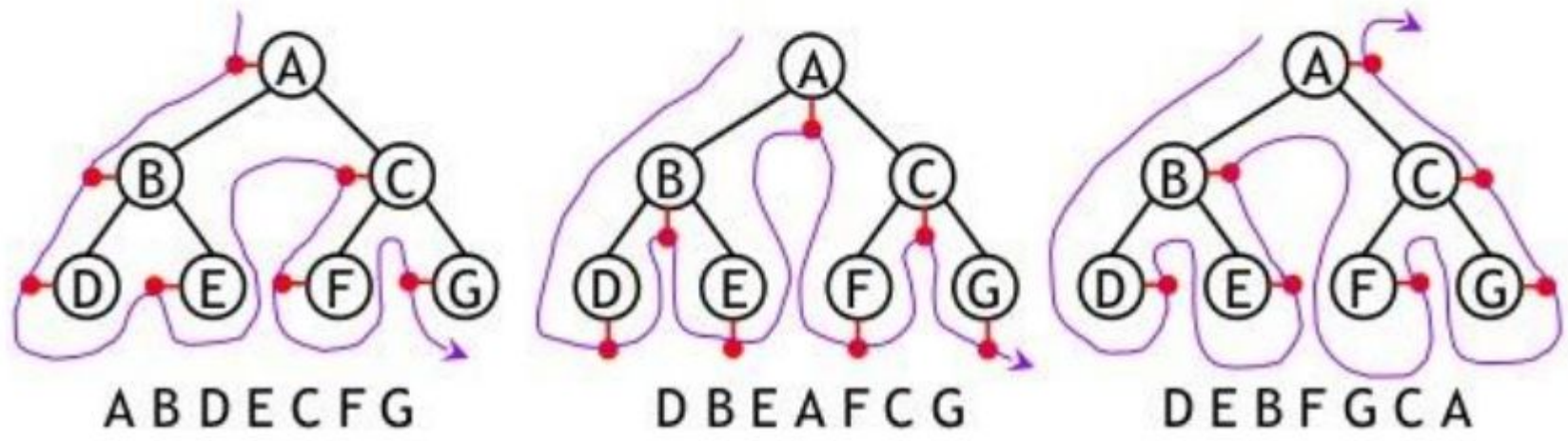
Recorrido de árbol binario

- Estrategia:
  - Usar marcas para cada tipo de recorrido



pre-orden **en-orden** **post-orden**

  - Unir todas las marcas con **una sola curva** para cada recorrido



Insertar nodo en AB

- Siempre se inserta un nodo hoja
- Requisitos: dato del nodo padre y tipo de hijo (izquierdo o derecho)

```
insertarNodoAB(arbol,padre,tipoHijo,dato)
    nuevo ← crearNodoVacio()
    nuevo→dato ← dato
    nuevo→izq ← NULO
    nuevo→der ← NULO
    nodoPadre ← buscarDatoAB(arbol,padre)
    si tipoHijo=izq entonces
        si nodoPadre→izq=NULO entonces
            nodoPadre→izq ← nuevo
    sino
        si nodoPadre→der=NULO entonces
            nodoPadre→der ← nuevo
```

Orden de complejidad: O(n)

Eliminar nodo en AB

- Siempre se elimina un nodo hoja o un nodo interno con un único hijo

```
eliminarNodoAB(arbol,nodoDato)
    nodoPadre ← buscarNodoAB(arbol,nodoDato)
    si esHojaAB(arbol,nodoDato) entonces
        si hijoIzquierdo(arbol,nodoPadre)=nodoDato entonces
            nodoPadre→izq=NULO
        sino
            nodoPadre→der=NULO
    sino
        si nodoDato→izq=NULO o nodoDato→der=NULO entonces
            si hijoIzquierdo(arbol,nodoPadre)=nodoDato entonces
                nodoPadre→izq=nodoDato→[izq|der]
            sino
                nodoPadre→der=nodoDato→[izq|der]
        liberar(nodoDato)
```

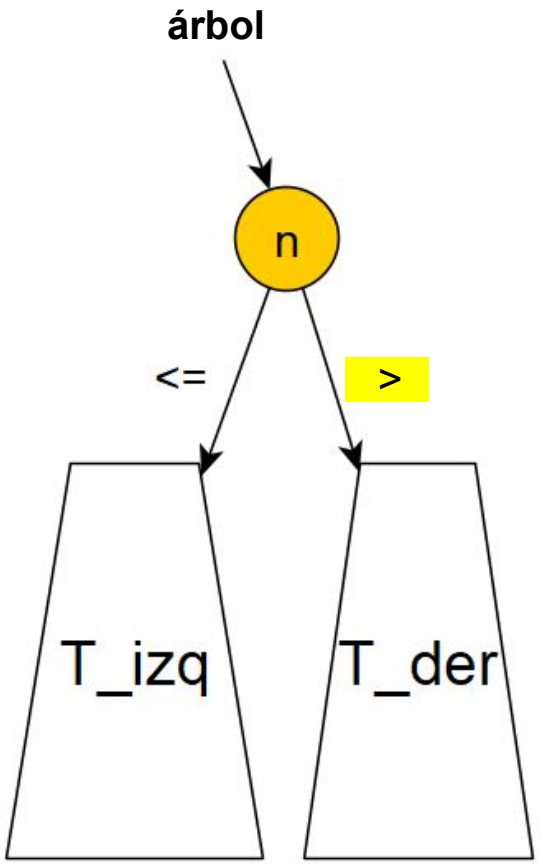
Orden de complejidad: O(n)

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

TDA árbol binario de búsqueda

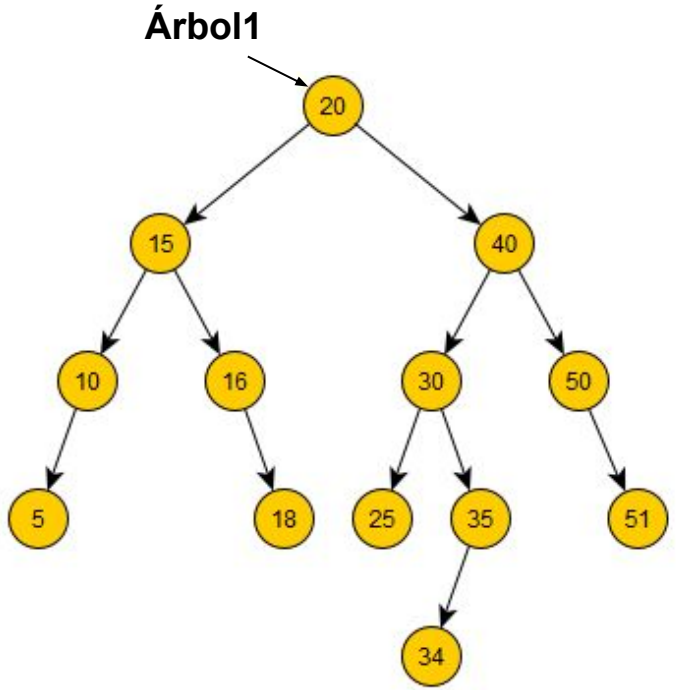
TDA árbol binario de búsqueda

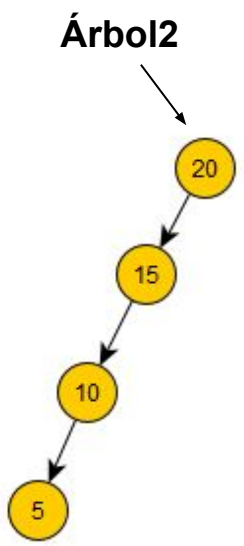
- **Estructura de datos**
  - **Puntero externo**: apunta a la raíz del árbol
  - Cada nodo contiene:
    - dato
    - puntero a hijo izquierdo
    - puntero a hijo derecho
  - Un árbol es ABB ssi:
    - $n_i \leq n, \forall n_i \in T_{izq}\ y\ T_{izq}\ es\ \ ABB$
    - $n < n_i, \forall n_i \in T_{der}\ y\ T_{der}\ es\ \ ABB$

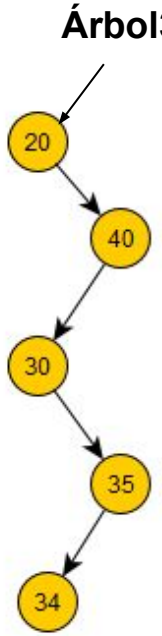


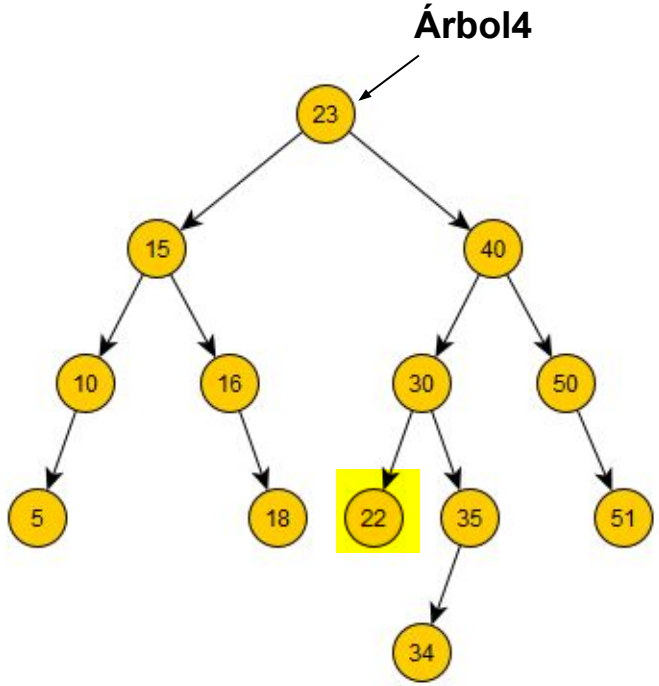
TDA árbol binario de búsqueda

- **¿Cuáles son árboles binarios de búsqueda (ABB)?**



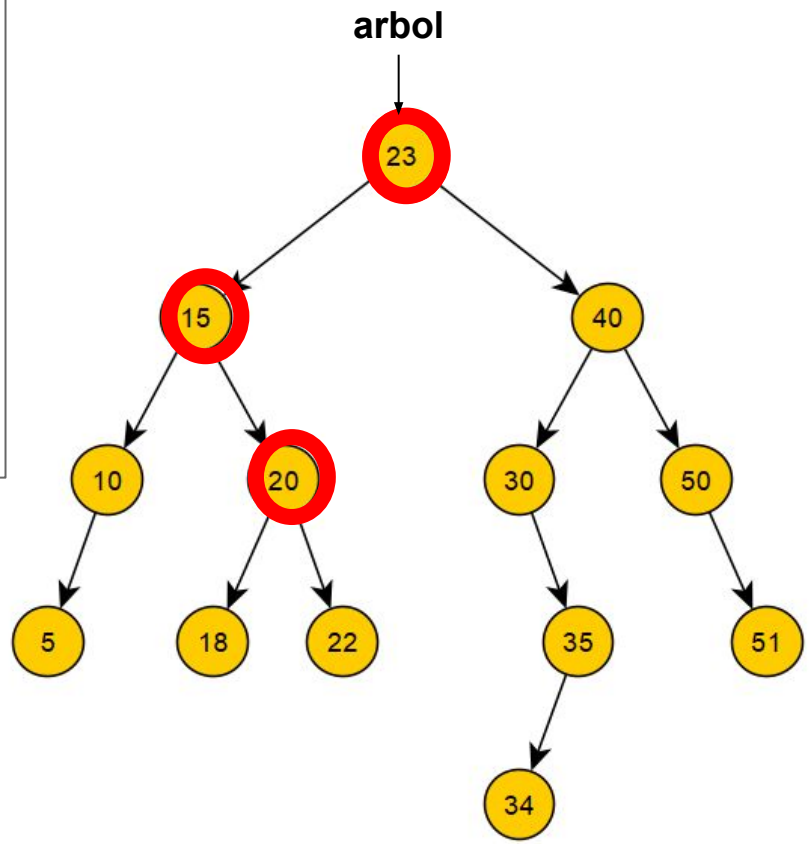






Buscar nodo en ABB

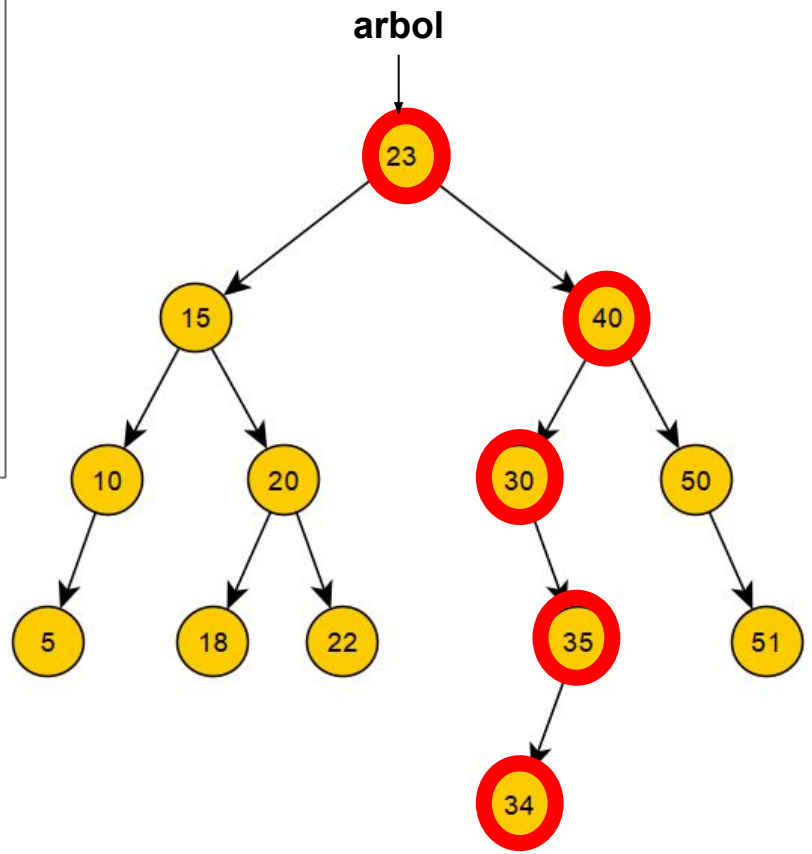
```
buscarNodoABB(arbol,nodoDato): nodo
    si esABBvacio(arbol) entonces
        devolver(NULO)
    sino
        si arbol→dato=nodoDato entonces
            devolver(arbol)
        sino
            si nodoDato<=arbol→dato entonces
                devolver(buscarNodoABB(arbol→izq,nodoDato))
            sino
                devolver(buscarNodoABB(arbol→der,nodoDato))
```



- Ejemplo:
  - buscarNodoABB(arbol,18)

Buscar nodo en ABB

```
buscarNodoABB(arbol,nodoDato): nodo
    si esABBvacio(arbol) entonces
        devolver(NULO)
    sino
        si arbol→dato=nodoDato entonces
            devolver(arbol)
        sino
            si nodoDato<=arbol→dato entonces
                devolver(buscarNodoABB(arbol→izq,nodoDato))
            sino
                devolver(buscarNodoABB(arbol→der,nodoDato))
```



- Ejemplo:
  - ~~buscarNodoABB(arbol,18)~~
  - buscarNodoABB(arbol,33)

Buscar nodo en ABB

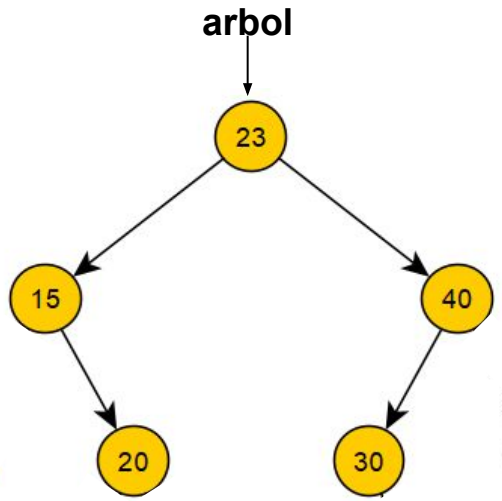
```
buscarNodoABB(arbol,nodoDato): nodo
    si esABBvacio(arbol) entonces
        devolver(NULO)
    sino
        si arbol→dato=nodoDato entonces
            devolver(arbol)
        sino
            si nodoDato<=arbol→dato entonces
                devolver(buscarNodoABB(arbol→izq,nodoDato))
            sino
                devolver(buscarNodoABB(arbol→der,nodoDato))
```

Orden de complejidad: $O(\log_2 n)$

Mejor caso

Insertar nodo en ABB

```
insertarNodoABB(arbol,dato)
    si no(esABBvacio(arbol) entonces
        si dato<=arbol→dato entonces
            insertarNodoABB(arbol→izq,dato)
        sino
            insertarNodoABB(arbol→der,dato)
    sino
        nuevo←crearNodoVacio()
        nuevo→dato←dato
        nuevo→izq←NULO
        nuevo→der←NULO
        arbol←nuevo
```



- Ejemplo:
  - insertarNodoABB(arbol,23)
  - insertarNodoABB(arbol,40)
  - insertarNodoABB(arbol,30)
  - insertarNodoABB(arbol,15)
  - insertarNodoABB(arbol,20)

Insertar nodo en ABB

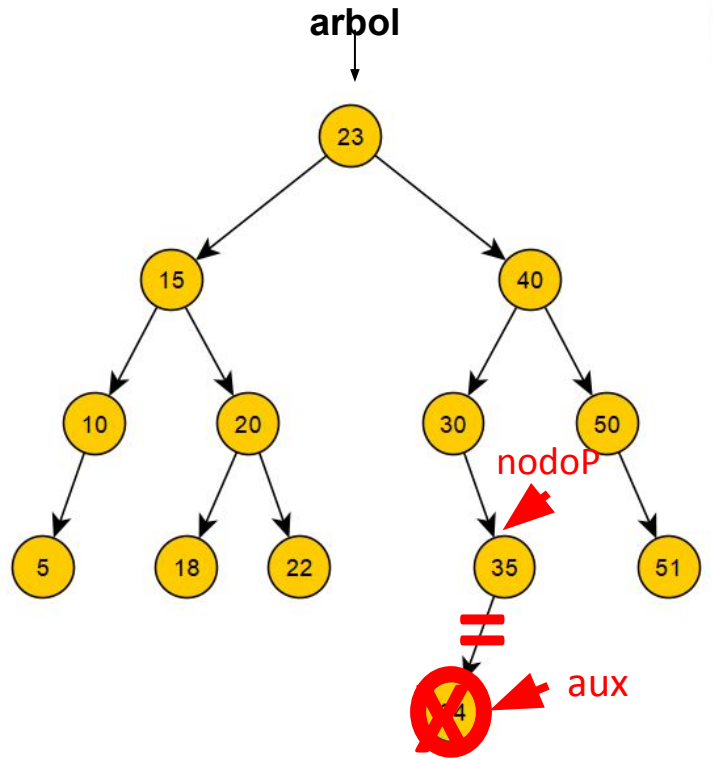
```
insertarNodoABB(arbol,dato)
     si no(esABBvacio(arbol)  entonces
          si dato<=arbol→dato  entonces
               insertarNodoABB(arbol→izq,dato)
          sino
               insertarNodoABB(arbol→der,dato)
     sino
          nuevo←crearNodoVacio()
          nuevo→dato←dato
          nuevo→izq←NULO
          nuevo→der←NULO
          arbol←nuevo
```

Orden de complejidad: $O(\log_2 n)$

Mejor caso

Eliminar nodo en ABB

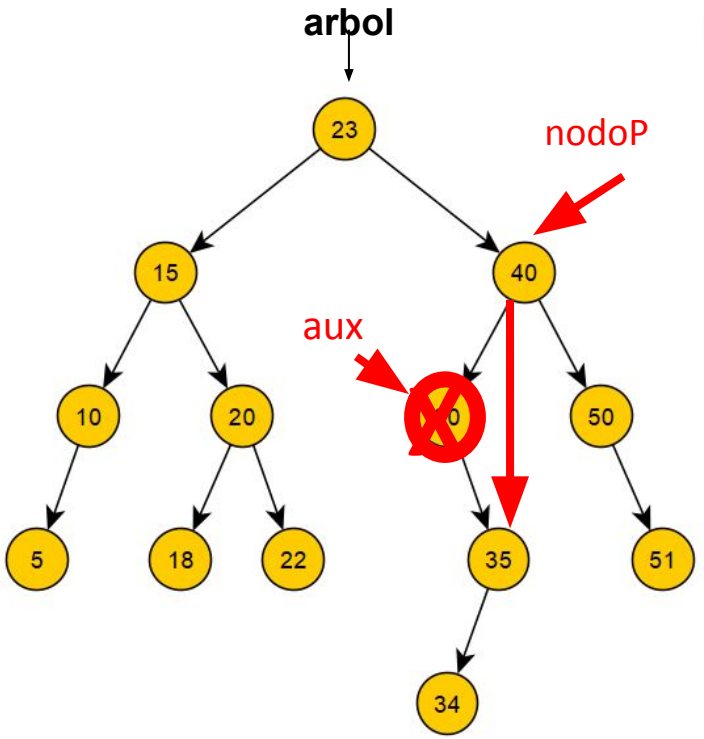
```
eliminarNodoABB(arbol,nodo)
      aux←nodo
      si esHojaABB(arbol,nodo)  entonces
            nodoP←buscarNodoPadreABB(arbol,nodo)
            si nodo→dato <= nodoP→dato  entonces
                  nodoP→izq←NULO
            sino
                  nodoP→der←NULO
            liberar(aux)
      sino
            si nodo→izq=NULO  o nodo→der=NULO  entonces
                  nodoP←buscarNodoPadreABB(arbol,nodo)
                  si nodo→dato <= nodoP→dato  entonces
                        nodoP→izq←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  sino
                        nodoP→der←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  liberar(aux)
            sino
                  aux1←buscarMenor(aux→der) … buscar menor nodo de subárbol derecho
                  intercambiar(aux→dato,aux1→dato)
                  eliminarNodoABB(aux→der,aux1)
```



- Ejemplo:
  - Eliminar nodo con dato 34

Eliminar nodo en ABB

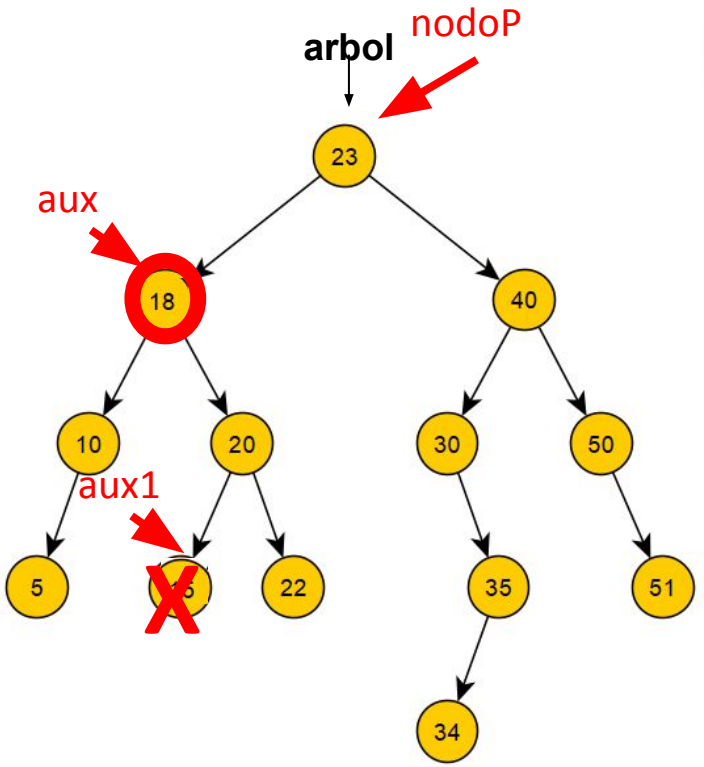
```
eliminarNodoABB(arbol,nodo)
      aux←nodo
      si esHojaABB(arbol,nodo)  entonces
            nodoP←buscarNodoPadreABB(arbol,nodo)
            si nodo→dato <= nodoP→dato  entonces
                  nodoP→izq←NULO
            sino
                  nodoP→der←NULO
            liberar(aux)
      sino
            si nodo→izq=NULO  o nodo→der=NULO  entonces
                  nodoP←buscarNodoPadreABB(arbol,nodo)
                  si nodo→dato <= nodoP→dato  entonces
                        nodoP→izq←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  sino
                        nodoP→der←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  liberar(aux)
            sino
                  aux1←buscarMenor(aux→der) … buscar menor nodo de subárbol derecho
                  intercambiar(aux→dato,aux1→dato)
                  eliminarNodoABB(aux→der,aux1)
```



- Ejemplo:
  - Eliminar nodo con dato 30

Eliminar nodo en ABB

```
eliminarNodoABB(arbol,nodo)
      aux←nodo
      si esHojaABB(arbol,nodo)  entonces
            nodoP←buscarNodoPadreABB(arbol,nodo)
            si nodo→dato <= nodoP→dato  entonces
                  nodoP→izq←NULO
            sino
                  nodoP→der←NULO
            liberar(aux)
      sino
            si nodo→izq=NULO  o nodo→der=NULO  entonces
                  nodoP←buscarNodoPadreABB(arbol,nodo)
                  si nodo→dato <= nodoP→dato  entonces
                        nodoP→izq←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  sino
                        nodoP→der←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  liberar(aux)
            sino
                  aux1←buscarMenor(aux→der) … buscar menor nodo de subárbol derecho
                  intercambiar(aux→dato,aux1→dato)
                  eliminarNodoABB(aux→der,aux1)
```



- Ejemplo:
  - Eliminar nodo con dato 15

Eliminar nodo en ABB

```
eliminarNodoABB(arbol,nodo)
      aux←nodo
      si esHojaABB(arbol,nodo)  entonces
            nodoP←buscarNodoPadreABB(arbol,nodo)
            si nodo→dato <= nodoP→dato  entonces
                  nodoP→izq←NULO
            sino
                  nodoP→der←NULO
            liberar(aux)
      sino
            si nodo→izq=NULO  o nodo→der=NULO  entonces
                  nodoP←buscarNodoPadreABB(arbol,nodo)
                  si nodo→dato <= nodoP→dato  entonces
                        nodoP→izq←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  sino
                        nodoP→der←aux→[izq|der] … depende de cuál sea el hijo NO nulo
                  liberar(aux)
            sino
                  aux1←buscarMenor(aux→der) … buscar menor nodo de subárbol derecho
                  intercambiar(aux→dato,aux1→dato)
                  eliminarNodoABB(aux→der,aux1)
```

Orden de complejidad: $O(\log_2 n)$

Mejor caso

Ejercicios AB y ABB

- **Ejercicio1**: Escribir un algoritmo que muestre todas las hojas de un árbol binario. El algoritmo debe retornar la cantidad de hojas del árbol binario. Cuál es la complejidad del algoritmo propuesto?

- **Ejercicio2**: Escribir un algoritmo que, dado un árbol binario, indique si es o no un árbol binario de búsqueda. Cuál es la complejidad del algoritmo propuesto?

TDA árbol binario de búsqueda

- **¿Por qué usar ABB en lugar de AB?**
  - Simplifica las operaciones sobre árboles, respecto a AB
  - Complejidad del caso medio
    - buscarNodoABB: $O(\log_2 n)$
    - insertarNodoABB: $O(\log_2 n)$
    - eliminarNodoABB: $O(\log_2 n)$
  - Complejidad del peor caso
    - buscarNodoABB: $O(n)$
    - insertarNodoABB: $O(n)$
    - eliminarNodoABB: $O(n)$

¿Cuándo se da este caso?

Actividad de cierre

código: 7536 3552

Próximas fechas...

- Resumen de la semana:
  - TDA árbol
  - TDA árbol binario
  - TDA árbol binario de búsqueda

  ~~cátedra~~ – refuerzo – laboratorio

- Próxima semana:
  - TDA árbol AVL

U4 - S11