

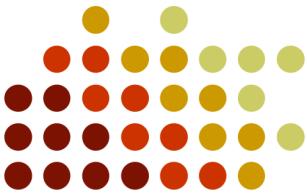
PROGRAMACION 3

TUDAI

Tema 2: Árboles

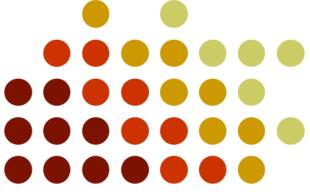
Federico Améndola y Sebastián Vallejos

Fac. Ciencias Exactas. UNICEN



Árboles





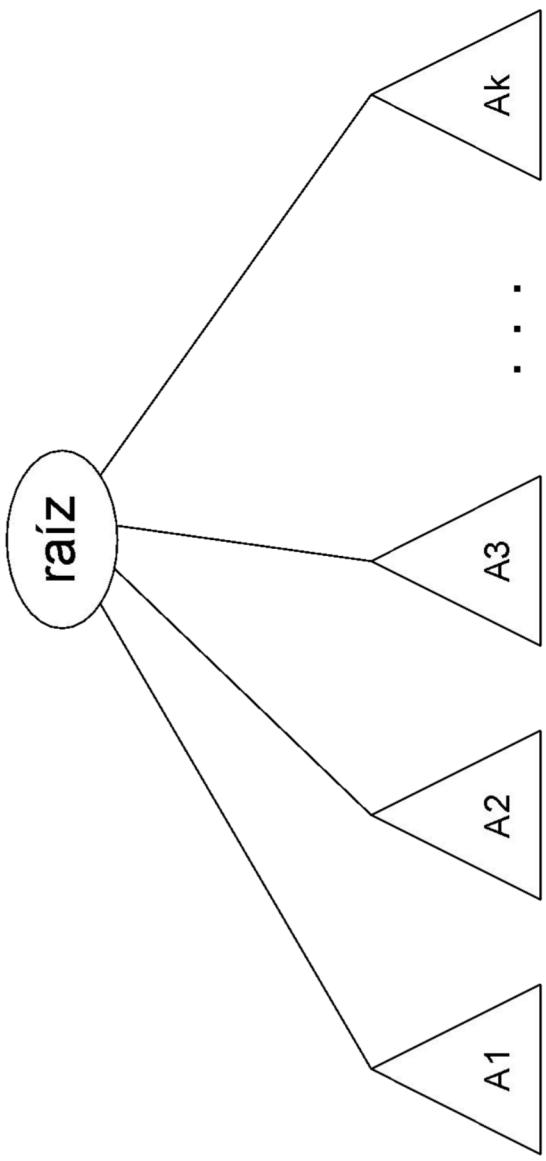
Árboles



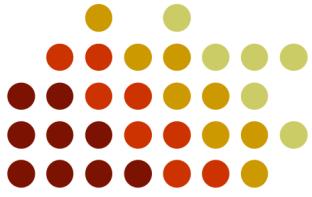
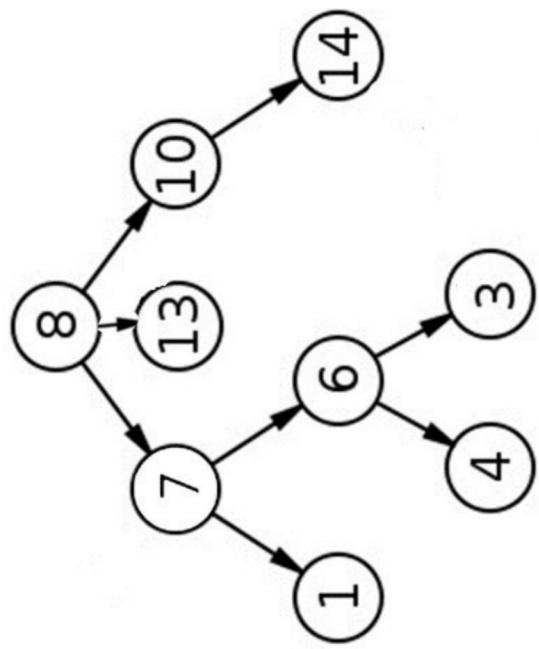
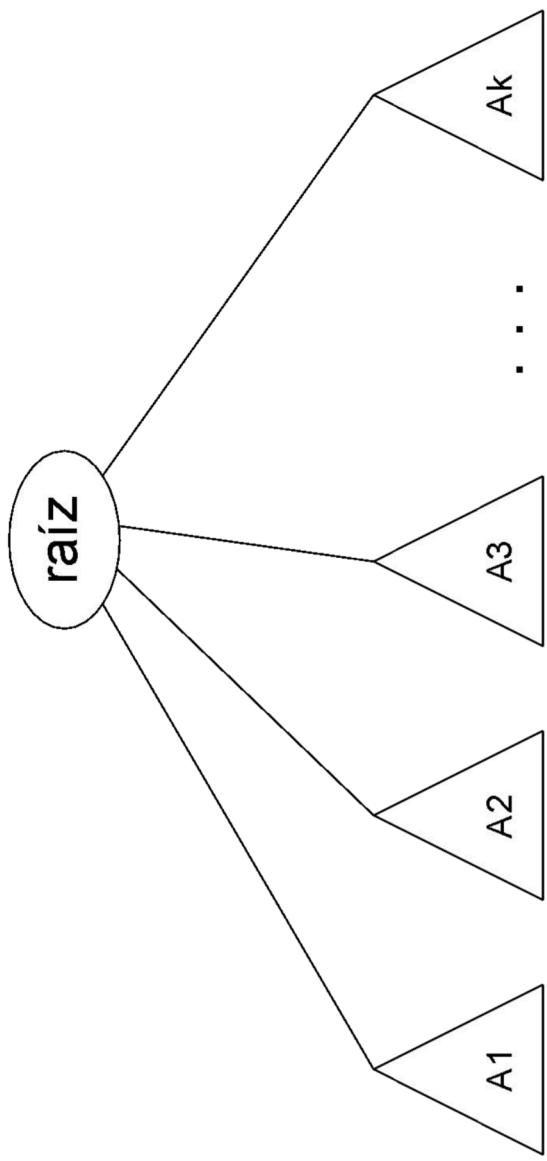
Árboles

Definición general

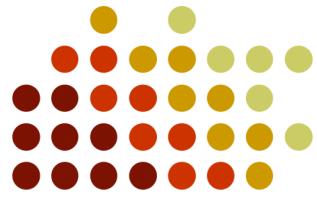
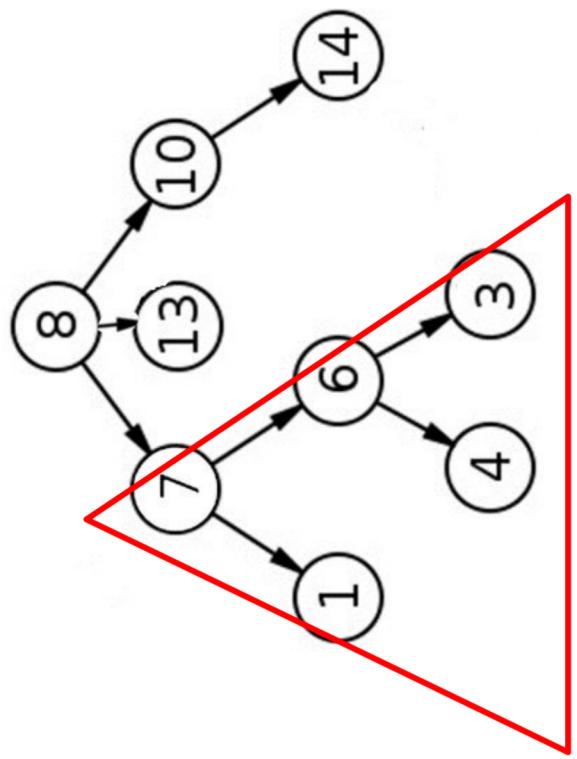
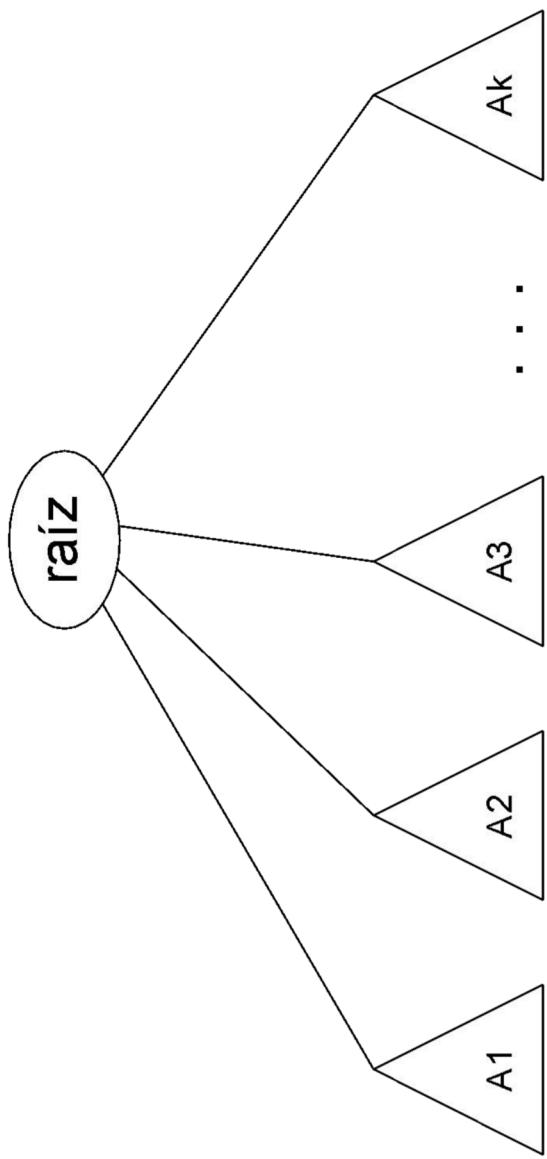
- Si no está vacío, es un conjunto de uno o más nodos, tal que existe un nodo especial llamado **raíz** (r) y donde los restantes nodos están separados en $k \geq 0$ conjuntos disjuntos, cada uno de los cuales es a su vez un árbol (**subárboles**) A_1, A_2, \dots, A_k . Para cada subárbol, su raíz está conectada a la raíz r por medio de un **arco**. (es una definición recursiva).
- El árbol puede ser vacío.
- Entonces un árbol no vacío tiene una raíz, y un conjunto de árboles.



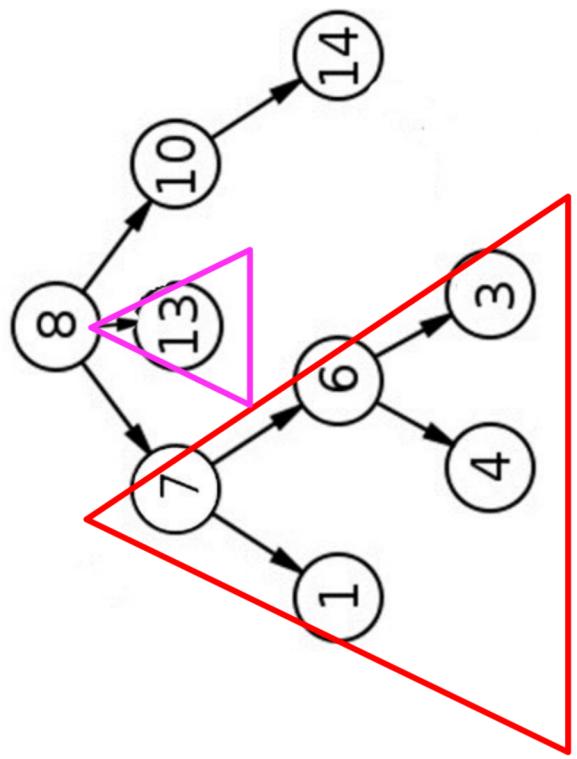
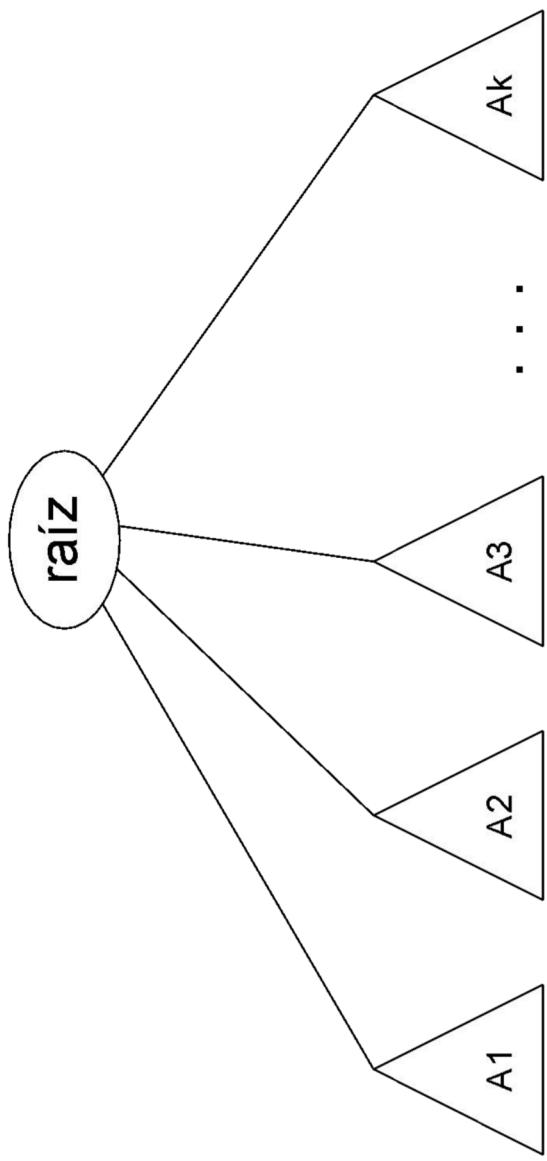
Árbol y subárboles



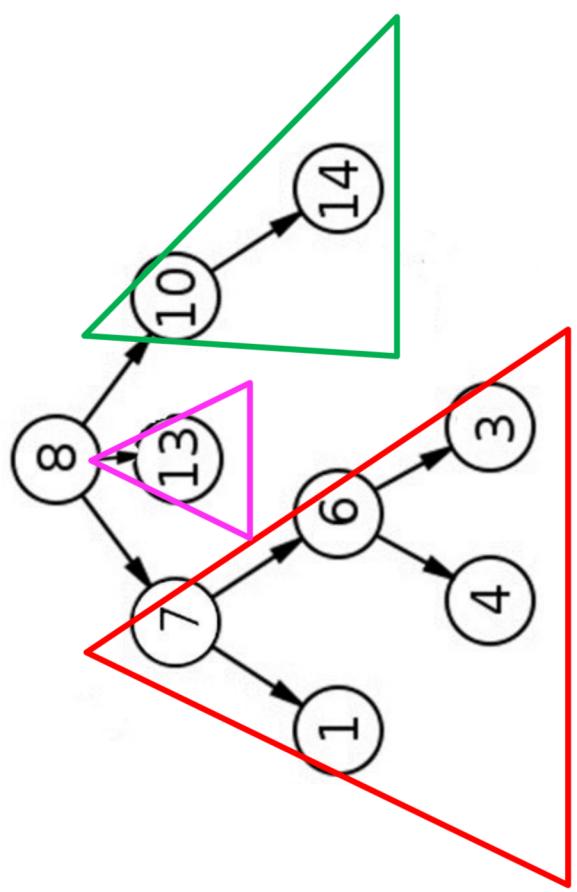
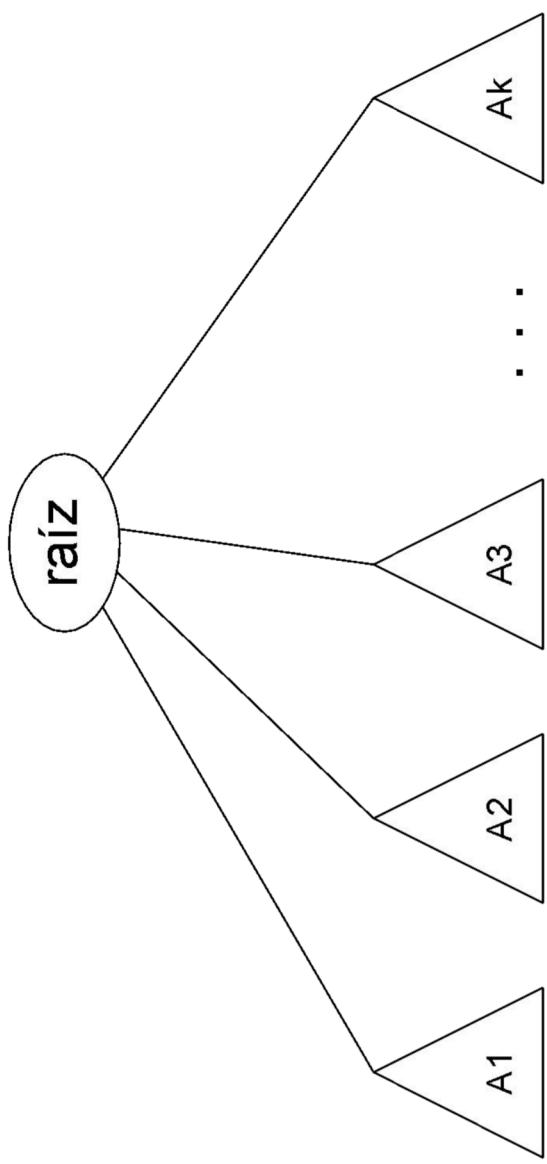
Árbol y subárboles



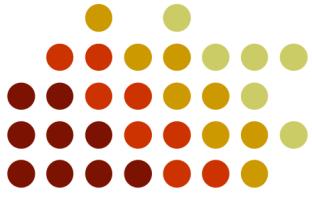
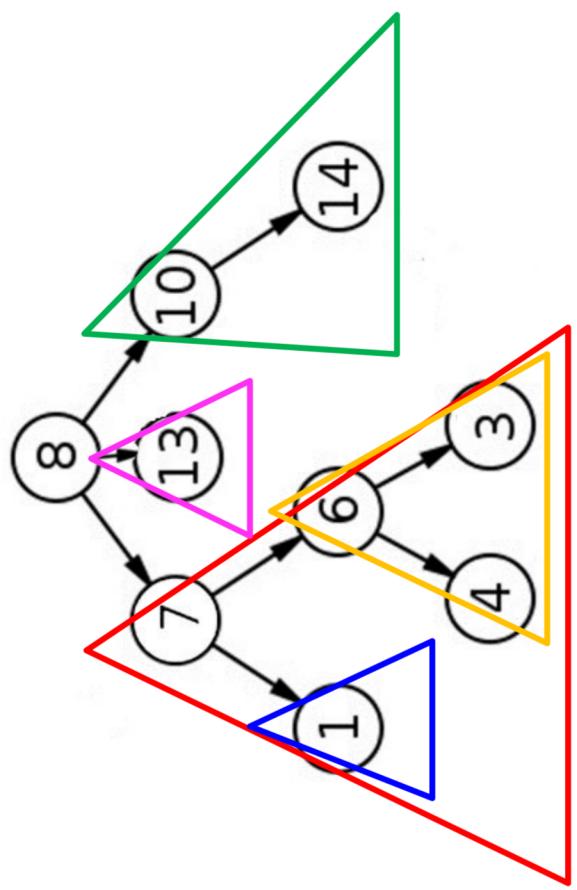
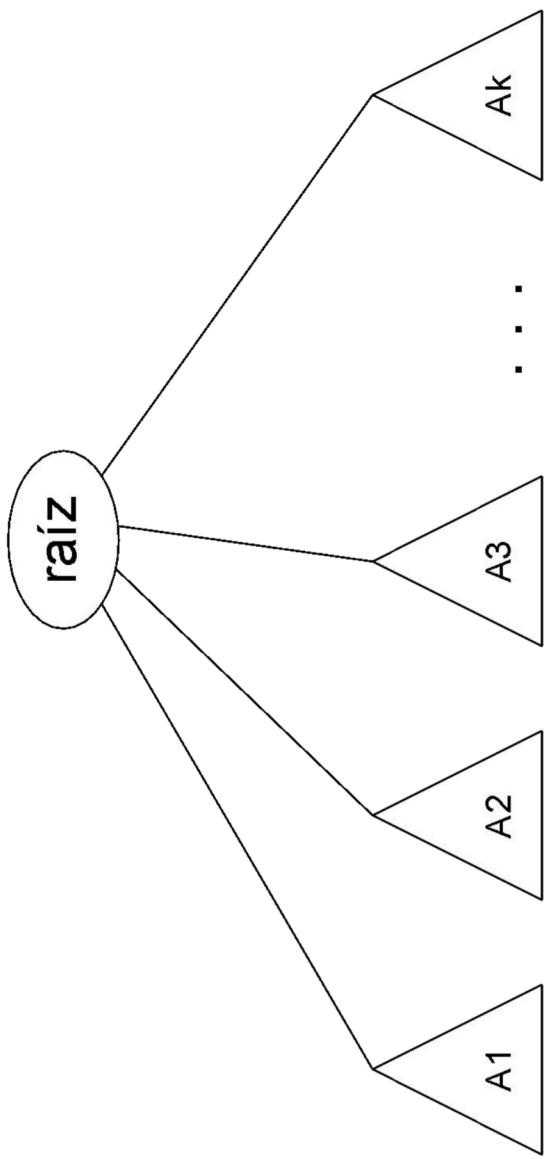
Árbol y subárboles



Árbol y subárboles



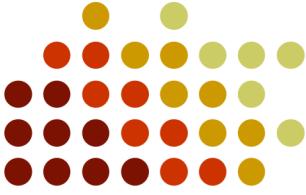
Árbol y subárboles



Terminología

- 
-
- *Hijo y Padre*
 - Cada nodo, excepto la raíz, tiene **un** parent.
 - Cada nodo puede tener un número arbitrario de hijos (dependerá del tipo de árbol).
 - *Hojas (ó nodos externos)*
 - Nodos sin hijos.
 - *Vecino o hermano*
 - Nodos con el mismo parent.

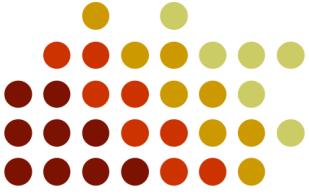
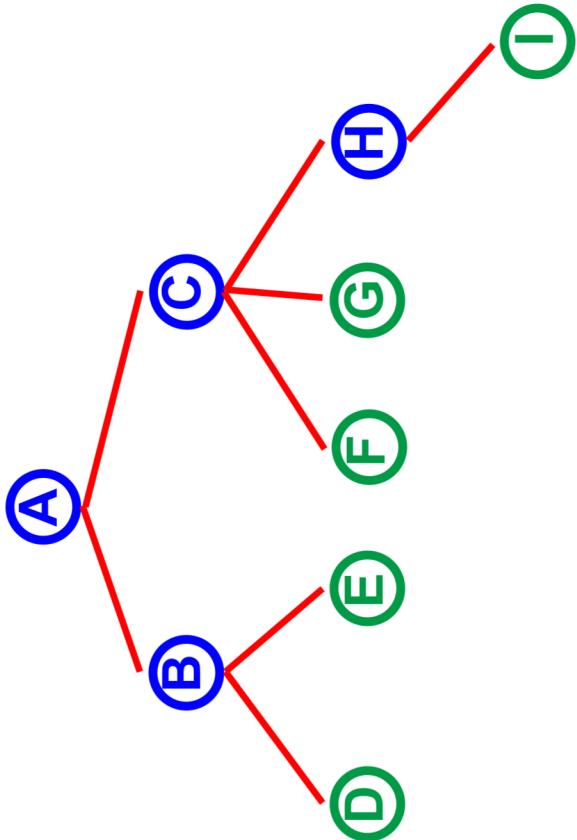
Terminología



- **Camino**
 - Conjunto de arcos desde un nodo padre a cualquier nodo de alguno de sus subarboles.
- **Ancestros y descendientes**
 - Nodos que anteceden en un camino y nodos que suceden.
- **Longitud de un camino**
 - Cantidad de arcos de un camino.
- **Profundidad de un nodo**
 - Longitud del único camino desde la raíz a ese nodo.
- **Altura de un árbol (h)**:
 - La profundidad de la más lejana de sus hojas.

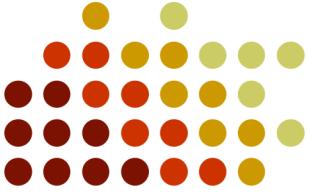
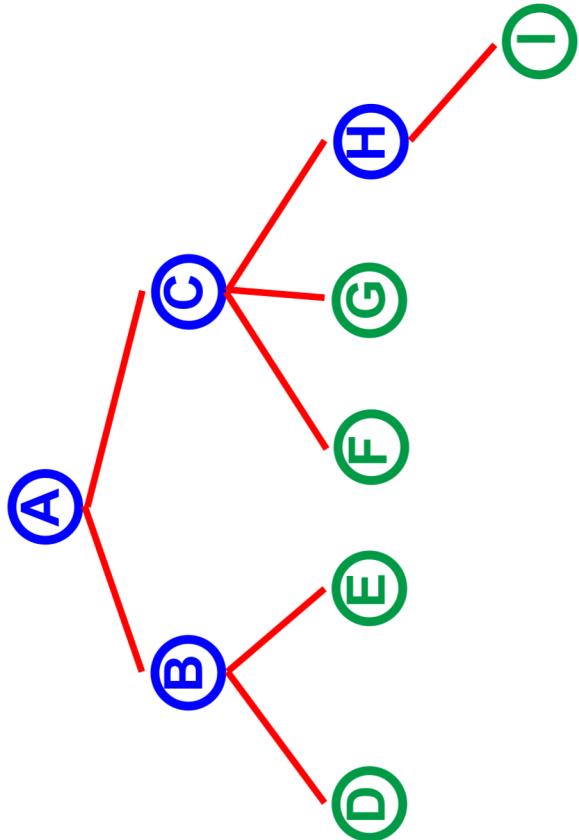
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?



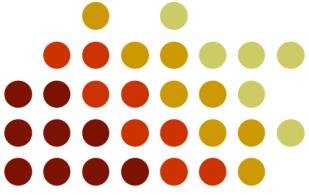
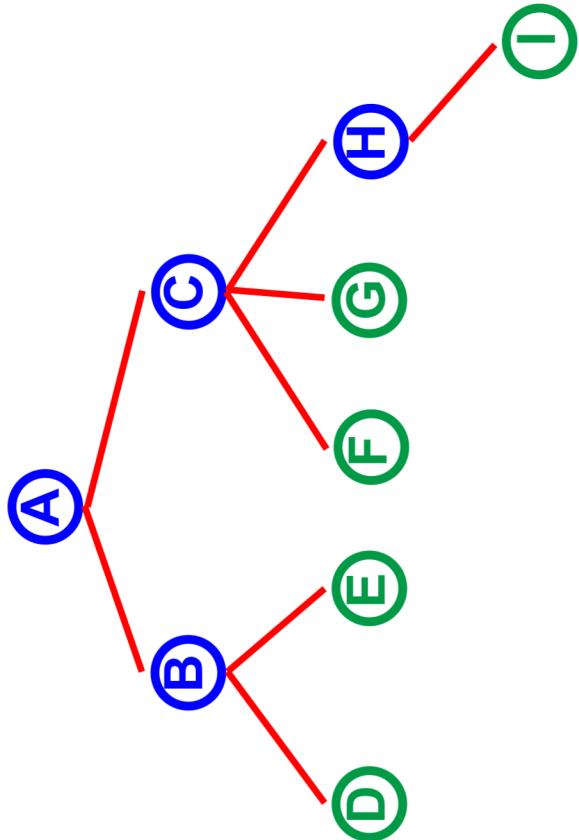
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E



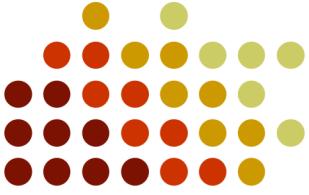
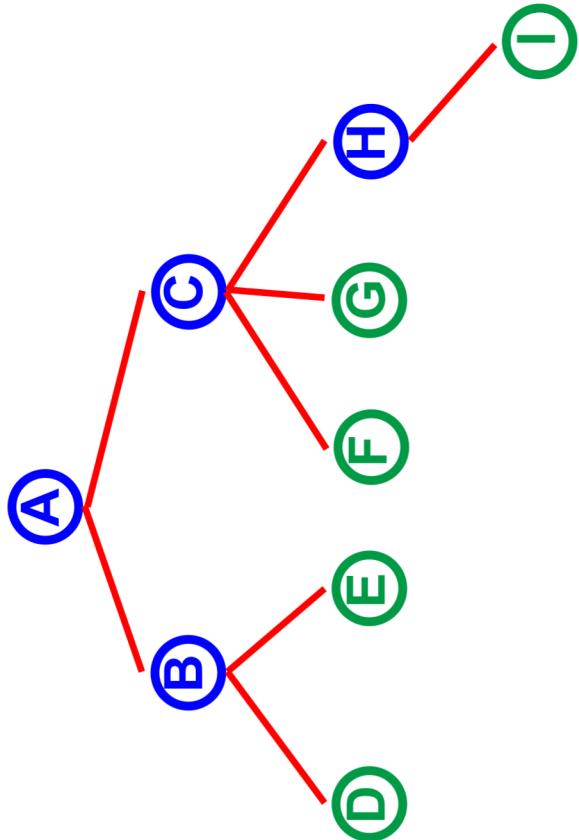
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B



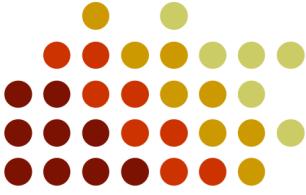
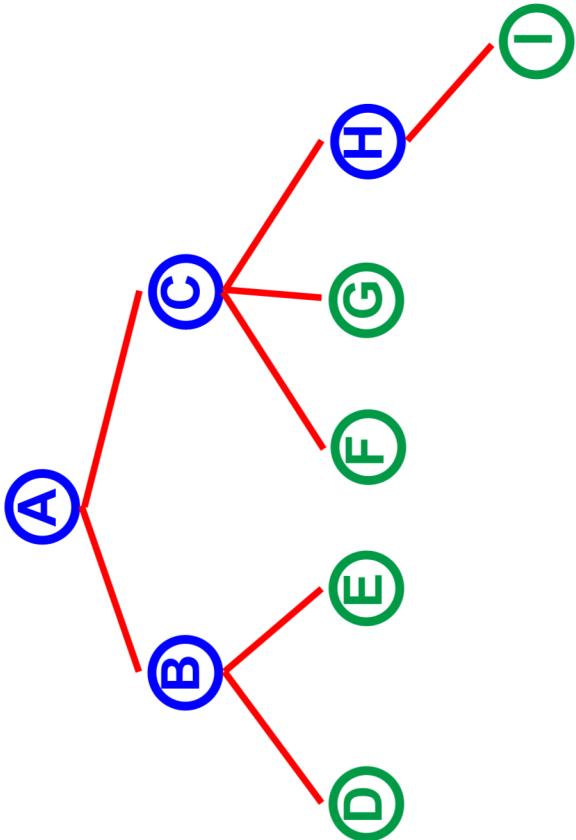
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano o vecino** de B



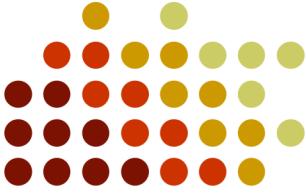
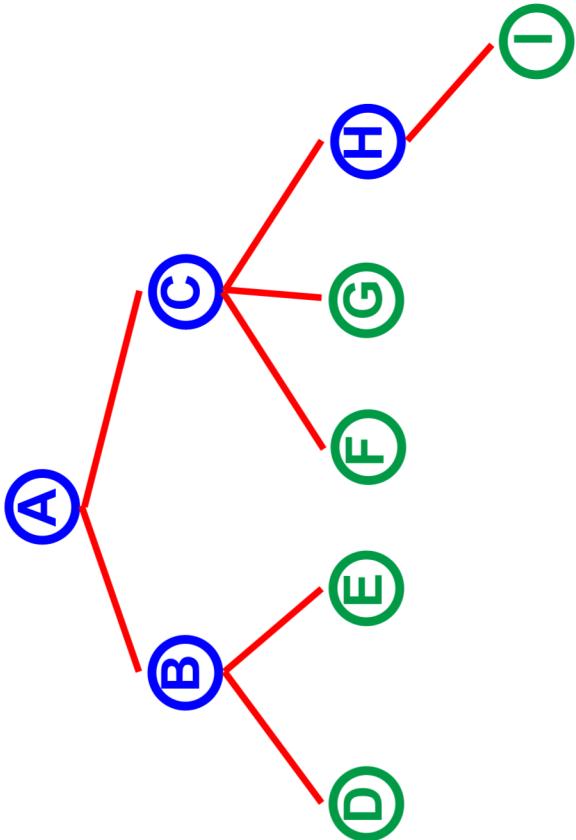
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**



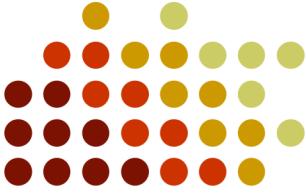
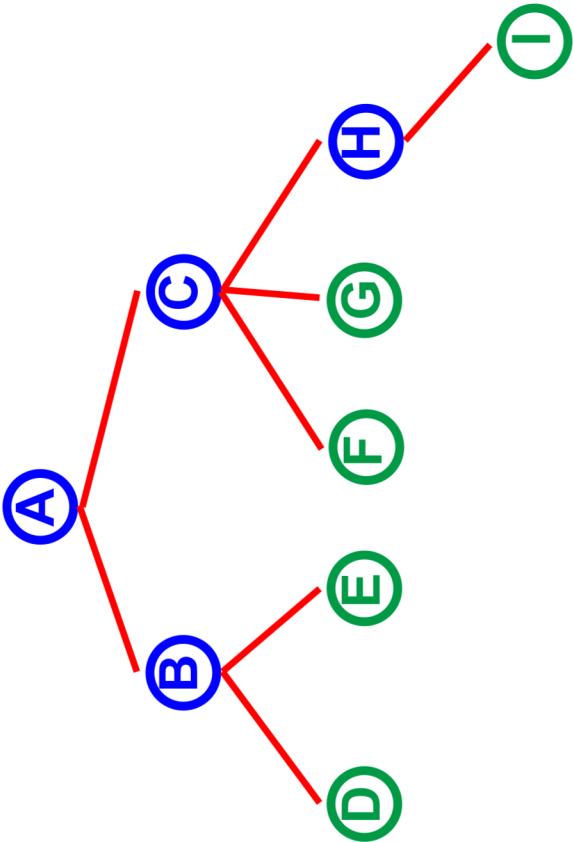
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**



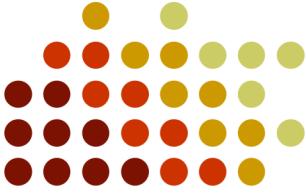
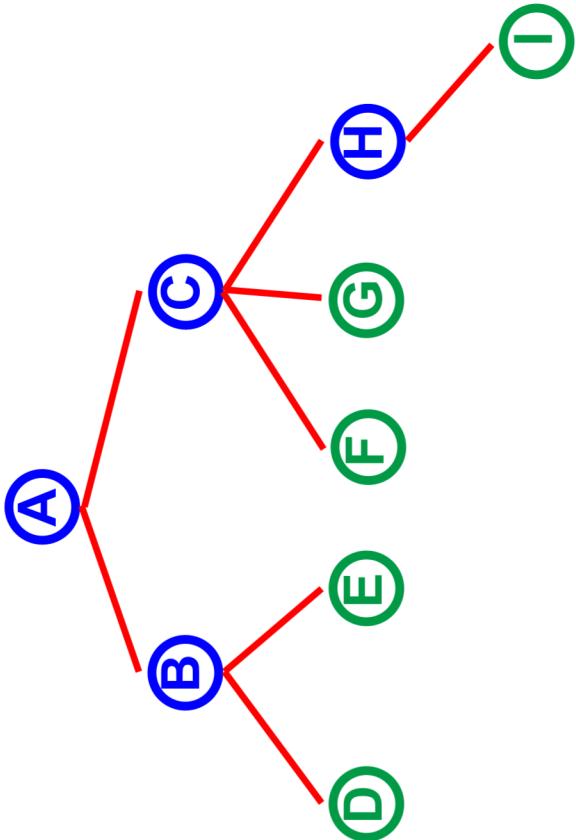
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**
- La **profundidad (nivel)** de E es 2



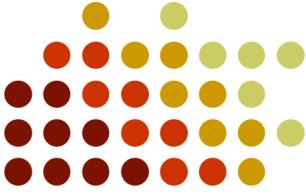
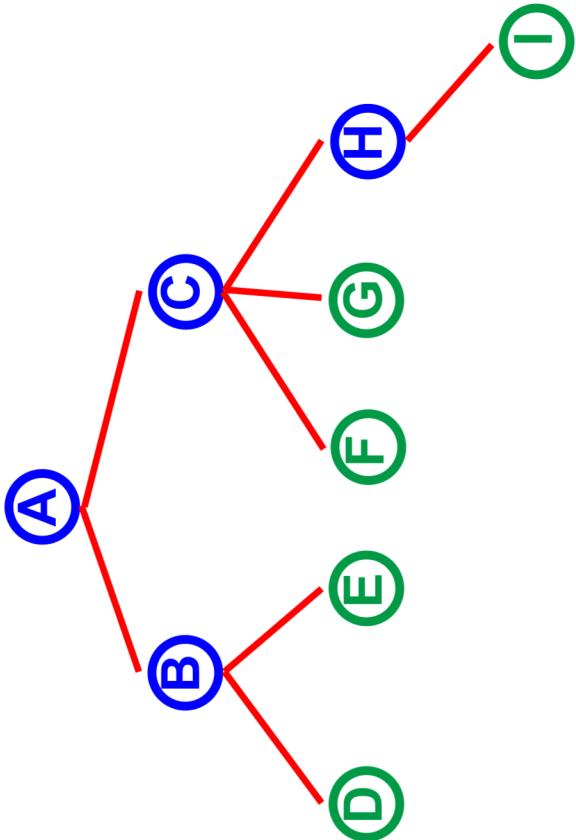
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**
- La **profundidad (nivel)** de E es 2
- La **altura** del árbol es 3 (= profundidad de I)



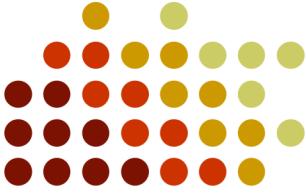
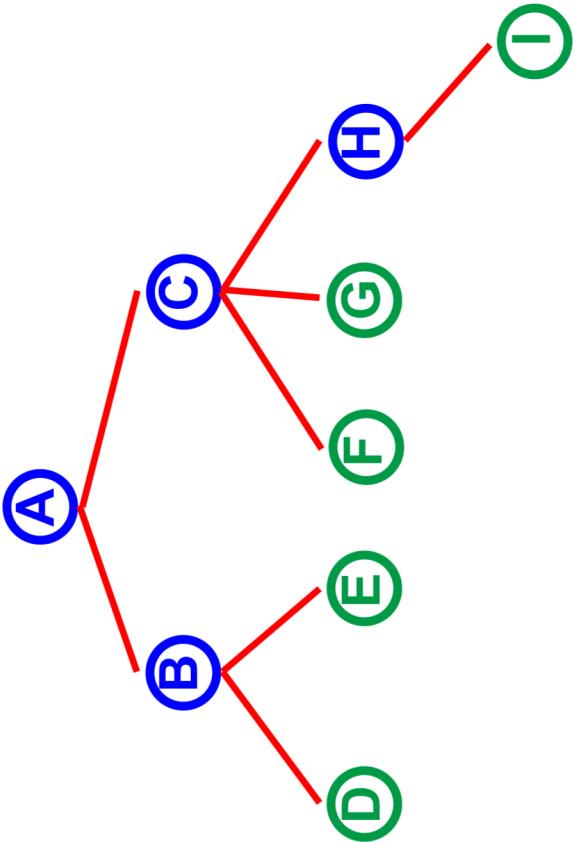
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**
- La **profundidad (nivel)** de E es 2
- La **altura** del árbol es 3 (= profundidad de I)
- El **camino** entre C e I es de **longitud** 2



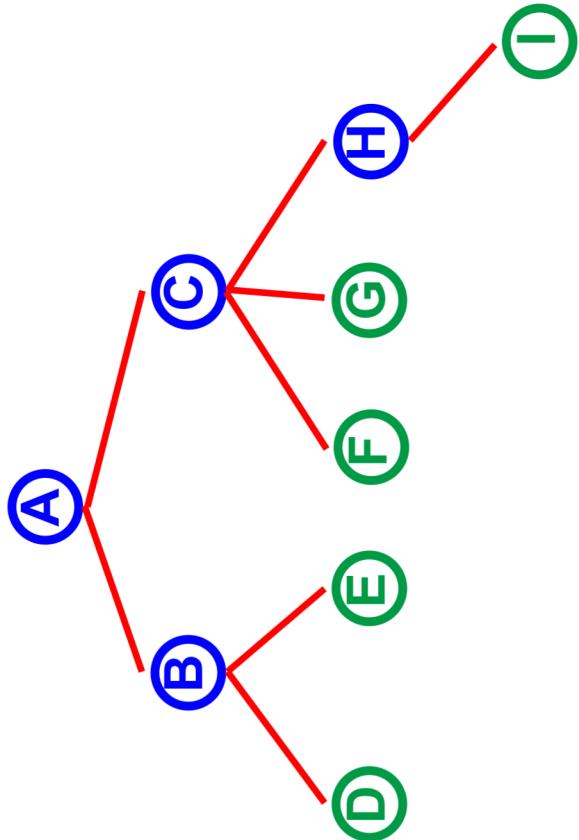
Terminología - Ejemplo

- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**
- La **profundidad (nivel)** de E es 2
- La **altura** del árbol es 3 (= profundidad de I)
- El **caminio** entre C e I es de **longitud** 2
- A y C son **ancestros** de G



Terminología - Ejemplo

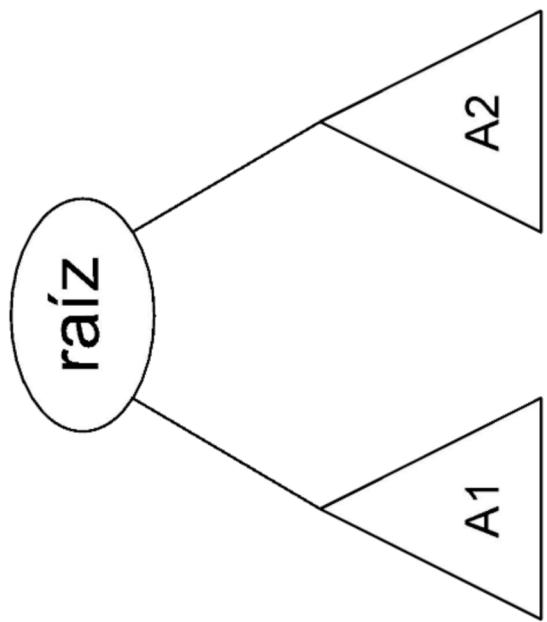
- A es el nodo **raíz**. ¿Qué profundidad tendrá?
- B es el **padre** de D y E
- D y E son los **hijos** de B
- C es el **hermano** o **vecino** de B
- D, E, F, G, I son **nodos externos** u **hojas**
- A, B, C, H son **nodos internos**
- La **profundidad (nivel)** de E es 2
- La **altura** del árbol es 3 (= profundidad de I)
- El **camino** entre C e I es de **longitud** 2
- A y C son **ancestros** de G



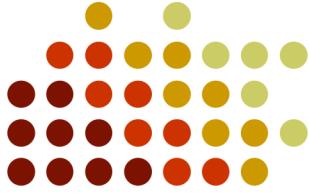
- Propiedad: $(\# \text{arcos}) = (\#\text{nodos}) - 1$

Árboles Binarios

- Es un árbol cuyos nodos no tienen más de 2 hijos

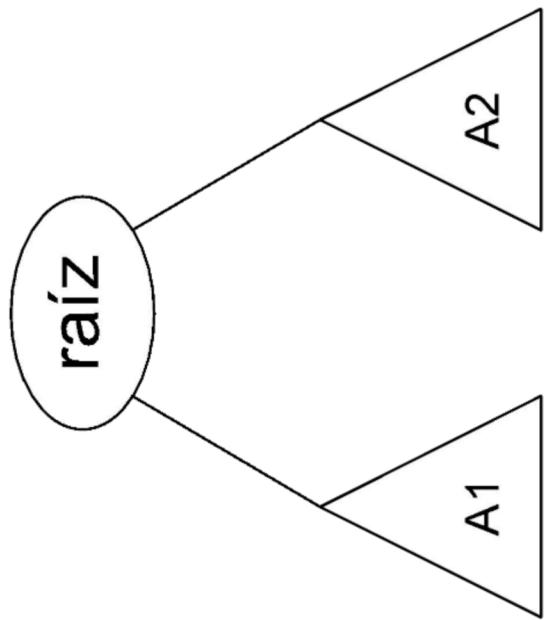


- A1 y A2 son árboles binarios también



Árboles Binarios

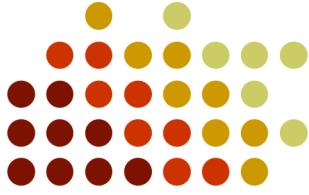
- Es un árbol cuyos nodos no tienen más de 2 hijos



```
class Node {  
  
    Integer key;  
    Node left;  
    Node right;
```

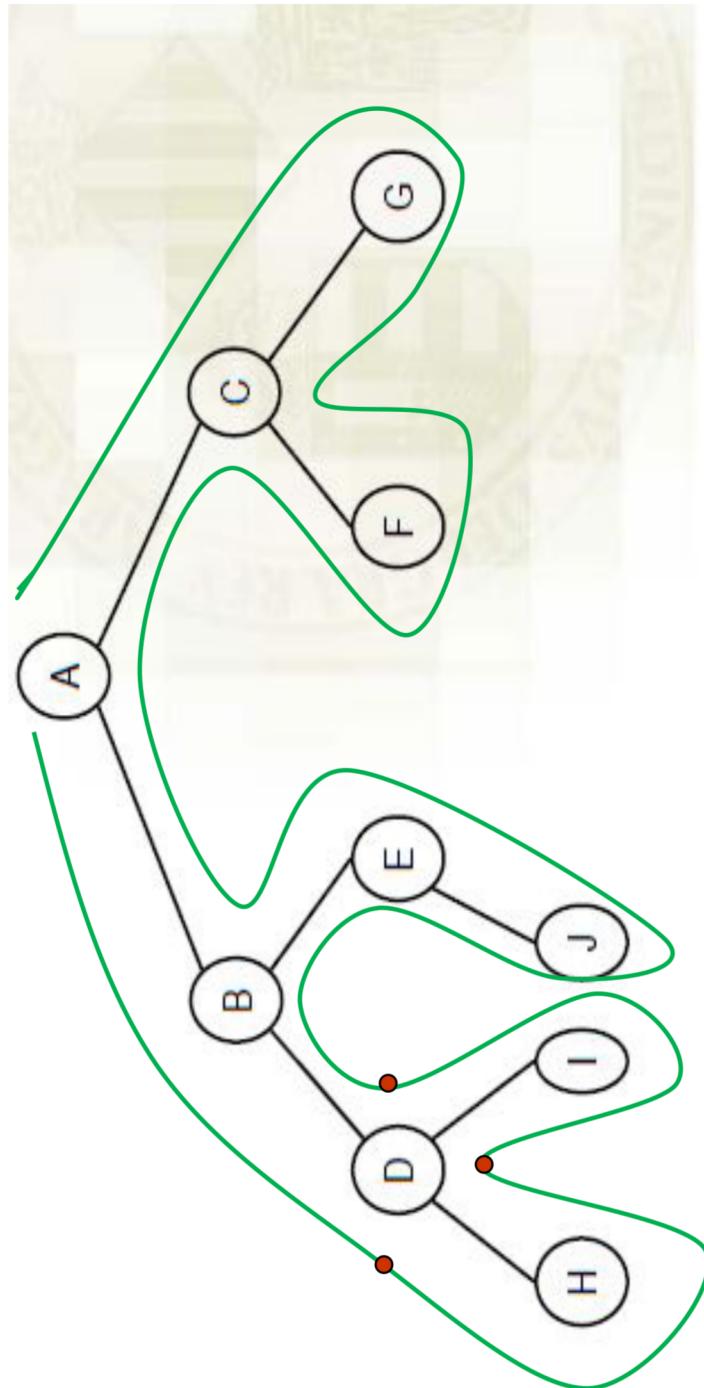
```
public Node(Integer key) {  
    this.key = key;  
    this.left = null;  
    this.right = null;  
}  
}
```

- A1 y A2 son árboles binarios también



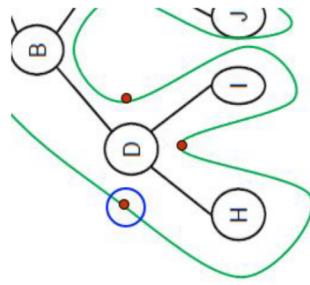
Recorridos de Árboles Binarios

- El recorrido más genérico de un árbol binario visita cada nodo tres veces (desde la izquierda, desde abajo y desde la derecha). Los puntos rojos indican estas visitas para el nodo D.
 - Se denomina recorrido de **Euler**.



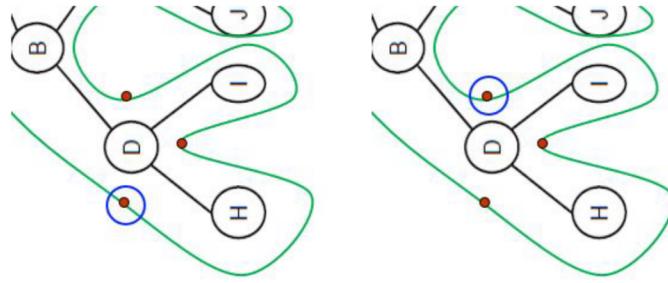
Recorridos de Árboles Binarios

- Usado para obtener los contenidos de los nodos en un orden determinado
- Son casos especiales del recorrido de Euler.
- **Recorrido Pre-orden**
 - Imprimir el dato de la raíz
 - Recursivamente obtener los datos del subárbol izquierdo
 - Recursivamente obtener los datos del subárbol derecho



Recorridos de Árboles Binarios

- Usado para obtener los contenidos de los nodos en un orden determinado
- Son casos especiales del recorrido de Euler.
- **Recorrido Pre-orden**
 - Imprimir el dato de la raíz
 - Recursivamente obtener los datos del subárbol izquierdo
 - Recursivamente obtener los datos del subárbol derecho
- **Recorrido Post-orden**
 - Recursivamente obtener los datos del subárbol izquierdo
 - Recursivamente obtener los datos del subárbol derecho
 - Imprimir el dato de la raíz

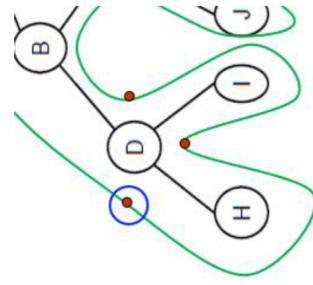


Recorridos de Árboles Binarios

- Usado para obtener los contenidos de los nodos en un orden determinado
- Son casos especiales del recorrido de Euler.

Recorrido Pre-orden

- Imprimir el dato de la raíz
- Recursivamente obtener los datos del subárbol izquierdo
- Recursivamente obtener los datos del subárbol derecho

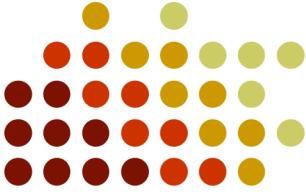
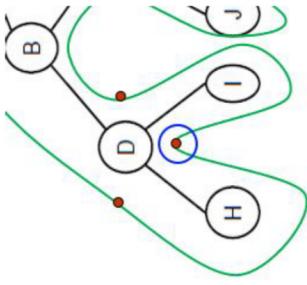


Recorrido Post-orden

- Recursivamente obtener los datos del subárbol izquierdo
- Recursivamente obtener los datos del subárbol derecho
- Imprimir el dato de la raíz

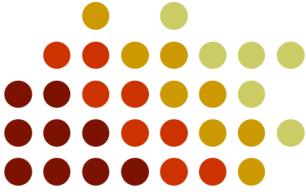
Recorrido En-orden

- Recursivamente obtener los datos del subárbol izquierdo
- Imprimir el dato de la raíz
- Recursivamente obtener los datos del subárbol derecho



Recorridos de Árboles Binarios

```
void printPreorder(Node node) {  
    if (node == null)  
        return;  
  
    System.out.print(node.key + " ");  
  
    printPreorder(node.left);  
  
    printPreorder(node.right);  
}  
  
void printInorder(Node node) {  
    if (node == null)  
        return;  
  
    printInorder(node.left);  
  
    System.out.print(node.key + " ");  
  
    printInorder(node.right);  
}  
  
void printPostorder(Node node) {  
    if (node == null)  
        return;  
  
    printPostorder(node.left);  
  
    printPostorder(node.right);  
  
    System.out.print(node.key + " ");  
}
```

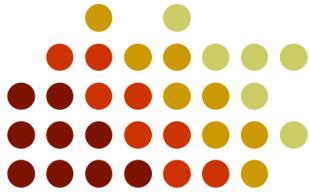
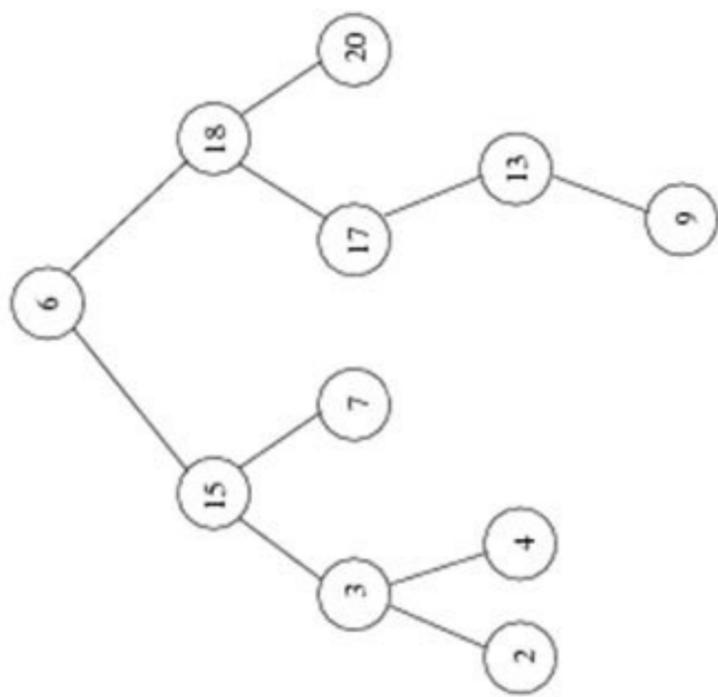


Recorridos de Árboles Binarios

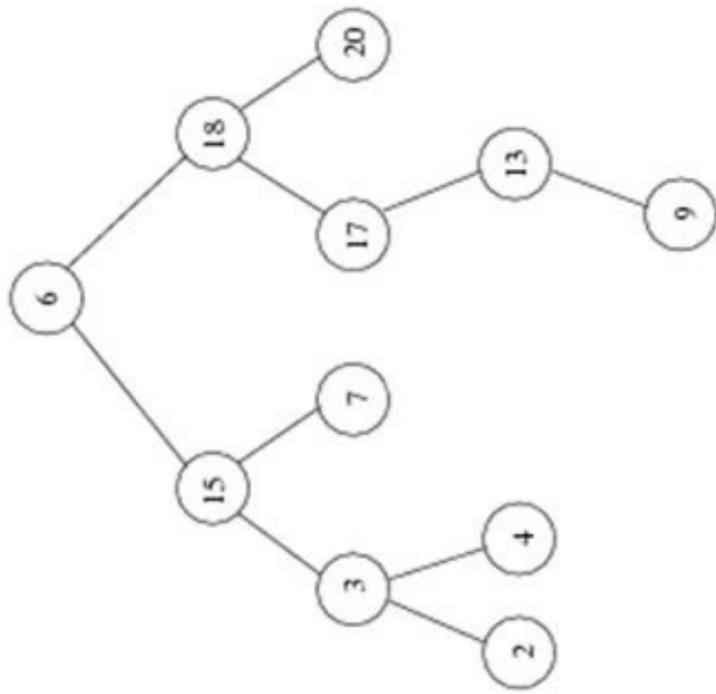
```
void printPreorder(Node node) {  
    if (node == null)  
        return;  
  
    System.out.print(node.key + " ");  
  
    printPreorder(node.left);  
  
    printPreorder(node.right);  
}  
  
void printInorder(Node node) {  
    if (node == null)  
        return;  
  
    printInorder(node.left);  
  
    System.out.print(node.key + " ");  
  
    printInorder(node.right);  
}  
  
void printPostorder(Node node) {  
    if (node == null)  
        return;  
  
    printPostorder(node.left);  
  
    printPostorder(node.right);  
  
    System.out.print(node.key + " ");  
}
```

O(n)

Recorridos de los árboles

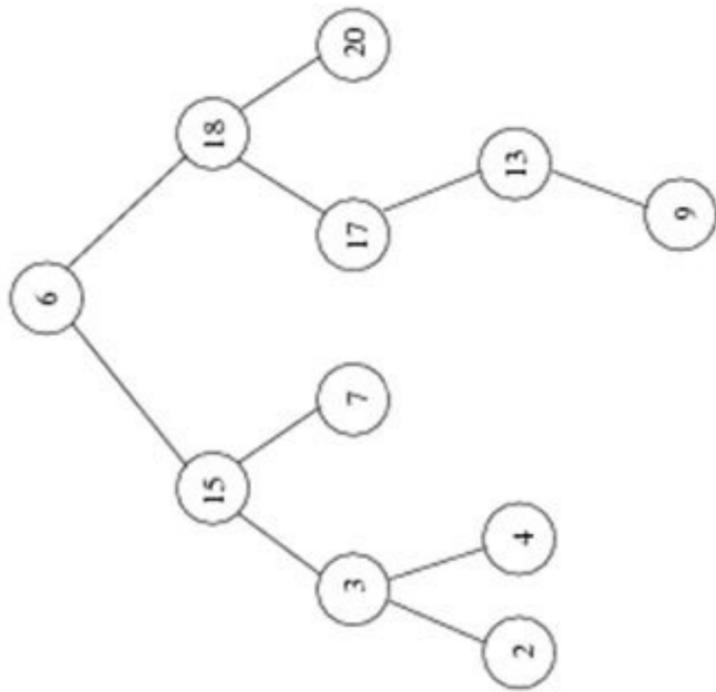


Recorridos de los árboles

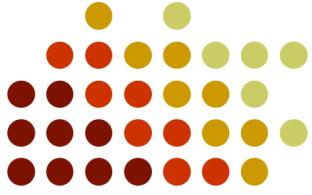


- Pre-orden **6, 15, 3, 2, 4, 7, 18, 17, 13, 9, 20**

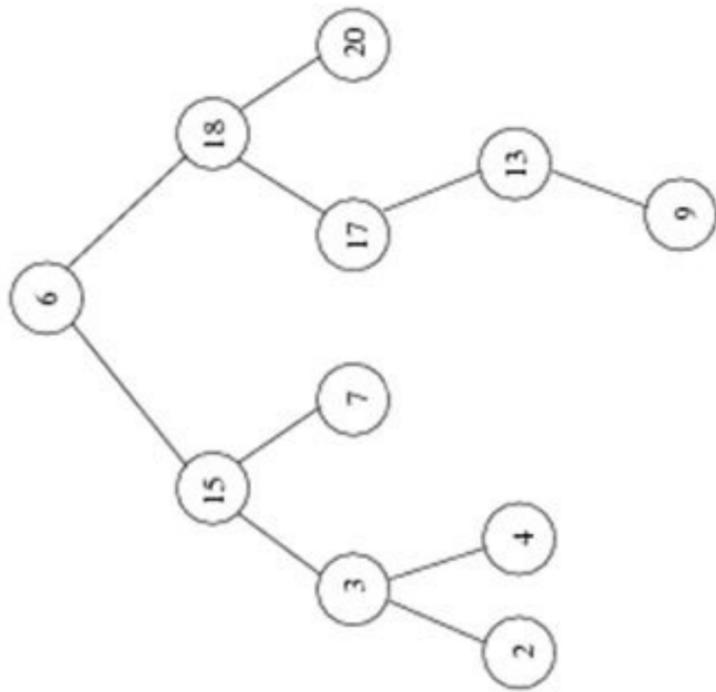
Recorridos de los árboles



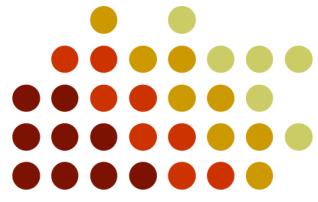
- Pre-orden 6, 15, 3, 2, 4, 7, 18, 17, 13, 9, 20
- Post-orden 2, 4, 3, 7, 15, 9, 13, 17, 20, 18, 6



Recorridos de los árboles

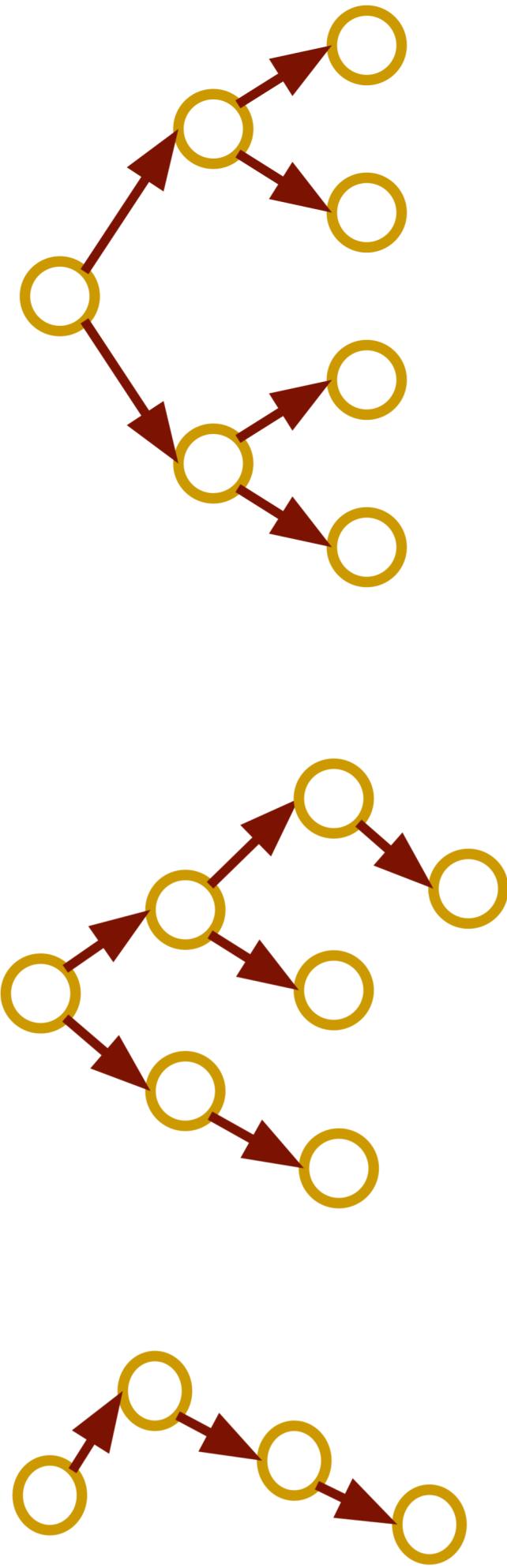


- Pre-orden 6, 15, 3, 2, 4, 7, 18, 17, 13, 9, 20
- Post-orden 2, 4, 3, 7, 15, 9, 13, 17, 20, 18, 6
- En-orden 2, 3, 4, 15, 7, 6, 17, 9, 13, 18, 20



Estructuras Especiales de Árboles Binarios

- **Degenerados / Enredaderas** → Nodos internos con solo un hijo.
- **Balanceados** → Las alturas de los dos subárboles de cualquier nodo difieren a lo sumo en 1.
- **Completos** → Todos los nodos internos tienen 2 hijos.



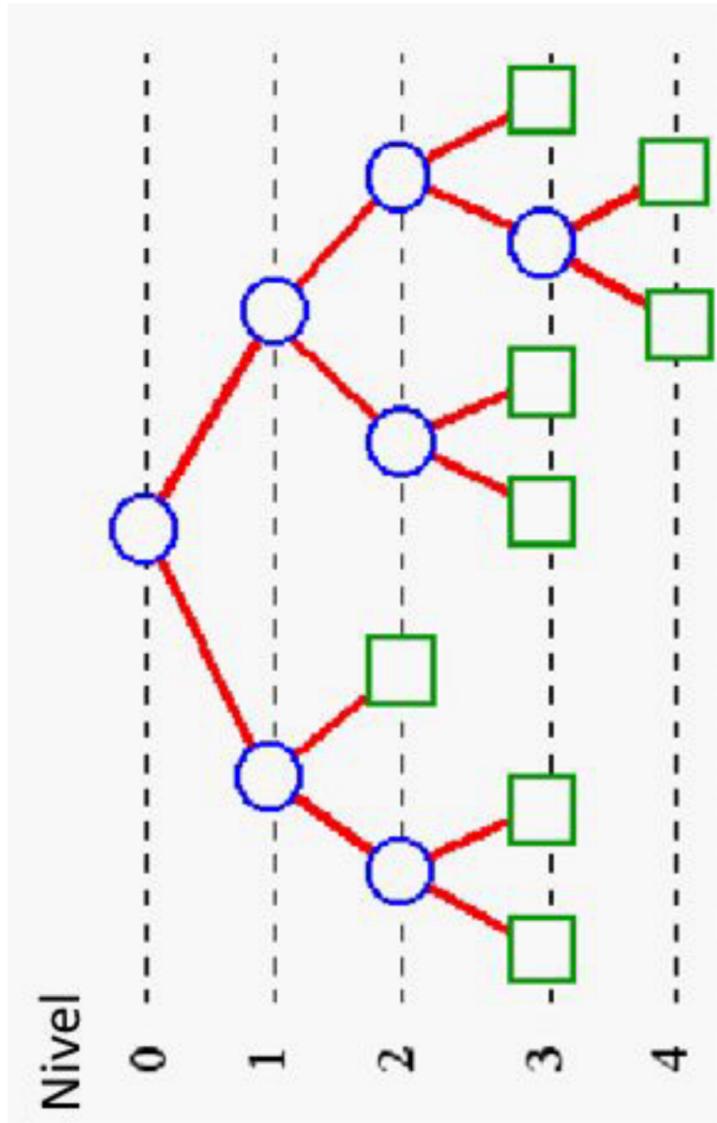
Enredadera
(se asemeja a una lista)

Balanceado

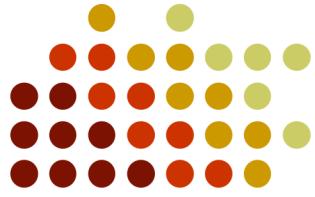
Completo

Propiedades de Árboles Binarios

- #Nodos i-ésimo nivel $\leq 2^i$ donde $i = [0, \dots, \text{altura}]$
- #Nodos externos $\leq 2^{\text{altura}}$
- Un árbol binario completo de altura h tiene $N = 2^{h+1} - 1$ nodos
- $\lceil \log_2(\#Nodos+1) \rceil - 1 < \text{altura } (h) < \#Nodos - 1$
(balanceado / completo)



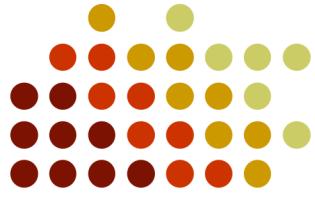
Operaciones en árboles binarios



Suponiendo que `p_raiz` es el puntero (o referencia) al nodo raíz de un árbol.

- **buscar(p_raíz, x)**. Busca el nodo que tiene clave x.
- **insertar(p_raíz, x)**. Inserta un nodo con clave x en el árbol, con su valor.
- **borrar(p_raíz, x)**. Remueve el nodo que tiene clave x.

Operaciones en árboles binarios

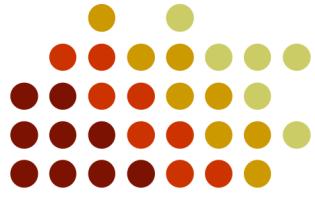


Suponiendo que `p_raiz` es el puntero (o referencia) al nodo raíz de un árbol.

- **buscar(p_raíz, x)**. Busca el nodo que tiene clave x.
- **insertar(p_raíz, x)**. Inserta un nodo con clave x en el árbol, con su valor.
- **borrar(p_raíz, x)**. Remueve el nodo que tiene clave x.

¿Cómo hacer estas operaciones?

Operaciones en árboles binarios



Suponiendo que `p_raíz` es el puntero (o referencia) al nodo raíz de un árbol.

- **buscar(p_raíz, x)**. Busca el nodo que tiene clave x.
- **insertar(p_raíz, x)**. Inserta un nodo con clave x en el árbol, con su valor.
- **borrar(p_raíz, x)**. Remueve el nodo que tiene clave x.

¿Cómo hacer estas operaciones?

Pero... los **recorridos** sobre un árbol (por ejemplo, para buscar un elemento) son similares a explorar listas, ya que en el peor caso voy a tener que pasar por todos los nodos $O(n)$

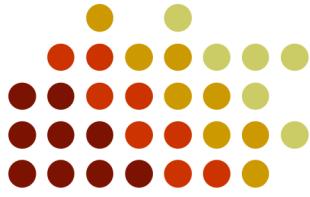
Entonces... ¿Dónde está entonces la ventaja respecto de utilizar listas vinculadas o un arreglo?

Motivación

- El tiempo de búsqueda en las **listas vinculadas** es $O(N)$... sin embargo, el costo de agregar / borrar es $O(1)$ o sea tiempo constante (no debo hacer corrimientos).
- En un **arreglo ordenado**, puedo buscar elementos en $O(\log_2 N)$ usando búsqueda binaria... Sin embargo, las inserciones (con desplazamientos) y borrados (sin huecos) implican corrimientos $O(N)$

Con una pequeña restricción, podemos convertir los árboles binarios en una solución que toma lo mejor de ambas estructuras

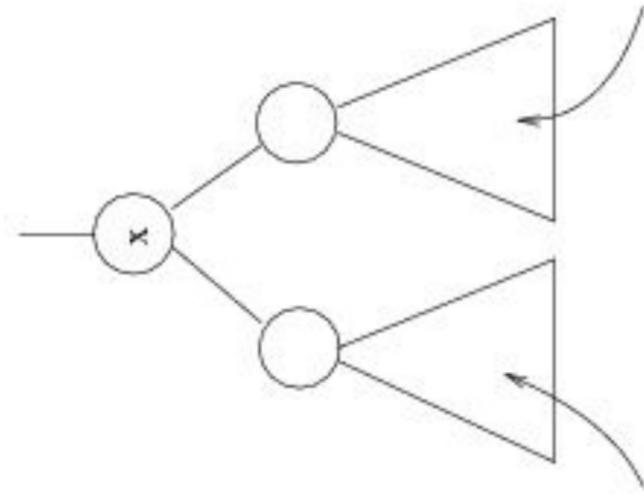
Árboles Binarios de Búsqueda



Un Árbol Binario de búsqueda (ABB) es un árbol binario que induce un orden entre los nodos, utilizando la clave (contenido del nodo).

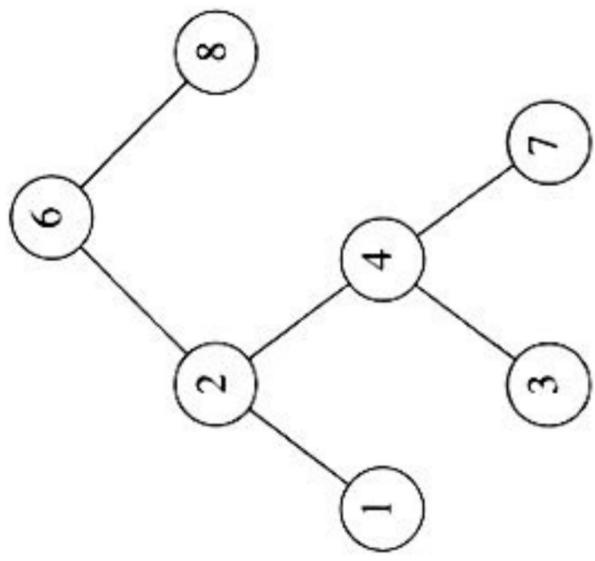
La propiedad de los ABB:

- Para cada nodo del ABB, todas las claves del subárbol **izquierdo son menores** que la de la raíz, y todas las del **derecho son mayores**.

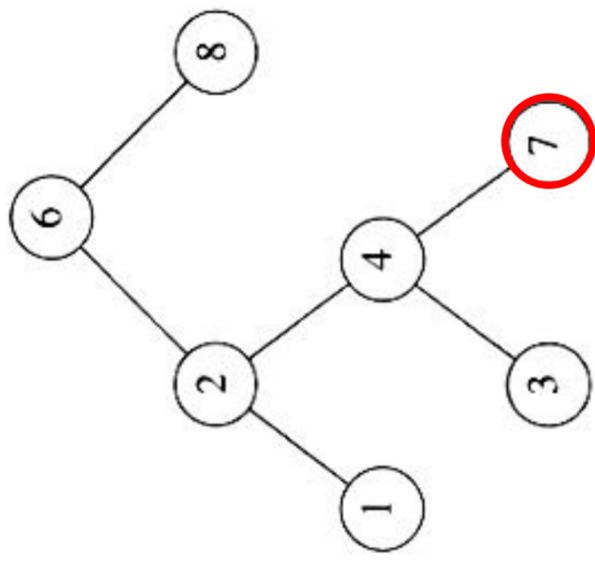


Todos los nodos tienen claves < x
Todos los nodos tienen claves > x

Árboles Binarios de Búsqueda

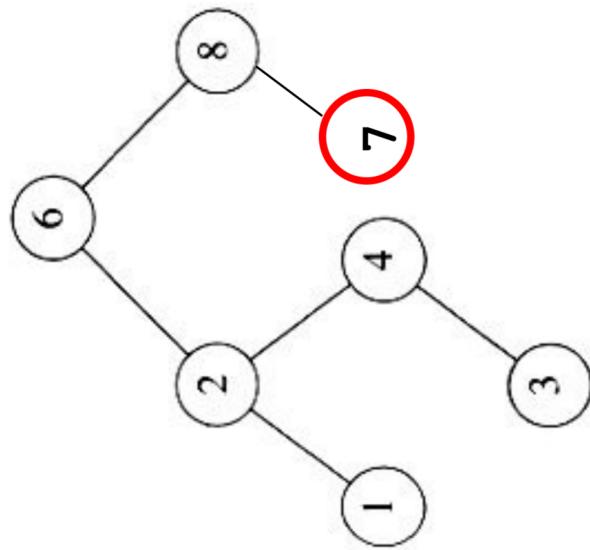
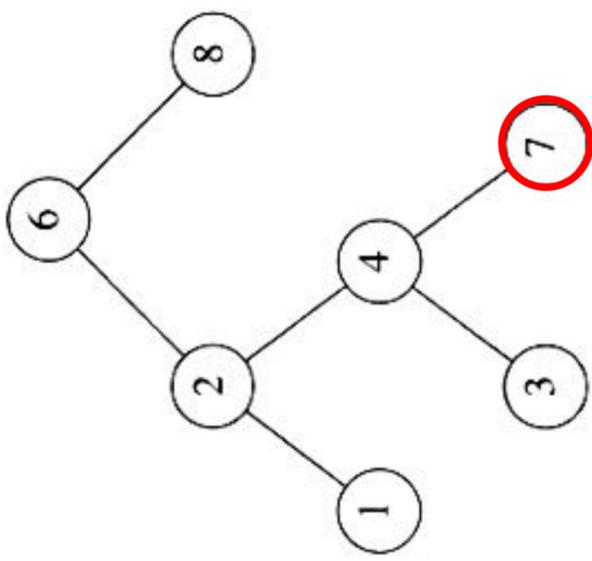


Árboles Binarios de Búsqueda



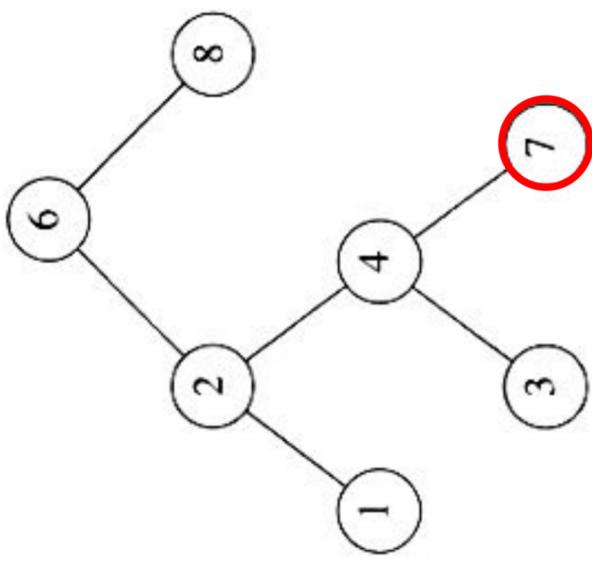
Es un árbol binario
Pero no un ABB.

Árboles Binarios de Búsqueda

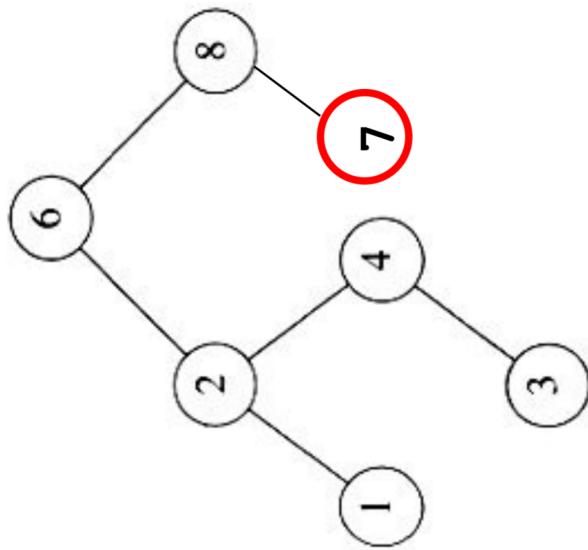


Es un árbol binario
Pero no un ABB.

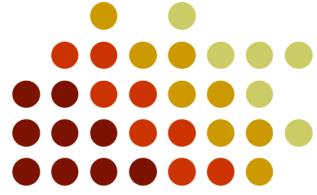
Árboles Binarios de Búsqueda



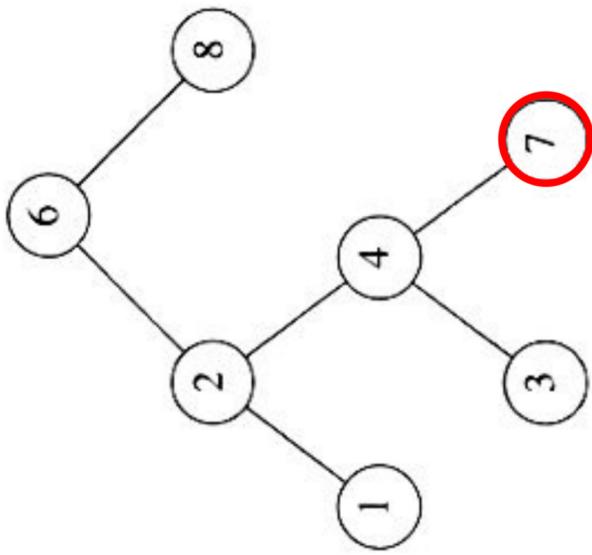
Es un árbol binario
Pero no un ABB.



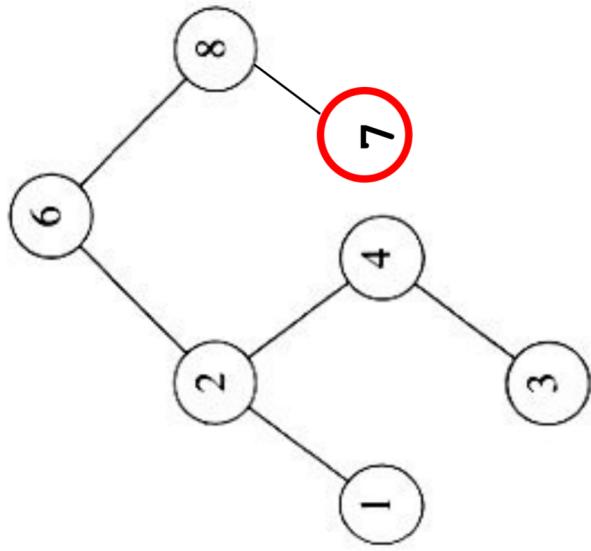
Es un ABB



Árboles Binarios de Búsqueda



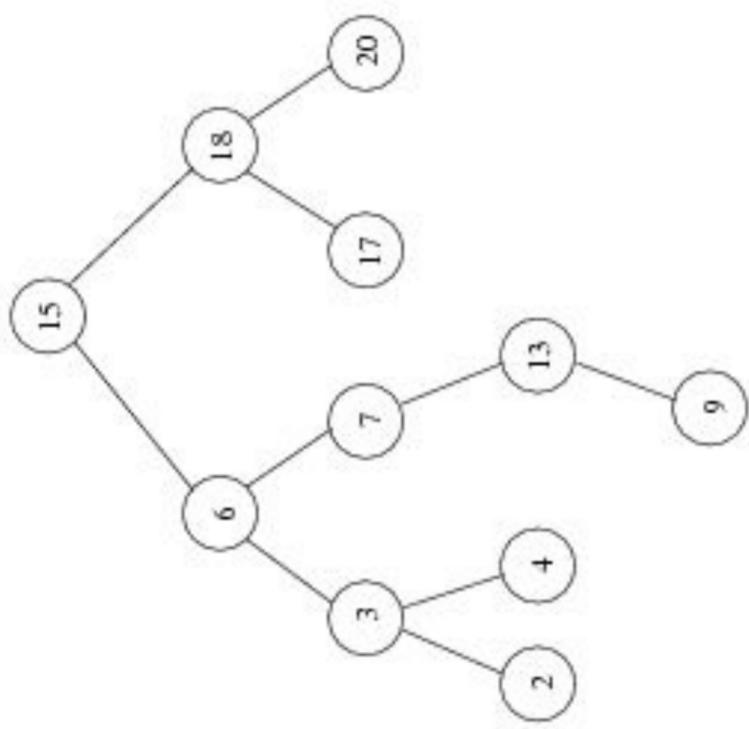
Es un árbol binario
Pero no un ABB.



Es un ABB

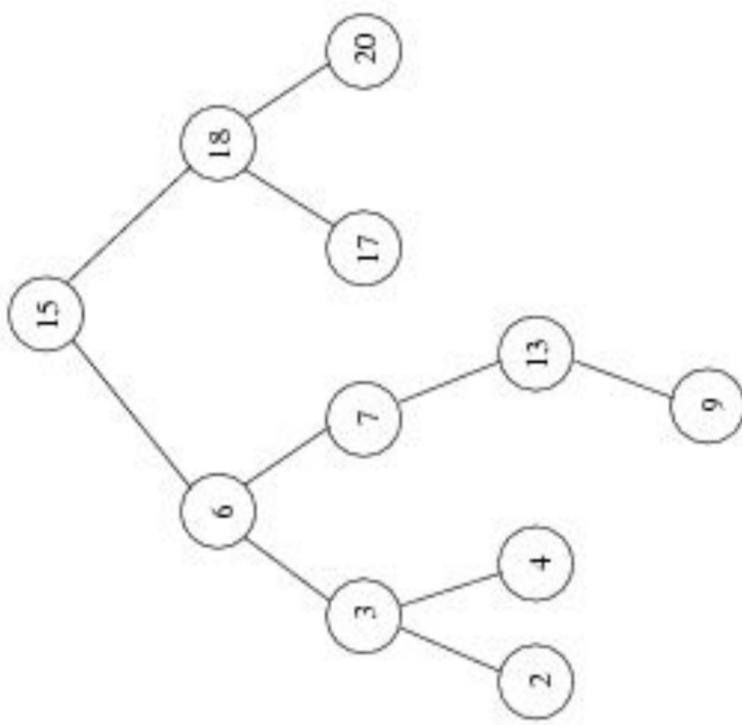
- Al contenido del nodo que sirve para establecer la relación de orden lo llamaremos CLAVE.
- Las claves deben ser únicas en el árbol.

Recorridos en ABB



- Pre-orden
- Post-orden
- En-orden

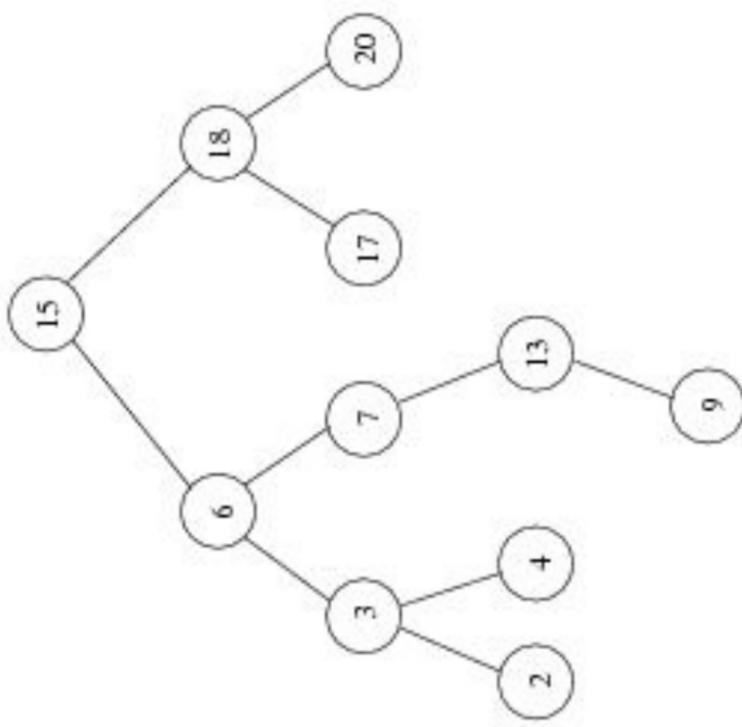
Recorridos en ABB



- Pre-orden
- Post-orden
- En-orden

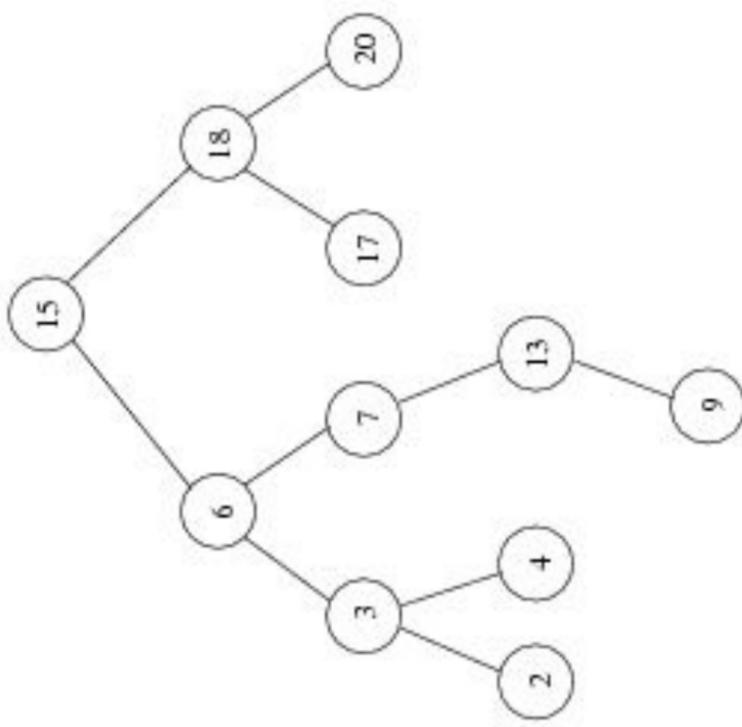
15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20

Recorridos en ABB

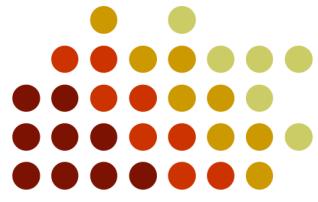


- Pre-orden 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
- Post-orden 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15
- En-orden

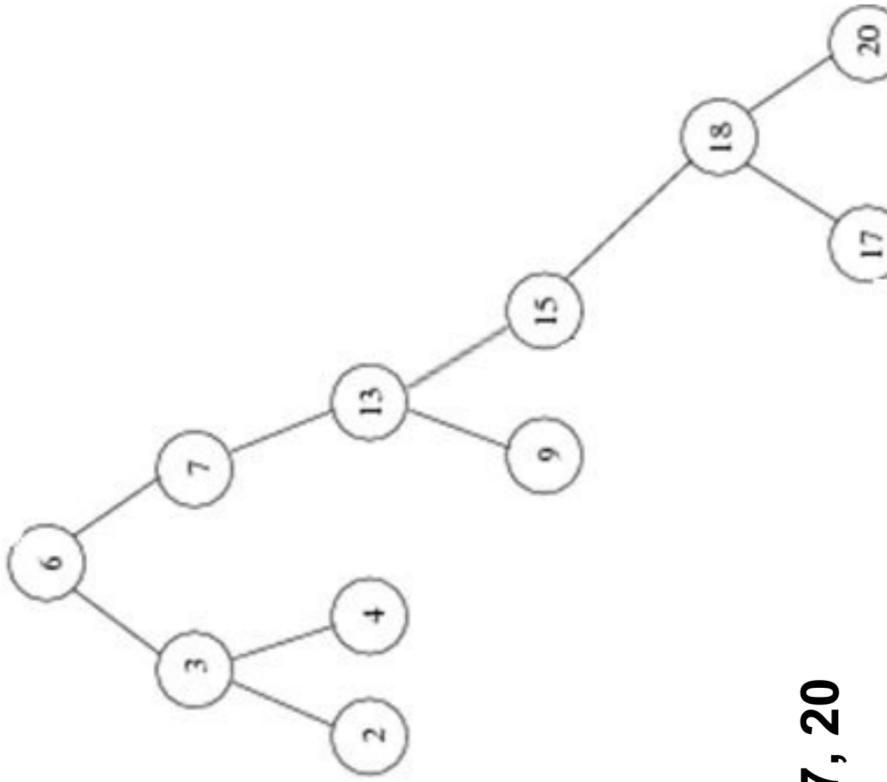
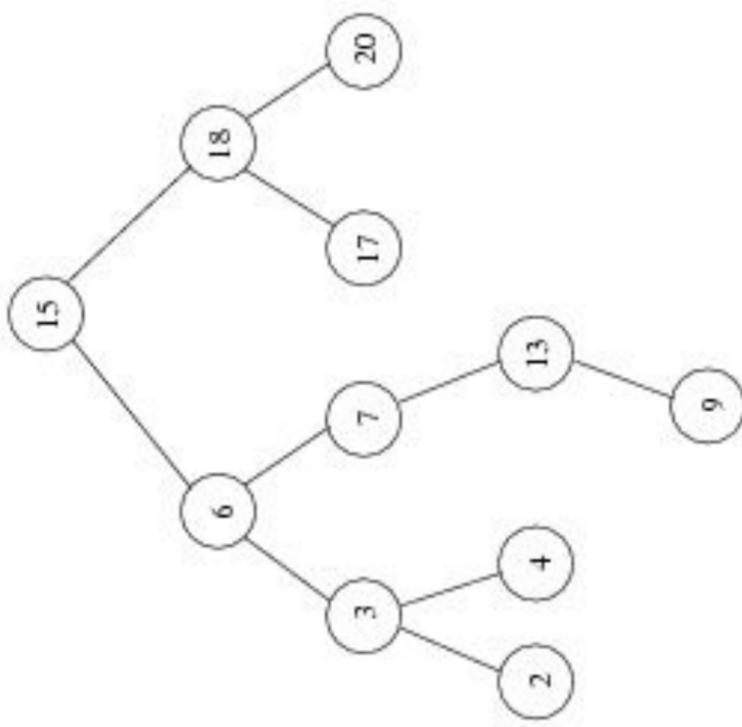
Recorridos en ABB



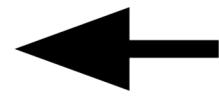
- Pre-orden 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
- Post-orden 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15
- En-orden 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20



Recorridos en ABB



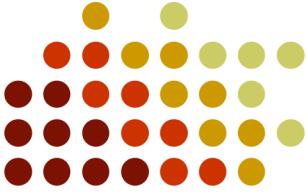
- Pre-orden **15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20**
- Post-orden **2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15**
- En-orden **2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20**



¿Si tuviera el mismo conjunto de claves en ABB diferente, cambian los recorridos?

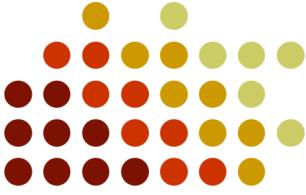
Claves y valores

- Los árboles binarios de búsqueda están ordenados según la clave (de bifurcación) contenida en cada nodo del árbol.
- La clave se utiliza para analizar el ordenamiento del nodo y proceder en las búsquedas.



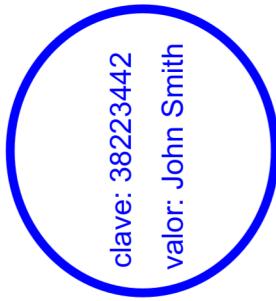
Claves y valores

- Los árboles binarios de búsqueda están ordenados según la clave (de bifurcación) contenida en cada nodo del árbol.
- La clave se utiliza para analizar el ordenamiento del nodo y proceder en las búsquedas.
- Puede resultar útil permitir que cada nodo contenga un valor (información) además de la clave.
 - El valor es un dato adicional del nodo, asociado a la clave.



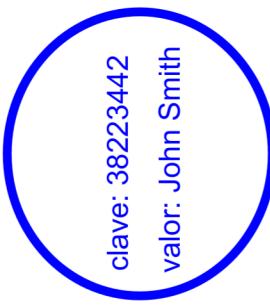
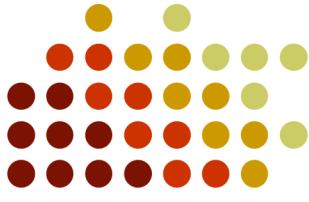
Claves y valores

- Los árboles binarios de búsqueda están ordenados según la clave (de bifurcación) contenida en cada nodo del árbol.
- La clave se utiliza para analizar el ordenamiento del nodo y proceder en las búsquedas.
- Puede resultar útil permitir que cada nodo contenga un valor (información) además de la clave.
 - El valor es un dato adicional del nodo, asociado a la clave.



Claves y valores

- Los árboles binarios de búsqueda están ordenados según la clave (de bifurcación) contenida en cada nodo del árbol.
- La clave se utiliza para analizar el ordenamiento del nodo y proceder en las búsquedas.
- Puede resultar útil permitir que cada nodo contenga un valor (información) además de la clave.
 - El valor es un dato adicional del nodo, asociado a la clave.
- Si en nuestro dominio existieran claves repetidas, podemos asociar al nodo con dicha clave una lista de valores (factoreo).
Por ej: si la clave es la fecha de nacimiento de los alumnos de la facultad.

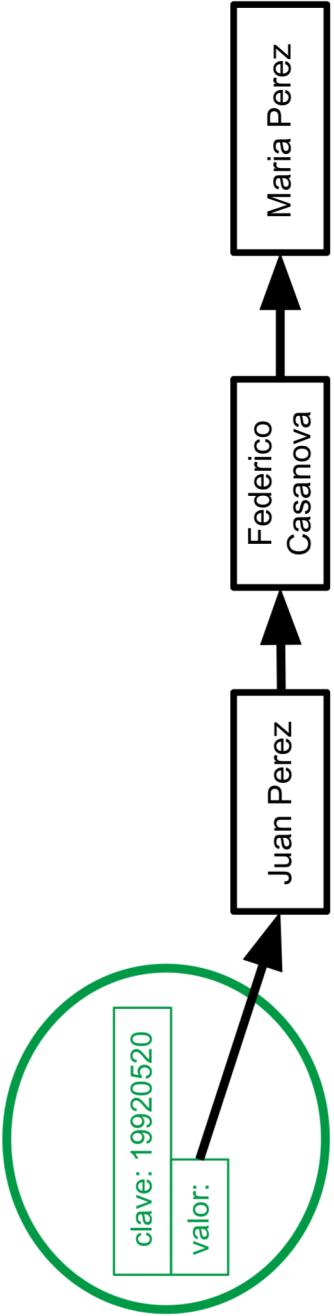


Claves y valores

- Los árboles binarios de búsqueda están ordenados según la clave (de bifurcación) contenida en cada nodo del árbol.
- La clave se utiliza para analizar el ordenamiento del nodo y proceder en las búsquedas.
- Puede resultar útil permitir que cada nodo contenga un valor (información) además de la clave.
- El valor es un dato adicional del nodo, asociado a la clave.



- Si en nuestro dominio existieran claves repetidas, podemos asociar al nodo con dicha clave una lista de valores (factoreo). Por ej: si la clave es la fecha de nacimiento de los alumnos de la facultad.



Operaciones en ABB

Suponiendo que `p_raíz` es el puntero (o referencia) al nodo raíz de un árbol.

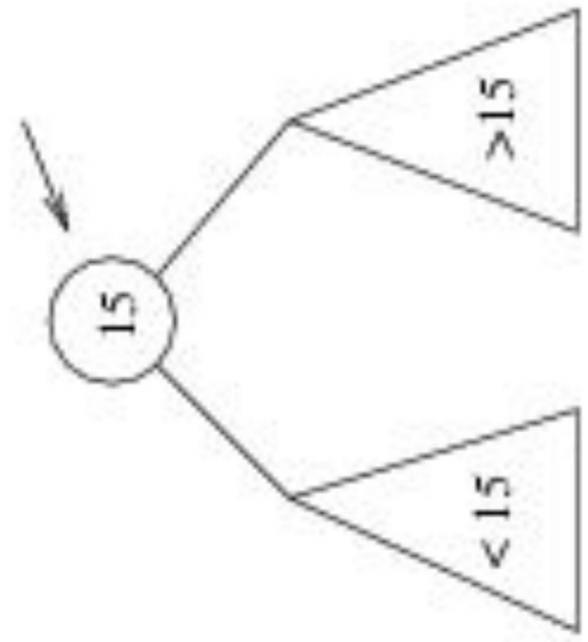
- **buscar(p_raíz, x)**. Busca el nodo que tiene clave x.
- **insertar(p_raíz, x)**. Inserta un nodo con clave x en el árbol, con su valor.
- **borrar(p_raíz, x)**. Remueve el nodo que tiene clave x.

¿Cómo cambian estas operaciones si estamos en un ABB?

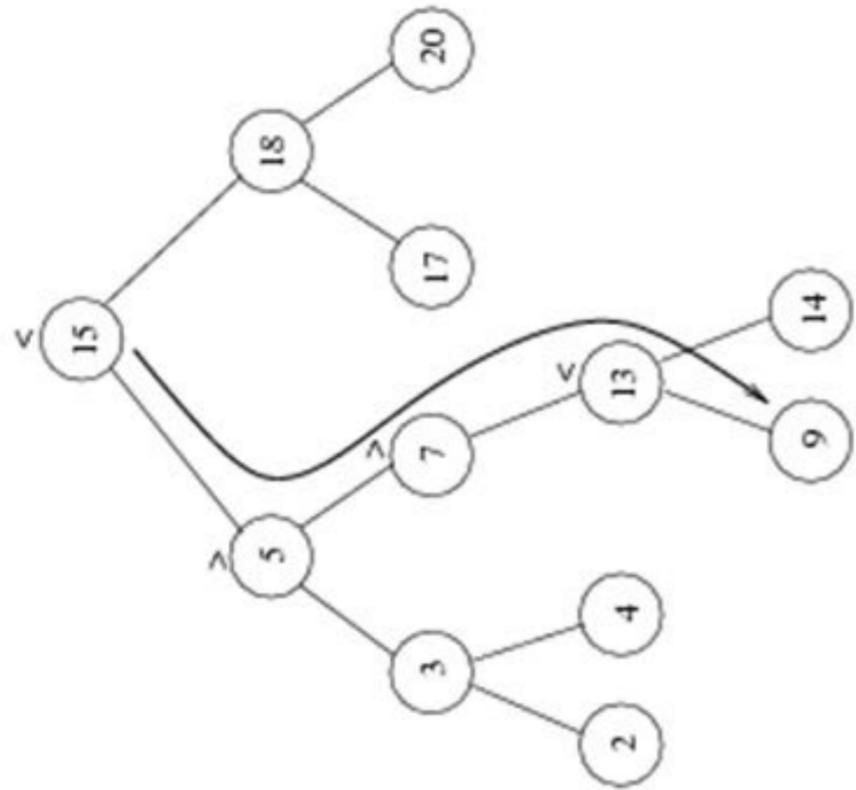


Búsqueda en ABB

- Búsqueda de clave 15 → Se visita la raíz y termina.
- Búsqueda de una clave $< 15 \rightarrow$ buscar en el subárbol izquierdo.
- Búsqueda de una clave $> 15 \rightarrow$ buscar en el subárbol derecho.



Búsqueda en ABB



Buscar la clave 9:

Comparar 9 con 15 → avanzar hacia el subárbol izquierdo

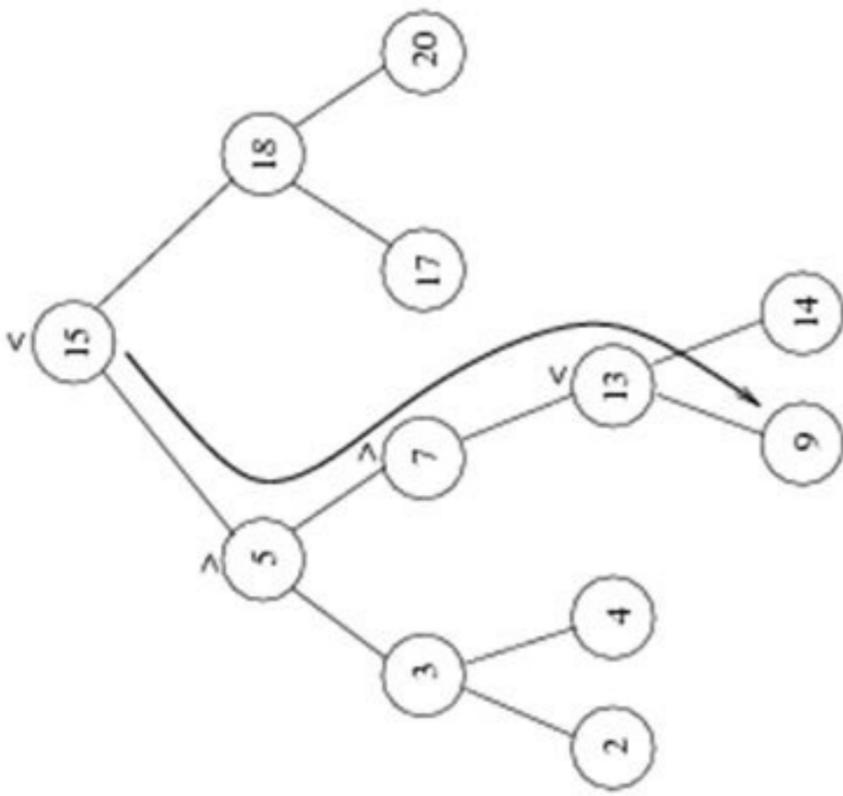
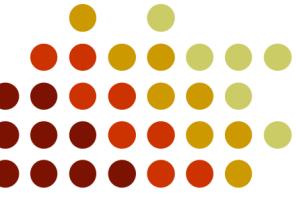
Comparar 9 con 5 → avanzar hacia el subárbol derecho

Comparar 9 con 7 → avanzar hacia el subárbol derecho

Comparar 9 con 13 → avanzar hacia el subárbol izquierdo

Comparar 9 con 9 → ¡éxito!

Búsqueda en ABB



Buscar la clave 9:

Comparar 9 con 15 → avanzar hacia el subárbol izquierdo

Comparar 9 con 5 → avanzar hacia el subárbol derecho

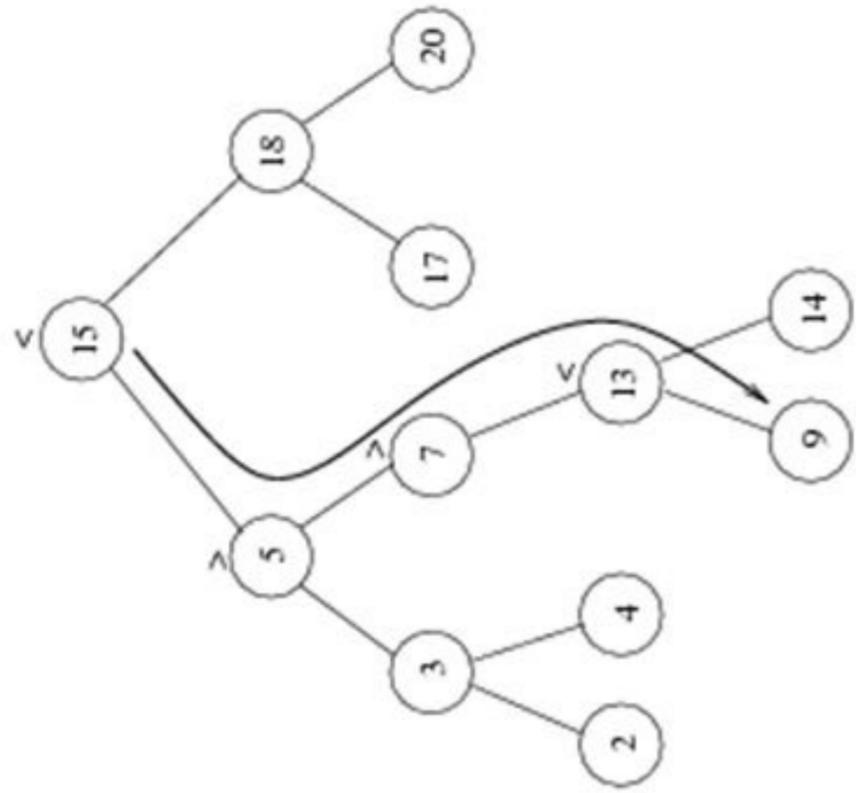
Comparar 9 con 7 → avanzar hacia el subárbol derecho

Comparar 9 con 13 → avanzar hacia el subárbol izquierdo

Comparar 9 con 9 → ¡éxito!

¿Y si hubiera que buscar la clave 6?

Búsqueda en ABB



Buscar la clave 9:

Comparar 9 con 15 → avanzar hacia el subárbol izquierdo

Comparar 9 con 5 → avanzar hacia el subárbol derecho

Comparar 9 con 7 → avanzar hacia el subárbol izquierdo

Comparar 9 con 9 → éxito!

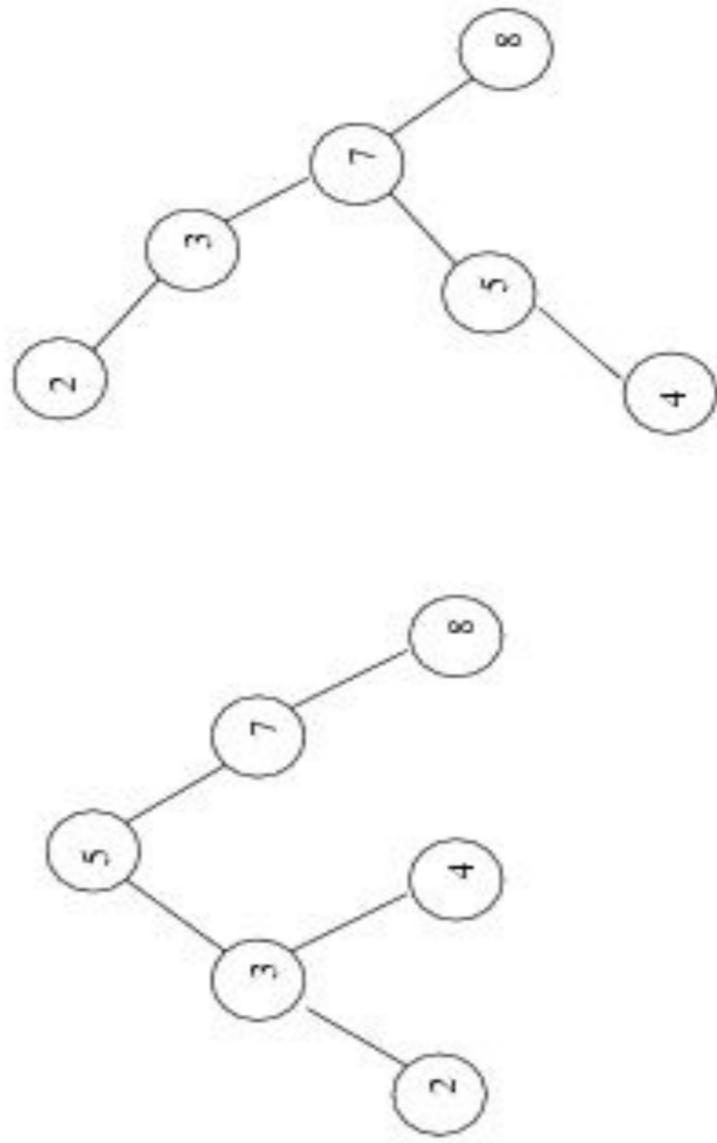
Complejidad temporal de búsqueda O(h)
¿Y si hubiera que buscar la clave 6?

Nodo mínimo y máximo

¿Cómo obtener el nodo con el menor valor y cómo el de mayor valor?

→ Nodo más izquierdo (NMI)

→ Nodo más derecho (NMD)



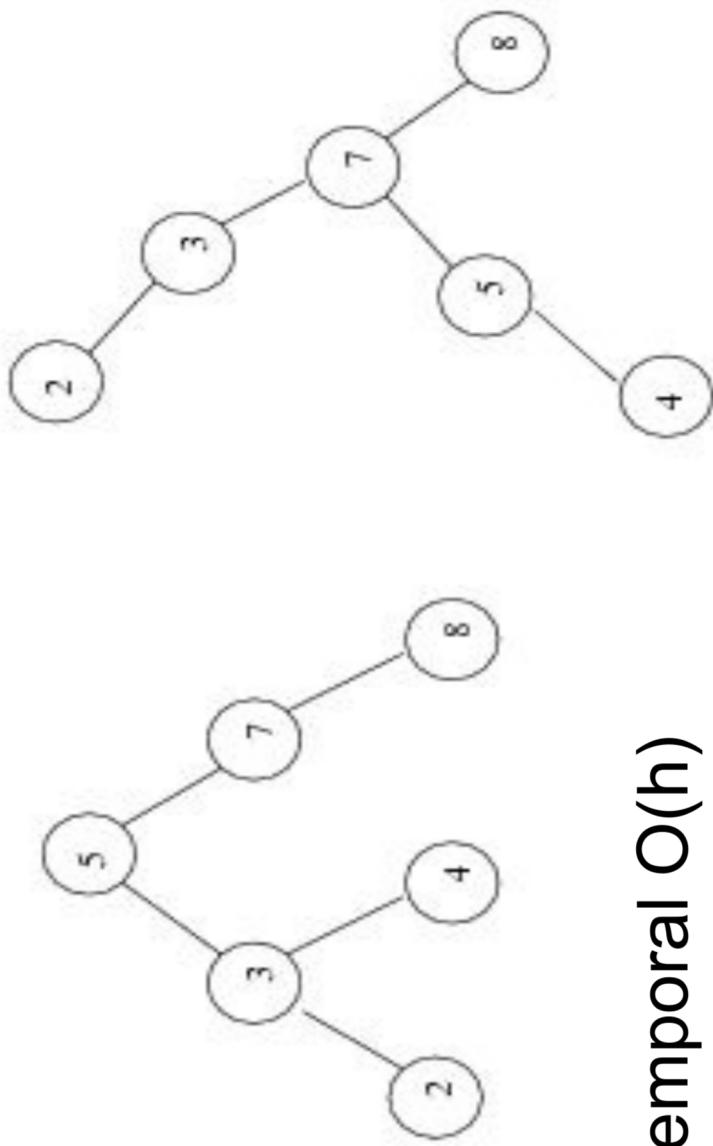
2 y 8 en ambos casos

Nodo mínimo y máximo

¿Cómo obtener el nodo con el menor valor y cómo el de mayor valor?

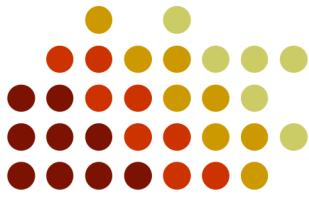
→ Nodo más izquierdo (NMI)

→ Nodo más derecho (NMD)



Complejidad temporal $O(h)$

2 y 8 en ambos casos



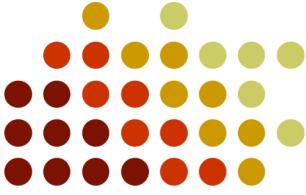


Construcción de un ABB

- Inserciones
- Borrados
- IMPORTANTE: mantener la propiedad de búsqueda.
 - Valores menores a la izquierda
 - Valores mayores a la derecha

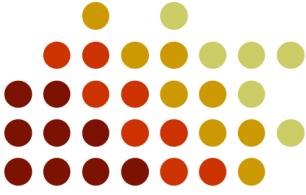
Inserción en un ABB

- Buscar el lugar para insertar X
- Si se encuentra X ¿termina? ¿O se inserta nuevamente?
- Si no se encuentra, insertar X en el último lugar alcanzado en la búsqueda



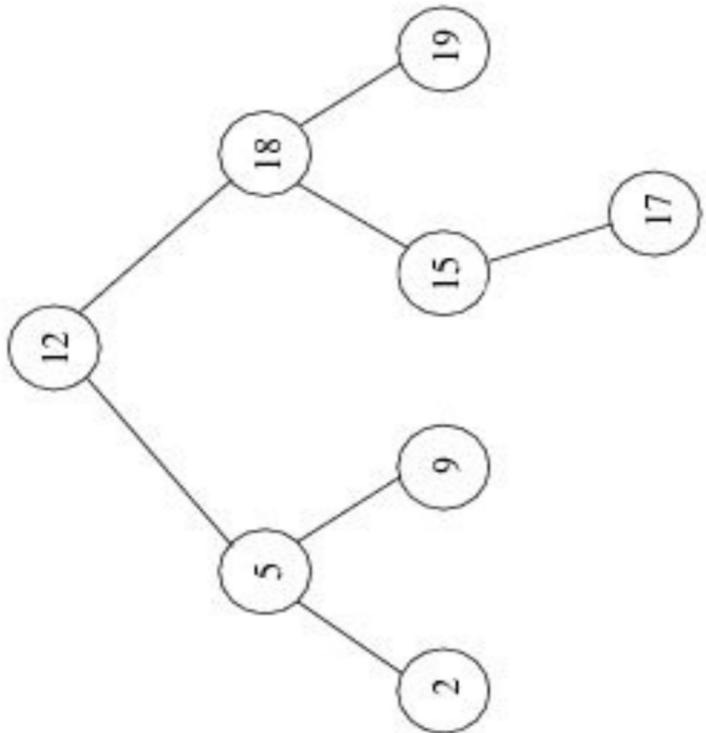
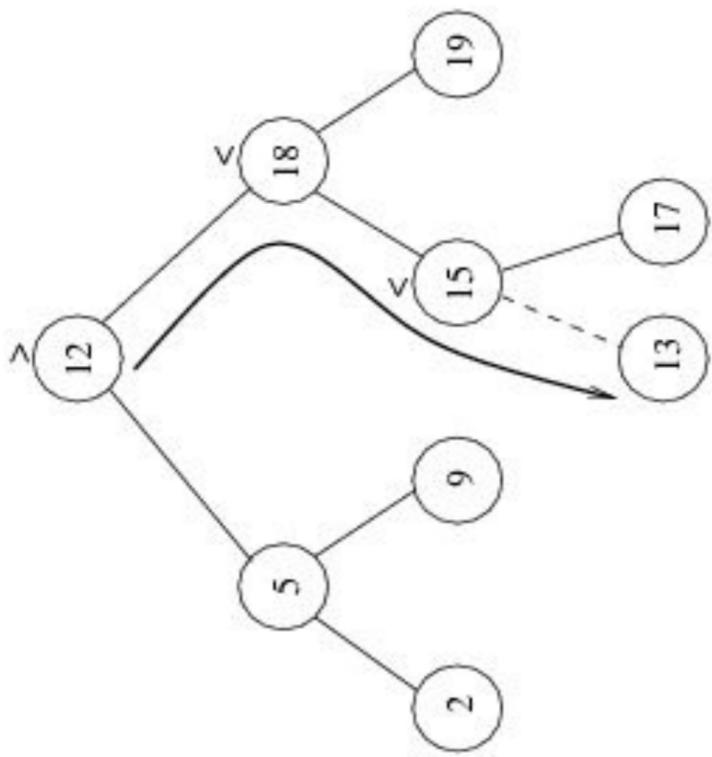
Inserción en un ABB

- Buscar el lugar para insertar X
- Si se encuentra X ¿termina? ~~O se inserta nuevamente?~~
- Si no se encuentra, insertar X en el último lugar alcanzado en la búsqueda



Inserción en un ABB

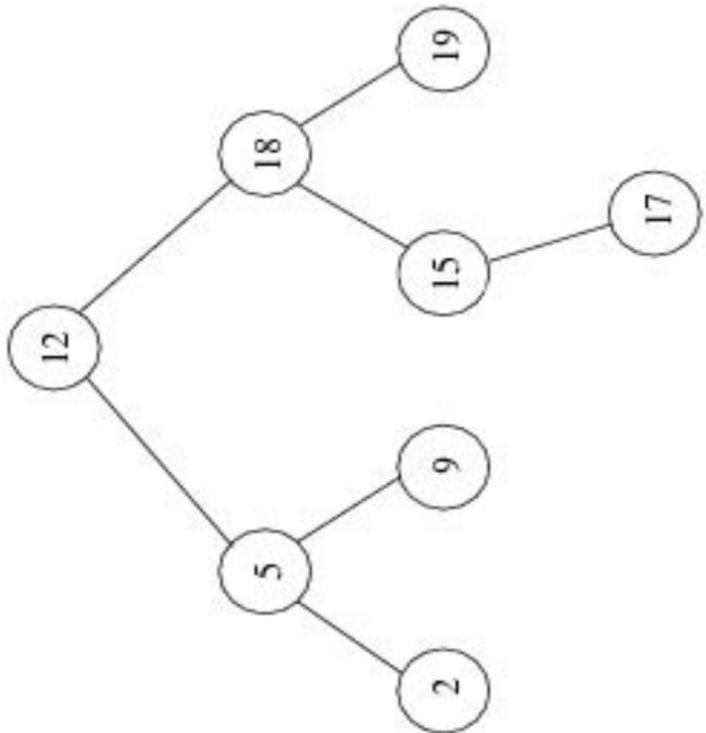
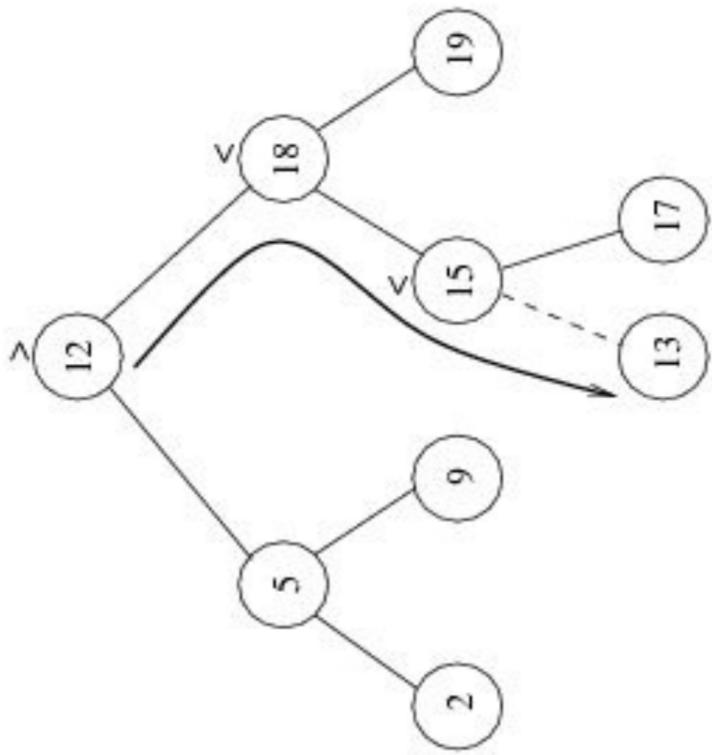
- Buscar el lugar para insertar X
- Si se encuentra X ¿termina? ~~O se inserta nuevamente?~~
- Si no se encuentra, insertar X en el último lugar alcanzado
- Por ej. insertamos el 13:



Inserción en un ABB

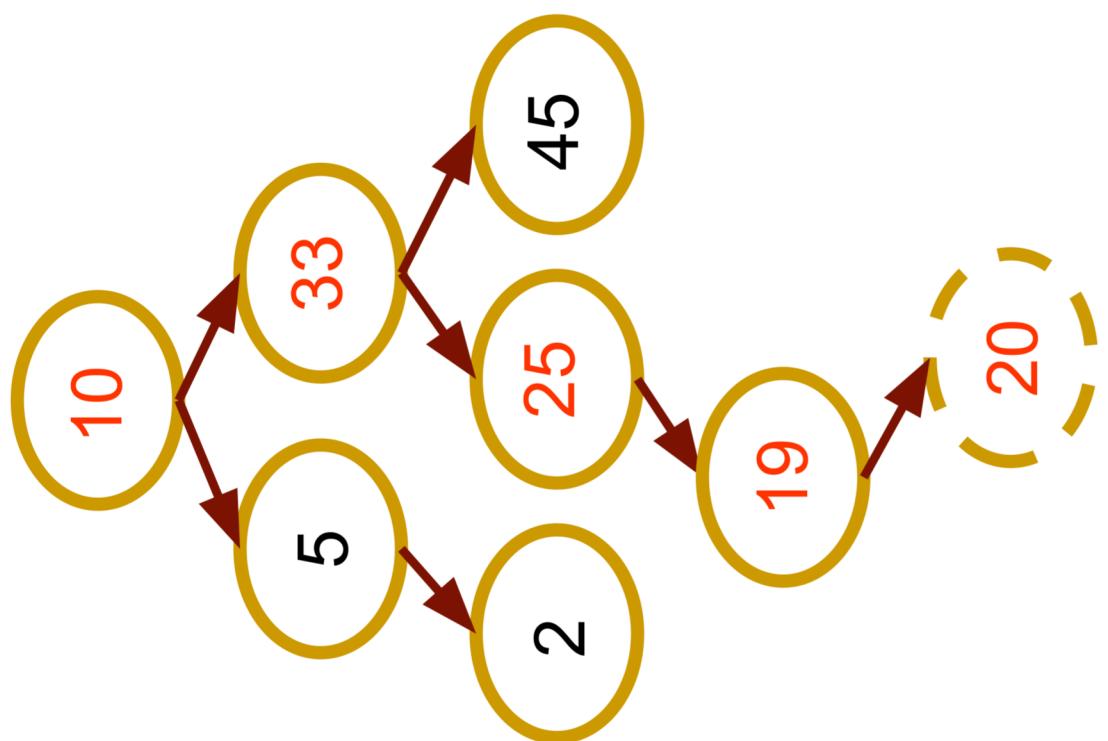
- Buscar el lugar para insertar X
- Si se encuentra X ¿termina? ~~O se inserta nuevamente?~~
- Si no se encuentra, insertar X en el último lugar alcanzado en la búsqueda
- Por ej. insertamos el 13:

Complejidad temporal $O(h)$



Inserción (ej.)

- Insertar (20)



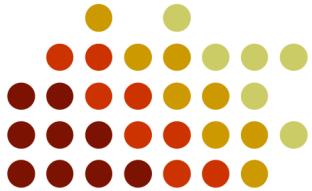
10 < 20, derecha

33 > 20, izquierda

25 > 20, izquierda

19 < 20, derecha

Insertar 20 a la derecha



Borrado en un ABB

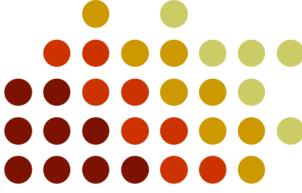
Podemos encontrar 3 casos:

Borrado en un ABB

Podemos encontrar 3 casos:

Caso 1) El nodo es una hoja:

- Borrar el nodo.



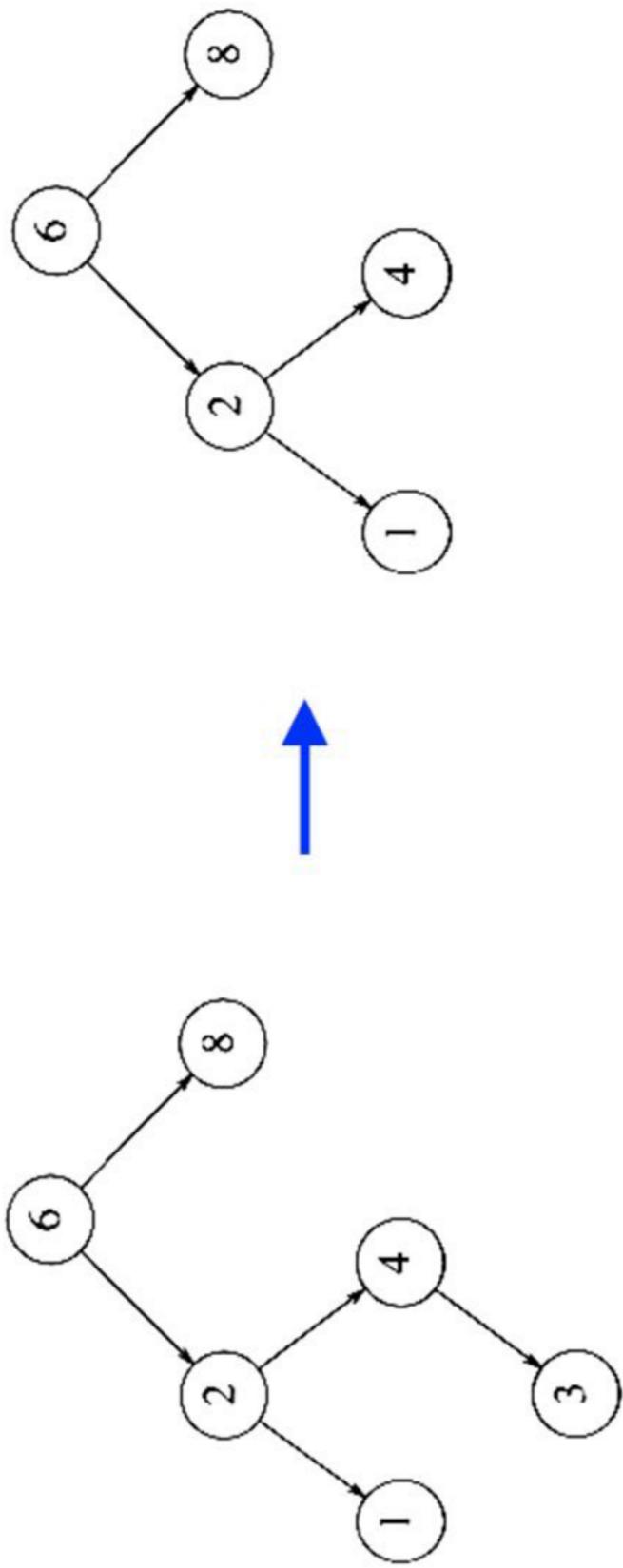
Borrado en un ABB

Podemos encontrar 3 casos:

Caso 1) El nodo es una hoja:

- Borrar el nodo.

Por ejemplo, borrar el 3:



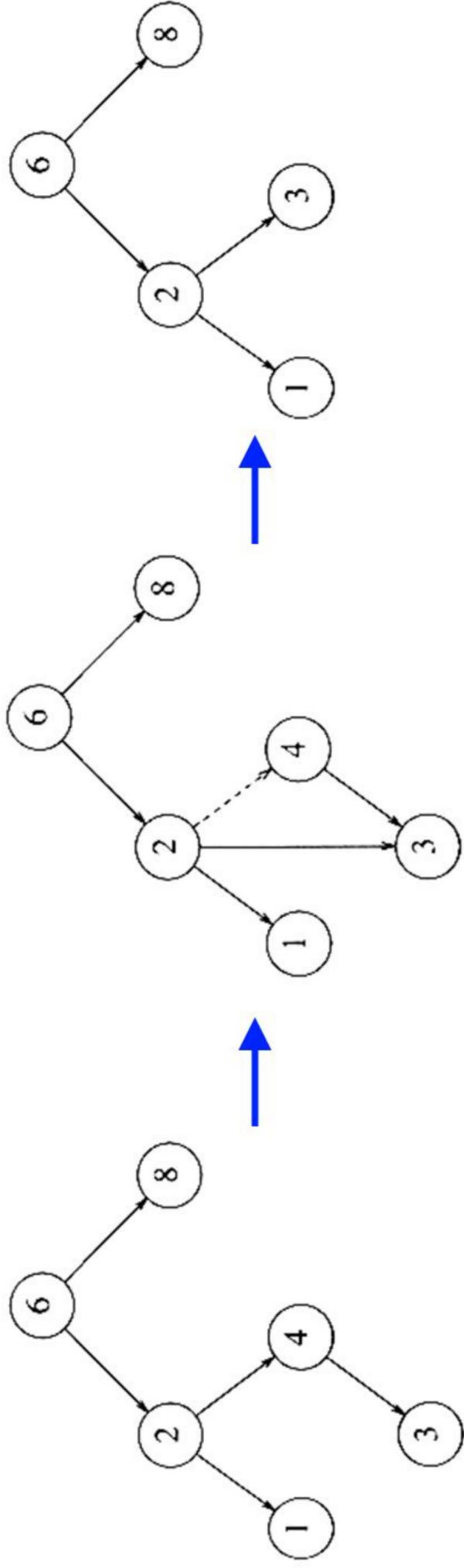
Borrado en un ABB

Podemos encontrar 3 casos:

Caso 2) El nodo tiene un solo hijo:

- Acomodar el puntero para ignorar el nodo borrado y apuntar al hijo.

Por ejemplo, borrar el 4:

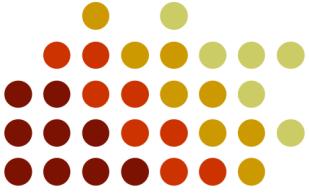


Borrado en un ABB

Podemos encontrar 3 casos:

Caso 3) El nodo tiene sus 2 hijos:

- Reemplazar con el NMI del subárbol derecho del nodo a borrar.
- Borrar el NMI del subárbol derecho
 - Se reduce a aplicar caso 1 o caso 2.
- Complejidad temporal = $O(\text{altura del árbol})$



Borrado en un ABB

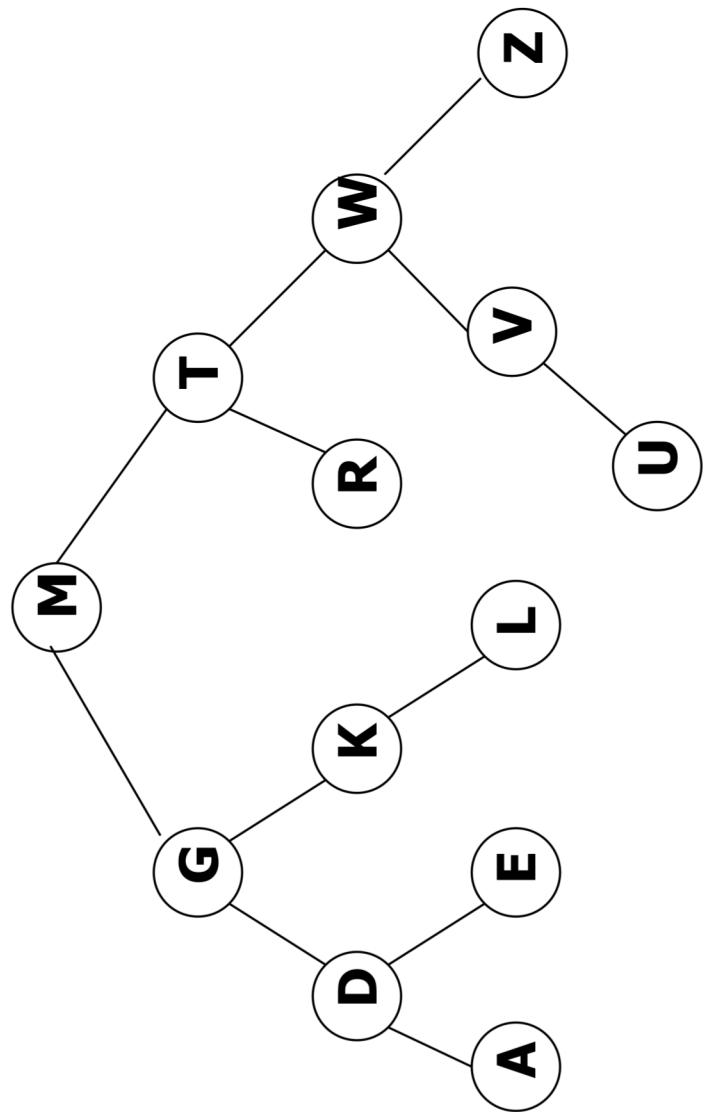
Podemos encontrar 3 casos:

Caso 3) El nodo tiene sus 2 hijos:

- Reemplazar con el NMI del subárbol derecho del nodo a borrar.
- Borrar el NMI del subárbol derecho
 - Se reduce a aplicar caso 1 o caso 2.
- Complejidad temporal = O(altura del árbol)

¿Qué es ese valor?

El NMI del subárbol derecho, es el menor de los mayores a esa clave.



Borrado en un ABB

Podemos encontrar 3 casos:

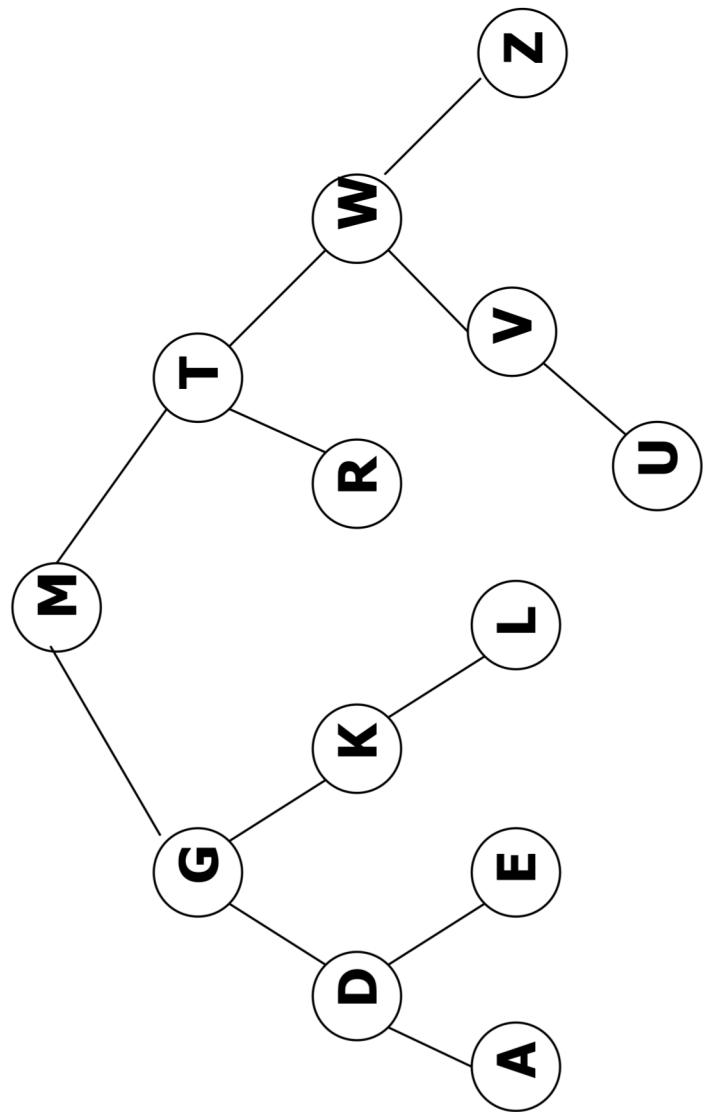
Caso 3) El nodo tiene sus 2 hijos:

- Reemplazar con el NMI del subárbol derecho del nodo a borrar.
- Borrar el NMI del subárbol derecho
 - Se reduce a aplicar caso 1 o caso 2.
- Complejidad temporal = O(altura del árbol)

¿Qué es ese valor?

El NMI del subárbol derecho, es el menor de los mayores a esa clave.

También podría usarse el NMD del subárbol izquierdo.



Borrado en un ABB

Podemos encontrar 3 casos:

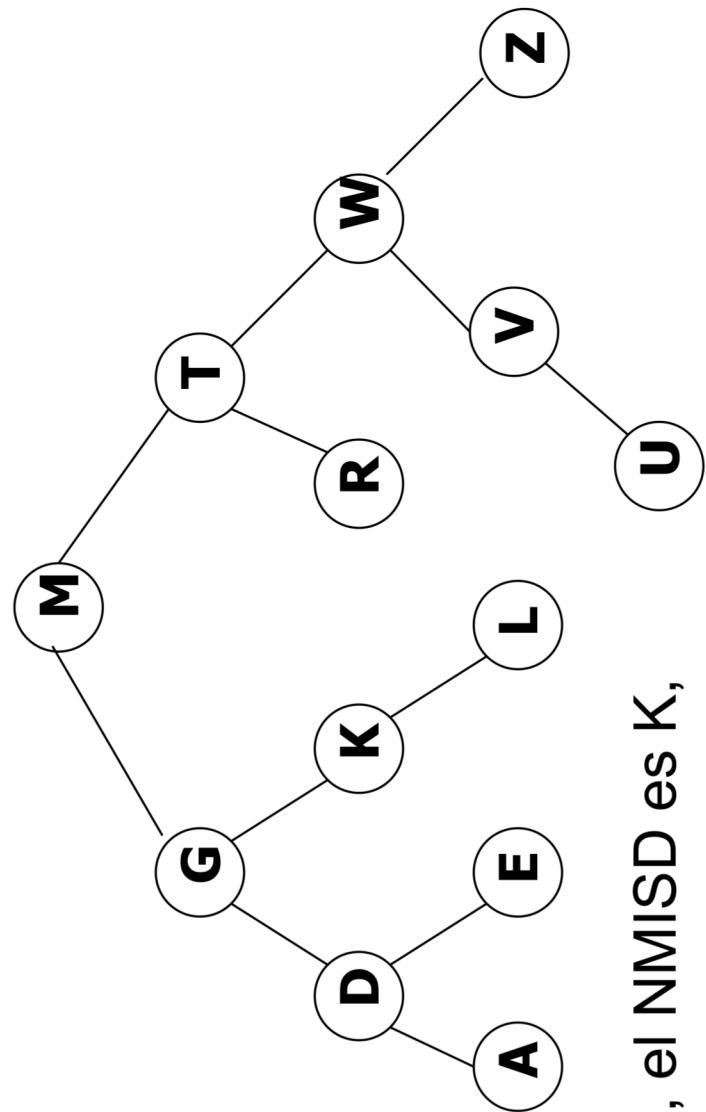
Caso 3) El nodo tiene sus 2 hijos:

- Reemplazar con el NMI del subárbol derecho del nodo a borrar.
- Borrar el NMI del subárbol derecho
 - Se reduce a aplicar caso 1 o caso 2.
- Complejidad temporal = O(altura del árbol)

¿Qué es ese valor?

El NMI del subárbol derecho, es el menor de los mayores a esa clave.

También podría usarse el NMD del subárbol izquierdo.



Supongamos queremos borrar la G, el NMISD es K, y el NMDSI es E

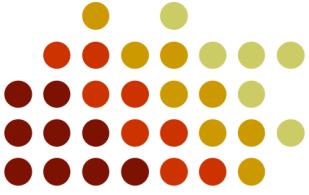
Borrado en un ABB

- Algoritmo: **Suprimir (x, A)**

1. Buscar la clave X → nodo N que la contiene o null
2. Si el nodo N es hoja, borrar N.
3. Si el nodo N tiene un solo hijo, borrar el nodo N y re-acomodar puntero.
4. Si el nodo N tiene dos hijos:
 - a) Reemplazar N con el NMi del subárbol derecho a N
 - b) **Suprimir (NMiSD, N.der)**

Complejidad temporal del borrado O(h)

La complejidad proviene del proceso de búsqueda del elemento que voy a borrar



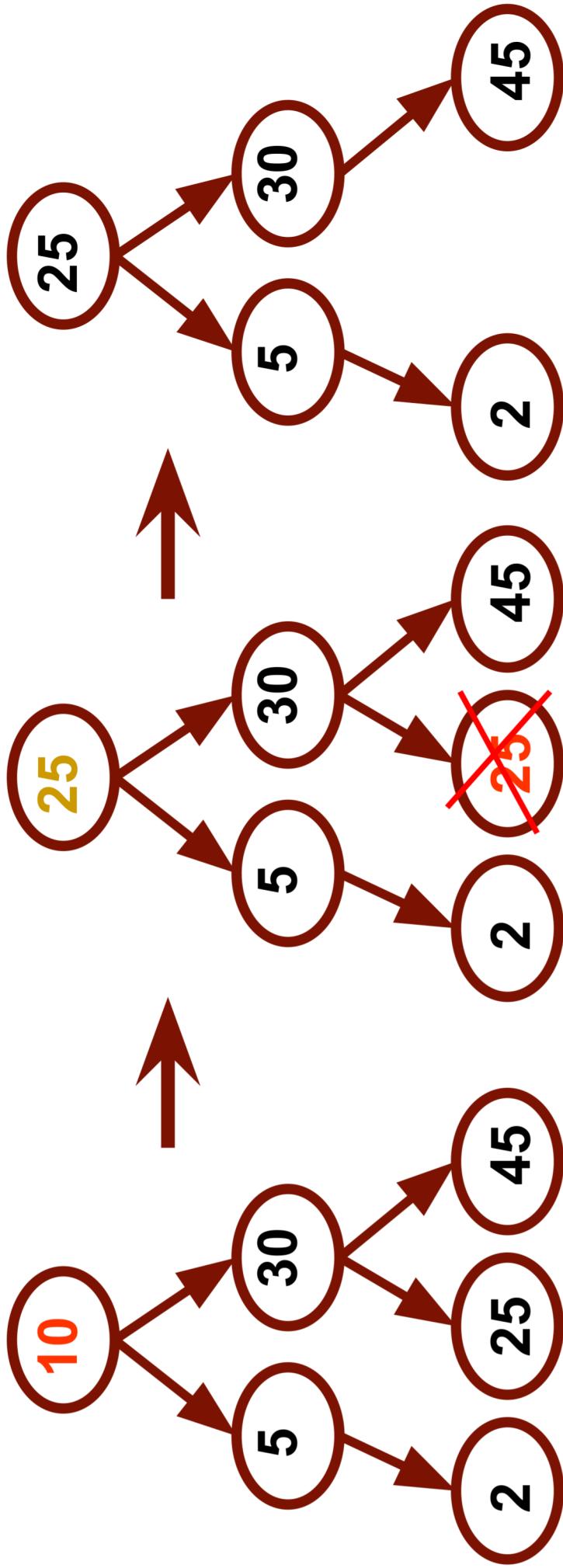
Borrado de una hoja

- Borrar (25)



Borrado de un nodo interno

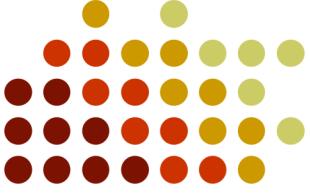
- Borrar (10)



Reemplazar 10
por el NMI del
subárbol derecho

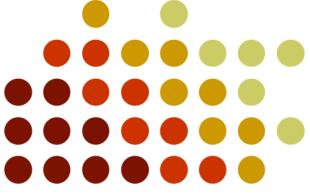
Borrar el NMI del
subárbol derecho
(25)

Resultado



La importancia de la altura en los ABBs

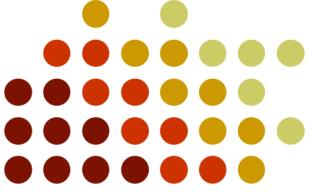
La mayoría de los métodos más importantes de un ABB (búsqueda, inserción y borrado), dependen de la altura de dicho árbol.



La importancia de la altura en los ABBs

La mayoría de los métodos más importantes de un ABB (búsqueda, inserción y borrado), dependen de la altura de dicho árbol.

¿Qué altura va a tener mi árbol? Va a depender no solo de la cantidad de elementos que inserte, sino del **ORDEN** en que inserte los elementos



La importancia de la altura en los ABBS

La mayoría de los métodos más importantes de un ABB (búsqueda, inserción y borrado), dependen de la altura de dicho árbol.

¿Qué altura va a tener mi árbol? Va a depender no solo de la cantidad de elementos que inserte, sino del **ORDEN** en que inserte los elementos

Árbol completo

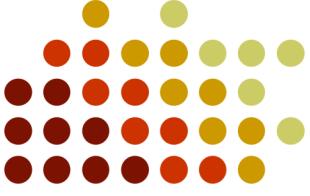
Mejor caso

$$h = \lceil \log_2(N+1) \rceil - 1$$

Enredadera

Peor caso

$$h = N - 1$$



La importancia de la altura en los ABBS

La mayoría de los métodos más importantes de un ABB (búsqueda, inserción y borrado), dependen de la altura de dicho árbol.

¿Qué altura va a tener mi árbol? Va a depender no solo de la cantidad de elementos que inserte, sino del **ORDEN** en que inserte los elementos

Árbol completo

Mejor caso

$$h = \lceil \log_2(N+1) \rceil - 1$$

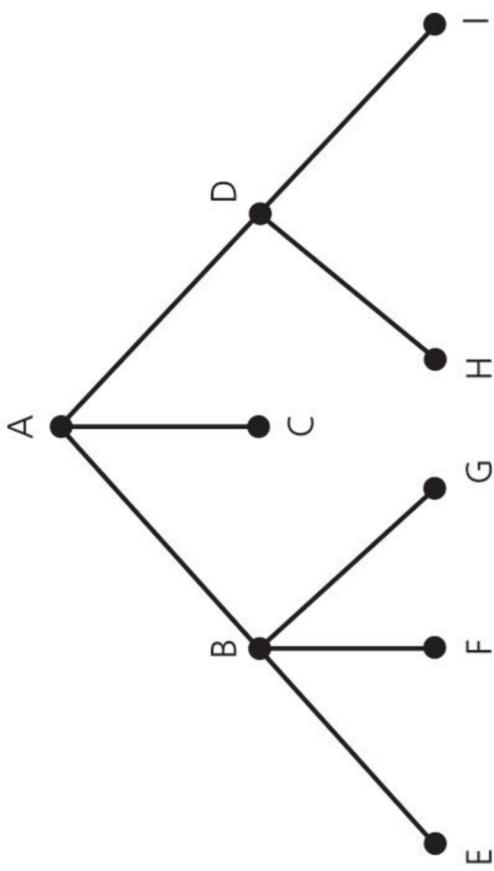
Peor caso

Enredadera

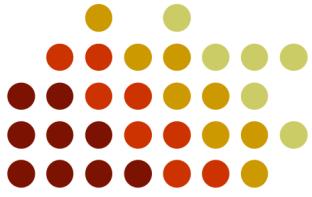
¿Cómo ataco este problema? → Técnicas de balanceo (ej. AVL)

Árboles n-arios

- Es una generalización de un árbol binario cuyos nodos pueden tener hasta N hijos.

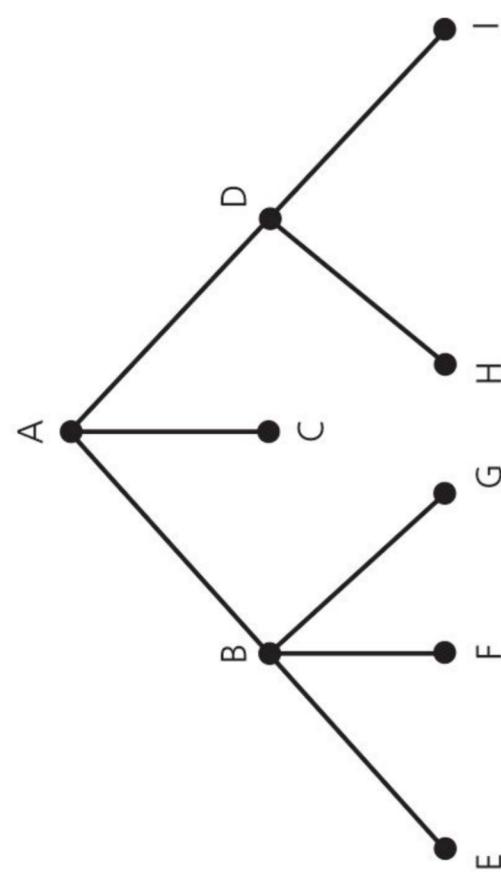


Un árbol ternario (3-ario)

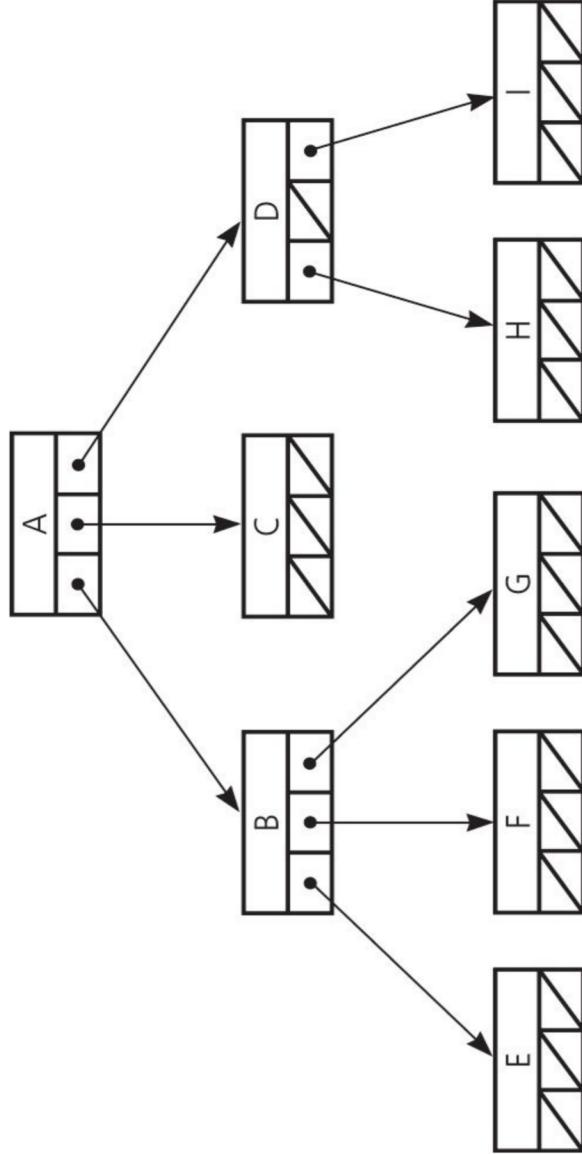


Árboles n-arios

- Es una generalización de un árbol binario cuyos nodos pueden tener hasta N hijos.



Un árbol ternario (3-ario)

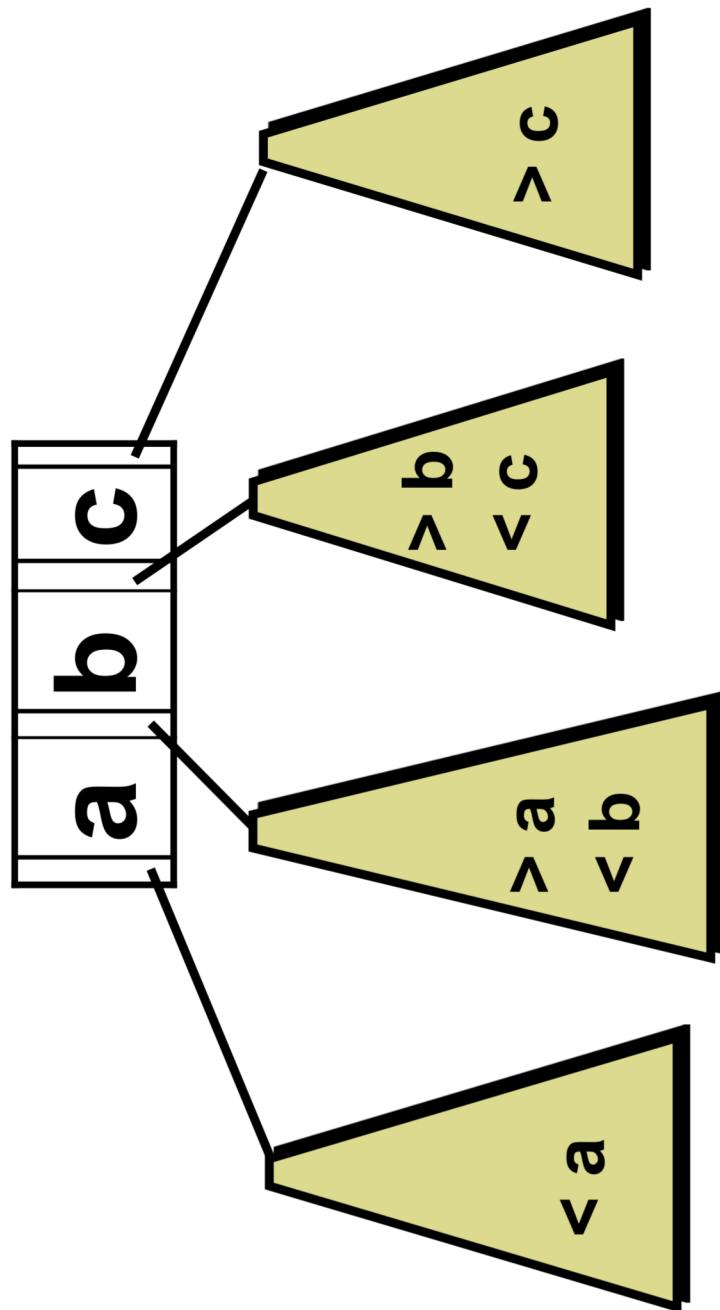


Possible implementación: arreglo en cada nodo

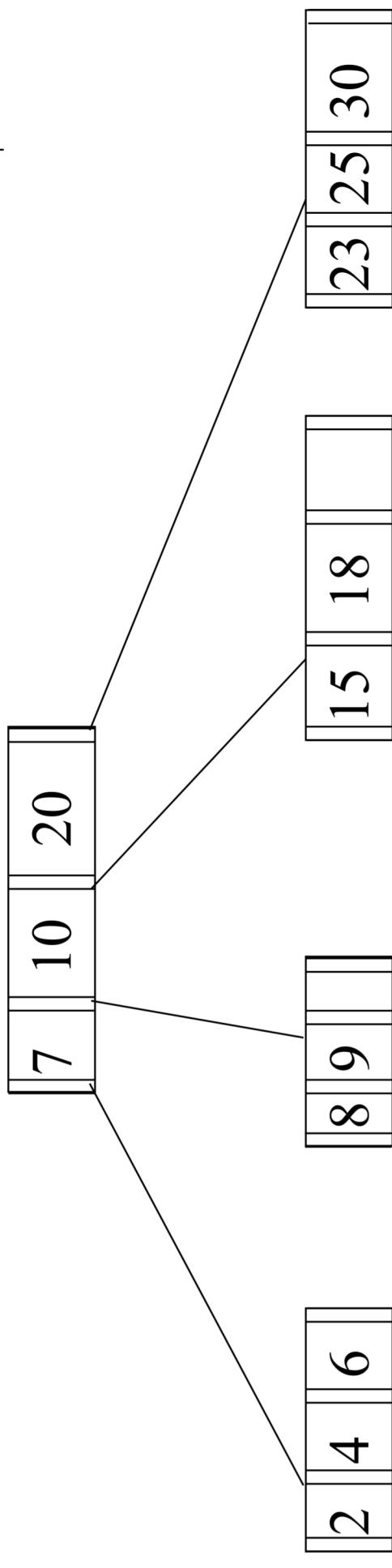
¿Otra? Una lista vinculada de hijos en cada nodo

Árbol n-ario de Búsqueda

- En cada nodo hay N-1 claves y N punteros a nodos hijos.



Árbol n-ario de Búsqueda

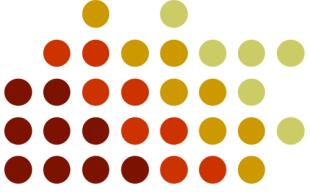


¿Qué se busca con esto?

Árboles más chatos (**menos altura h**)

→ Más eficiente, ya que en el peor caso las operaciones eran $O(h)$

Por ejemplo, familia de árboles B: son árboles n-arios de búsqueda y balanceados.



Tarea:

Obtener el ABB resultante de ingresar los siguientes elementos:

- A) J, N, B, D, W, E, T, Y, H, M, F
- B) B, D, E, F, H, J, N, M, T, W, Y

Efectúe los diferentes recorridos transversales del árbol obtenido en A) compare con los correspondientes en B)

Ejercicios II

¿Cuál es el árbol resultante luego de suprimir W, T, A, y M?

Nota: Usar NMISD

