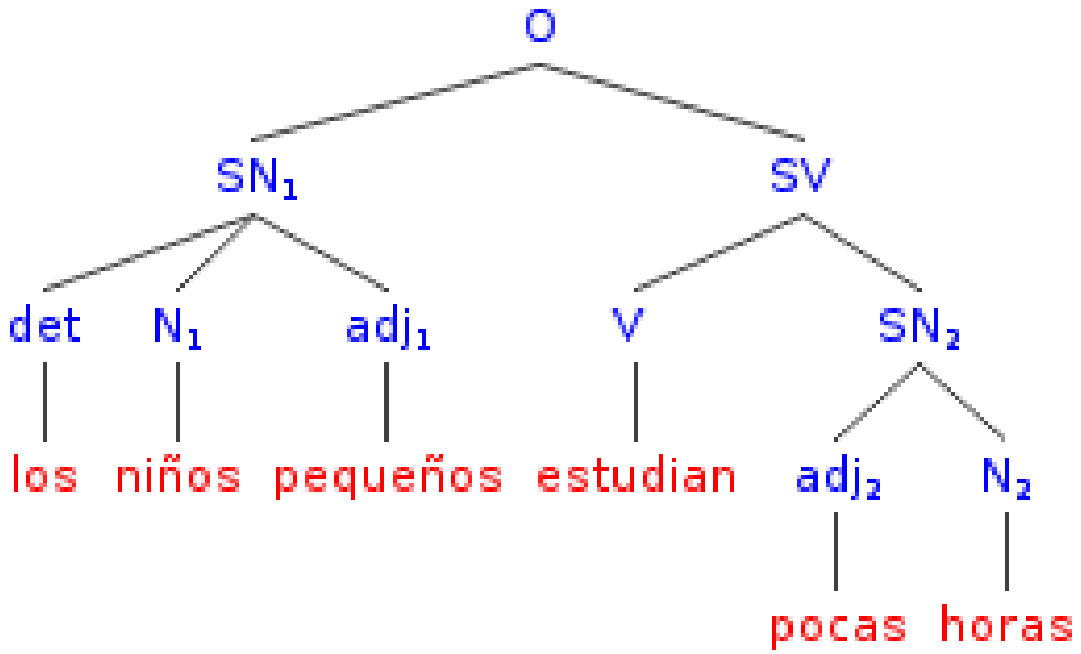
ESTRUCTURAS JERARQUICAS

Estructura de Datos

2023-2024

Objetivo



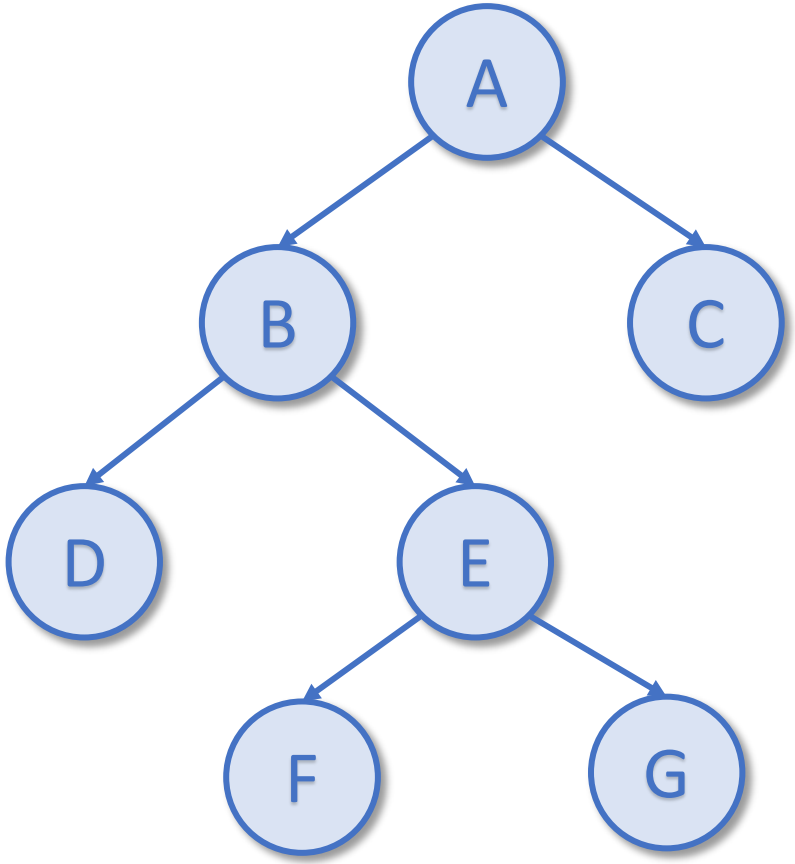
- Modelar relaciones de orden y/o de clasificación entre elementos
  - Jerarquías sociales: ejército, iglesia, organigrama empresarial …
  - Modelado de gramáticas: árboles sintácticos, árboles léxicos
  - Modelos informáticos: jerarquía de clases, sistemas de ficheros …
  - Sistemas de clasificación: rangos taxonómicos, árboles filogenéticos, genealógicos, deportivos, …

¿Qué es un árbol?

- El concepto de **árbol** implica una estructura en la que los datos se organizan de modo que los elementos de información están relacionados entre sí a través de ramas
- En **informática** llamamos **árbol** a un grafo conexo sin ciclos, con un nodo raíz y tal que sólo existe un camino desde el nodo raíz a cualquier otro nodo del árbol
- Definición **recursiva** de árbol. Un árbol es:
  - Vacío
  - O tiene un nodo determinado, llamado raíz, del que jerárquicamente descienden cero o más subárboles, que son también árboles

Elementos del un árbol

- Raíz
- Hijo (descendiente directo)
- Padre (ascendiente directo)
- Hoja (nodo terminal)
- Nodo interior
- Grado de un nodo
- Grado de un árbol
- Nivel de un nodo
- Altura (profundidad)
- Longitud del camino

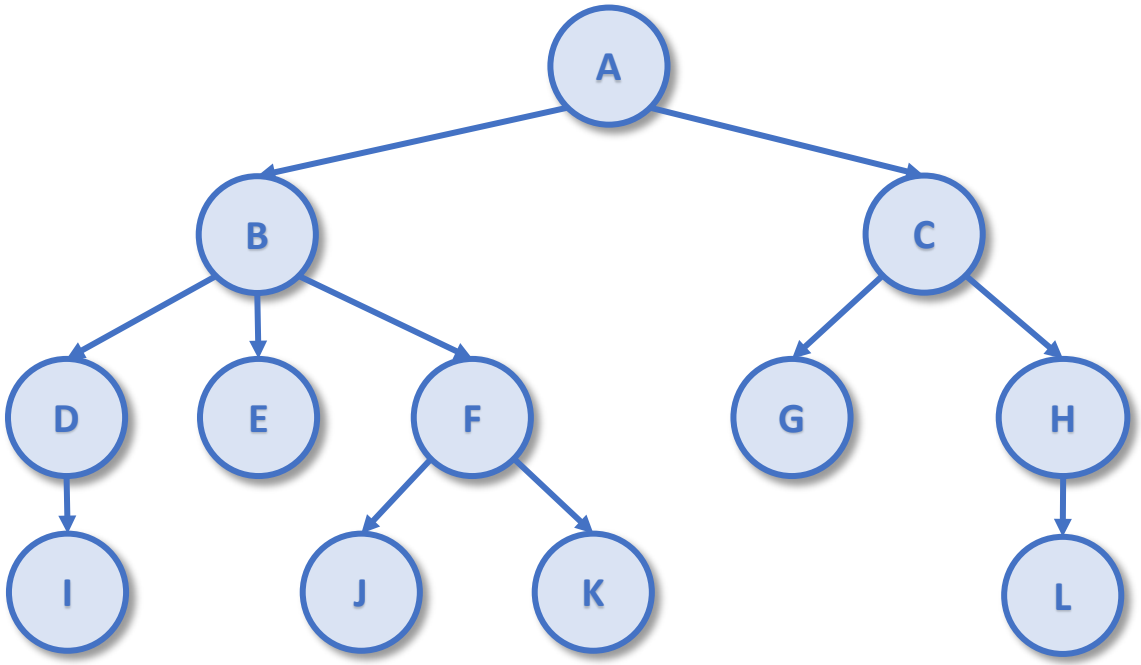


Camino Interno

- La longitud de camino interno (LCI) del árbol es la suma de las longitudes de camino de todos los nodos del árbol
- Esta medida es importante porque permite conocer, en promedio, el número de decisiones que se deben de tomar para llegar a un determinado nodo partiendo de la raíz
- Se calcula de la siguiente fórmula:
  - i → nivel del árbol
  - h → altura o profundidad del árbol
  - n_i → número de nodos del nivel

$$LCI = \sum_{i=1}^{h} n_i * i$$

Ejemplo de LCI



- LCI = $\sum_{i=1}^{h} n_i * i$
  - Nivel i=1 → 1
  - Nivel i=2 → 2
  - Nivel i=3 → 5
  - Nivel i=4 → 4
- LCI = 1*1 + 2*2 + 5*3 + 4*4
- LCI = 36
- Para 12 nodos (media)
  - LCI_m = 36 / 12 = 3

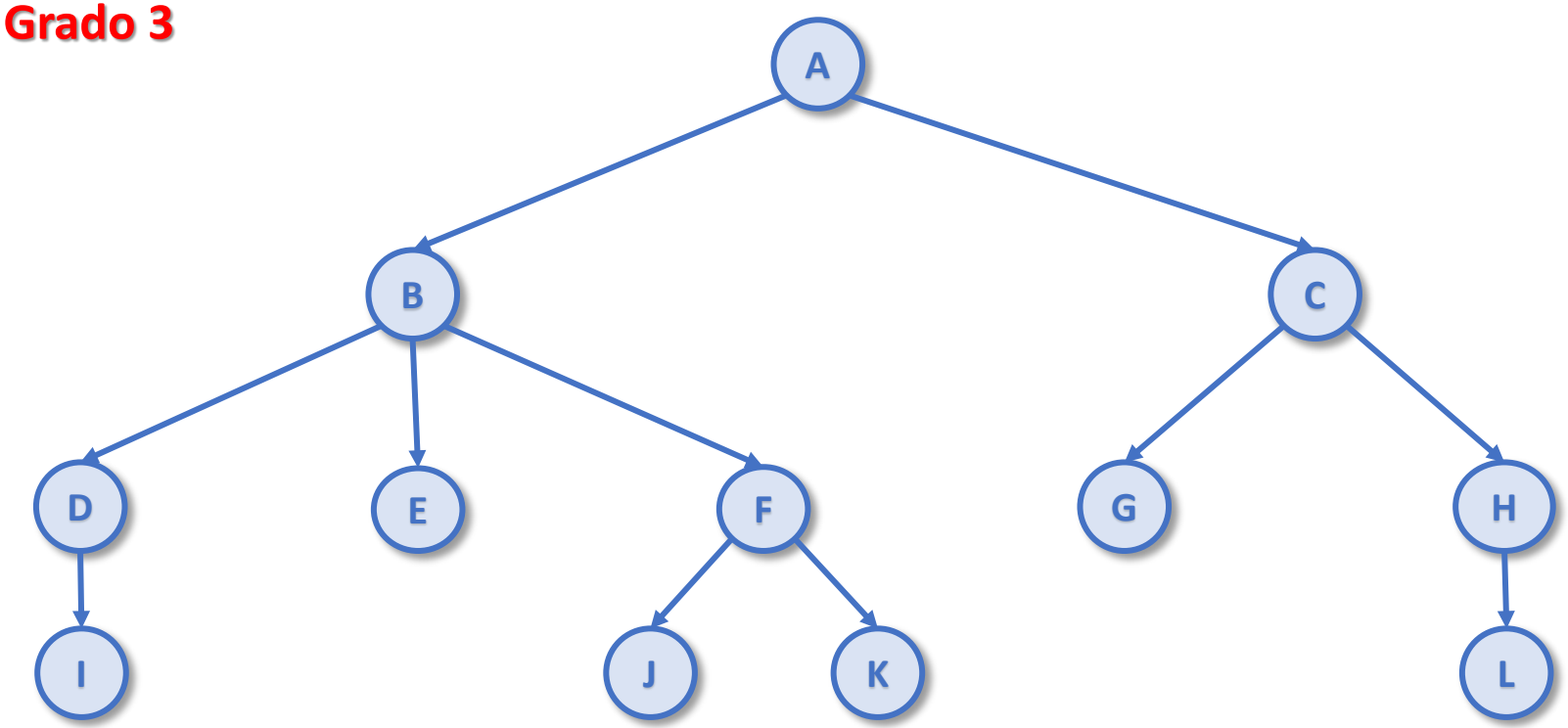
Camino Externo

- Un **árbol extendido** es aquel en el que el número de hijos de cada nodo es igual al grado del árbol. Si alguno de los nodos del árbol no cumple con esta condición entonces debe incorporarse al mismo tantos nodos especiales como se requiera para llegar a cumplirla
- Los **nodos especiales** tienen como objetivo remplazar las ramas vacías o nulas que pueden tener descendientes
- La longitud de camino externo (LCE) de un árbol es la suma de las longitudes de camino de todos los nodos especiales del árbol
- Se calcula de la siguiente fórmula:
  - i → nivel del árbol
  - h → altura o profundidad del árbol
  - n_{ei} → número de nodos especiales del nivel

$$LCE = \sum_{i=2}^{h+1} n_{ei} * i$$

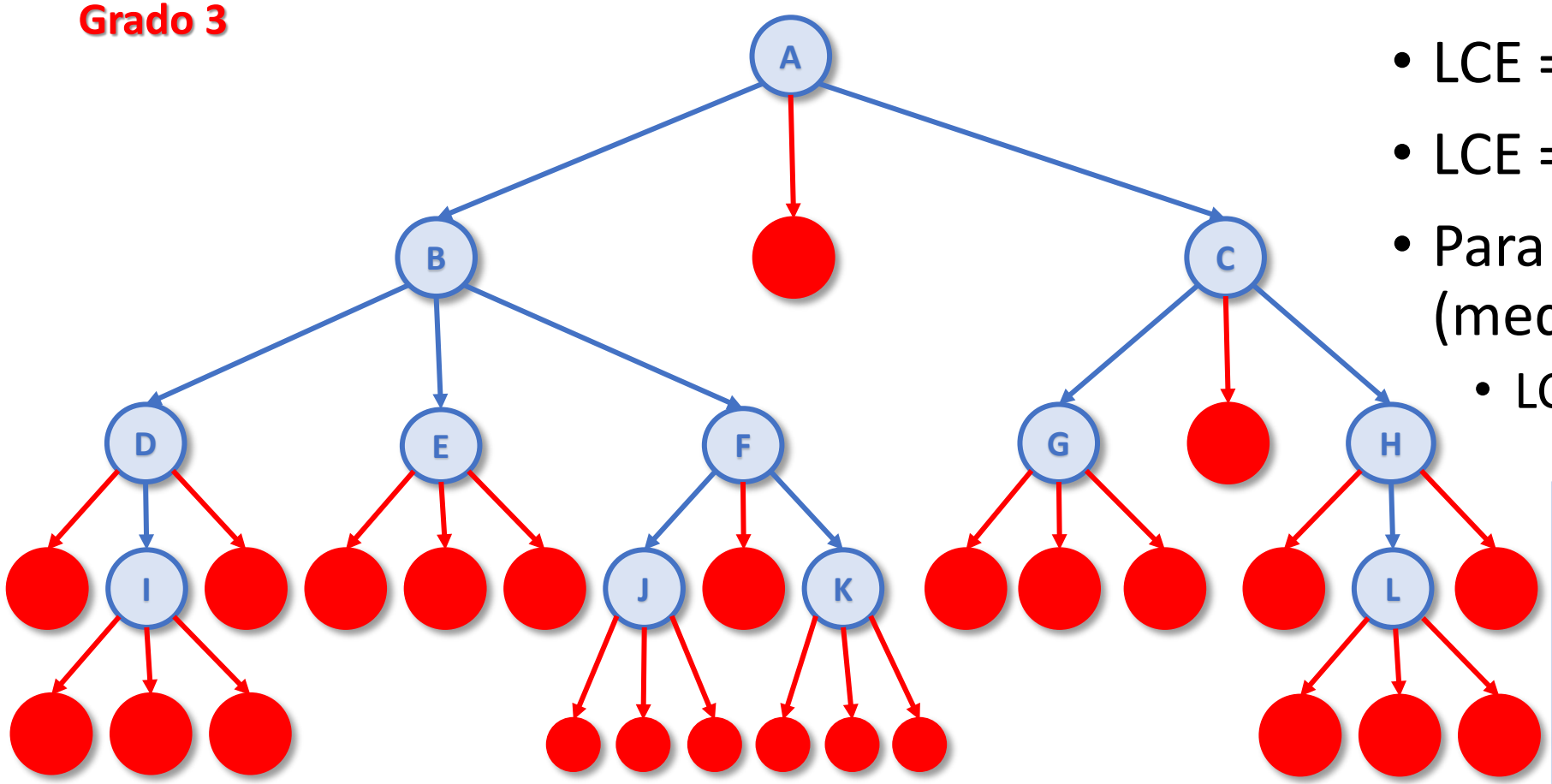
Ejemplo de LCE

Grado 3



Ejemplo de LCE

Grado 3

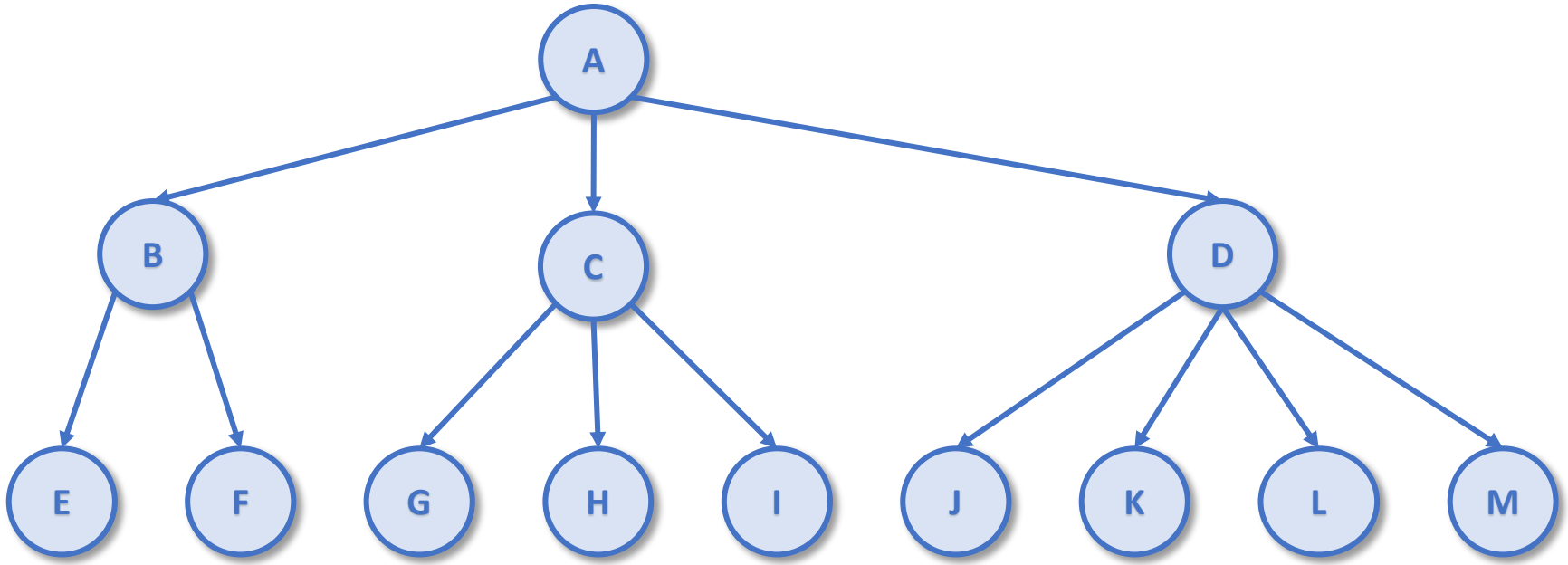


- LCE = $\sum_{i=2}^{h+1} n_{ei} * i$
  - Nivel i=2 → 1
  - Nivel i=3 → 1
  - Nivel i=4 → 11
  - Nivel i=5 → 12
- LCE = 1*2+1*3+11*4+12*5
- LCE = 109
- Para 25 nodos especiales (media)
  - LCE_m = 109 / 25 = 4.36

Esta medida es importante porque permite conocer, en promedio, el número de decisiones que se deben de tomar para llegar a un determinado nodo especial partiendo de la raíz

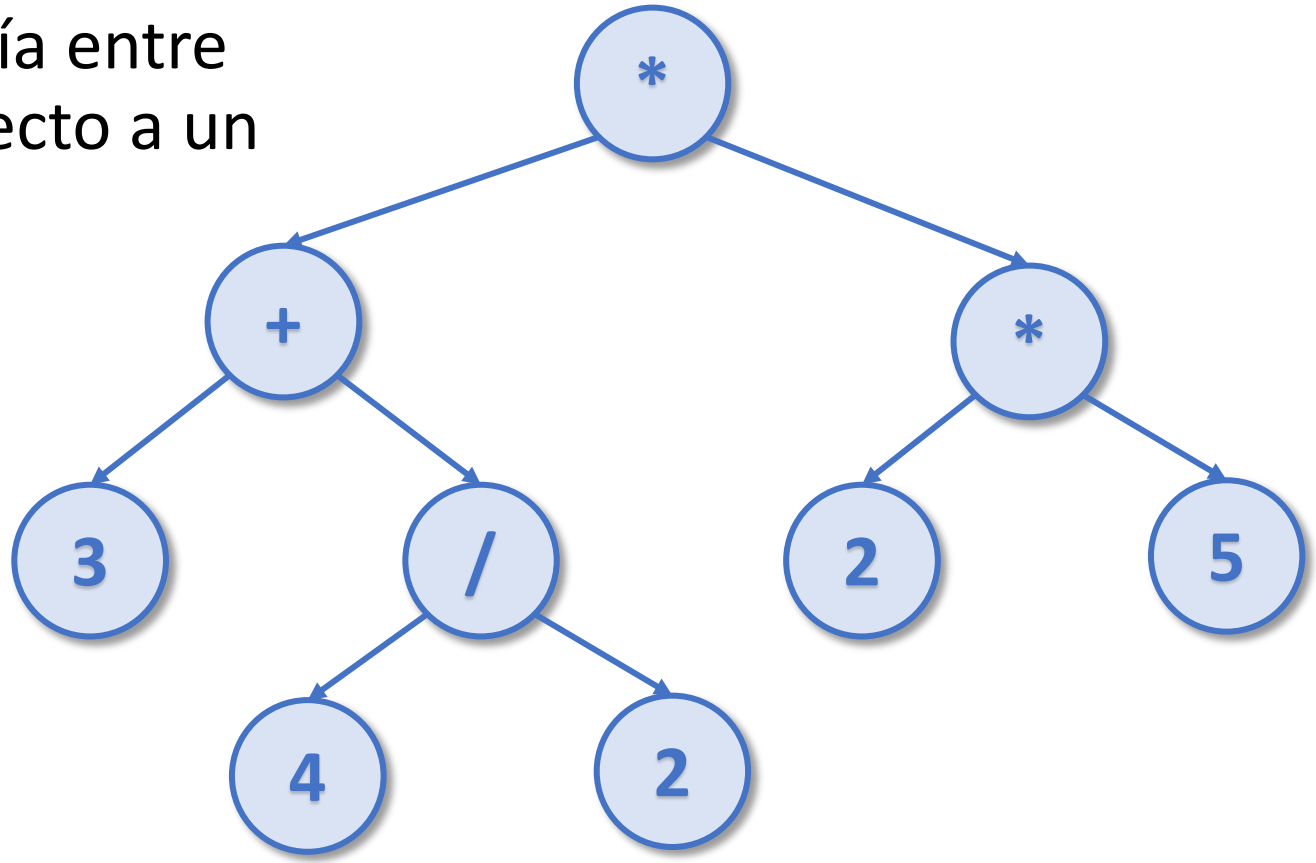
Ejercicio

- Calcular la LCI_m y la LCE_m para el siguiente árbol de grado 4



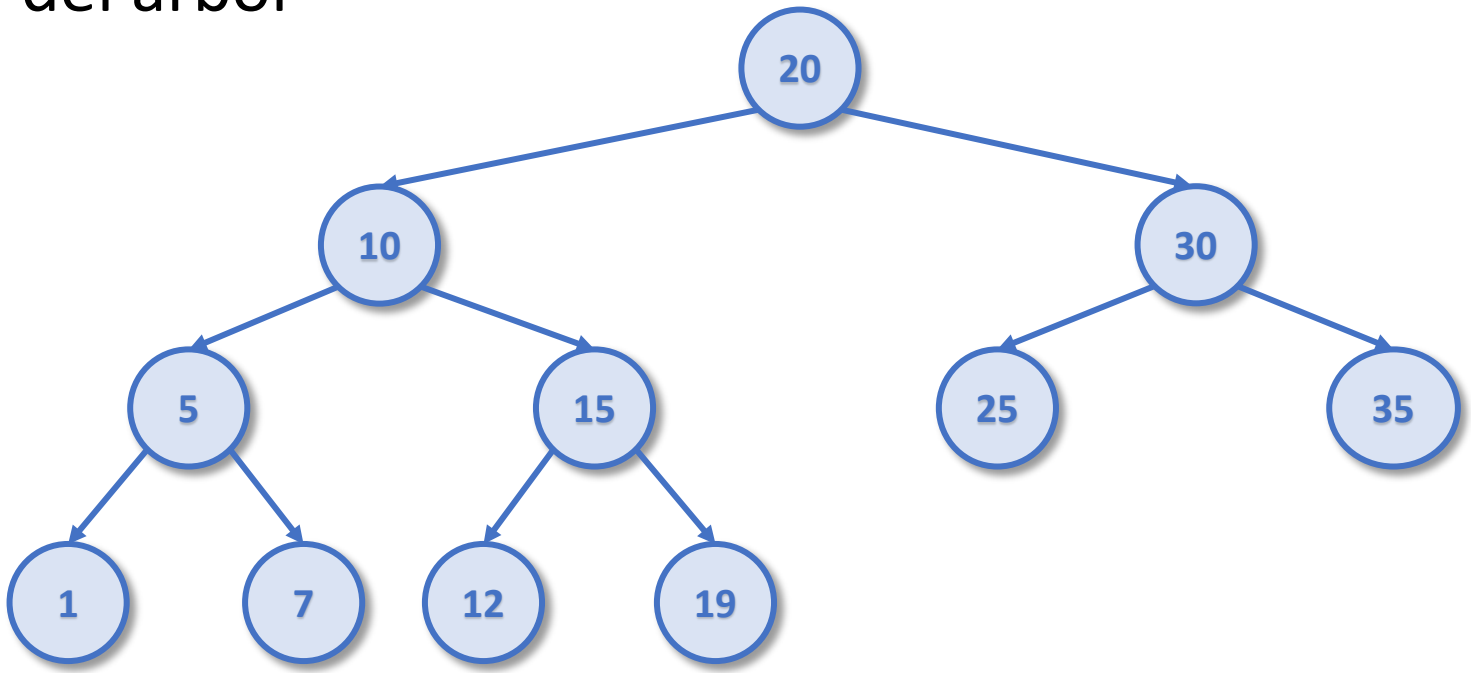
Árbol binario

- Árbol de grado 2
- Modela relaciones de jerarquía entre pares de elementos con respecto a un nodo superior
  - Árboles genealógicos
  - Competiciones de copa
  - Operaciones binarias



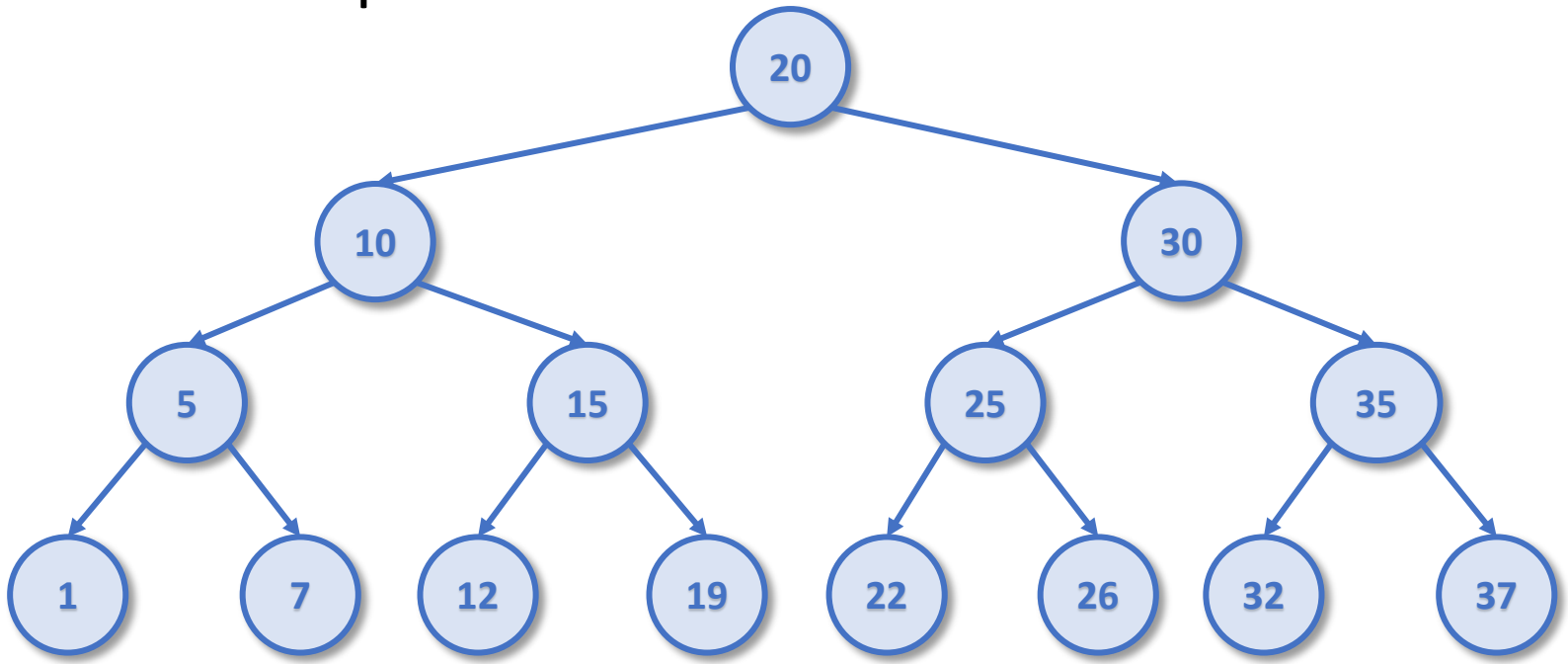
Árbol binario completo

- Un árbol binario **completo** de profundidad o altura h es un árbol en el que para cada nivel, del 0 al nivel n, tiene un conjunto lleno de nodos, y todos los nodos hoja a nivel n ocupan las posiciones más a la izquierda del árbol



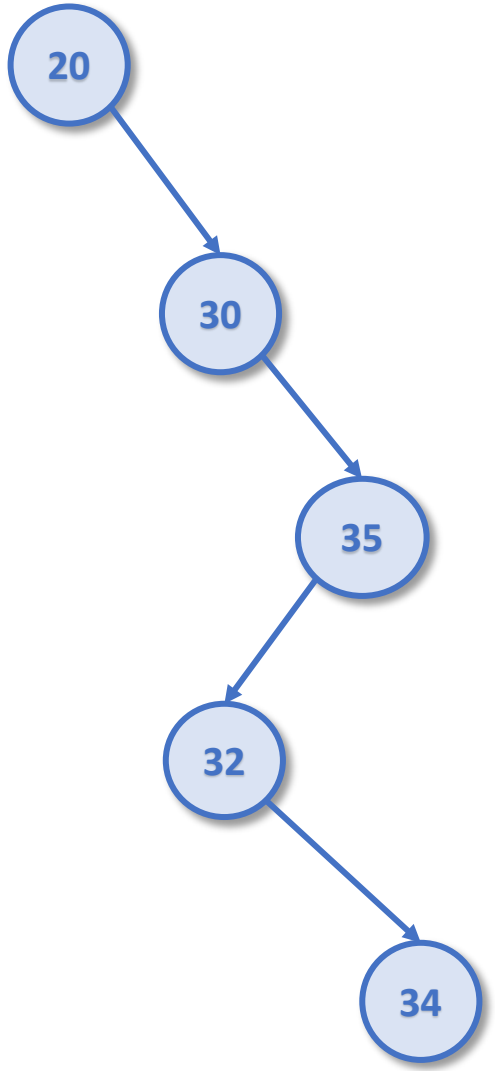
Árbol binario lleno

- Aquel que tiene el máximo número de **nodos** posibles para su **altura h** y su **grado g**
- Tiene todos los niveles llenos
- Máximo nivel de eficiencia en búsquedas desde la raíz
- Se cumple que
  - nodos = 2^h-1
  - h = $\log_2$(nodos+1)



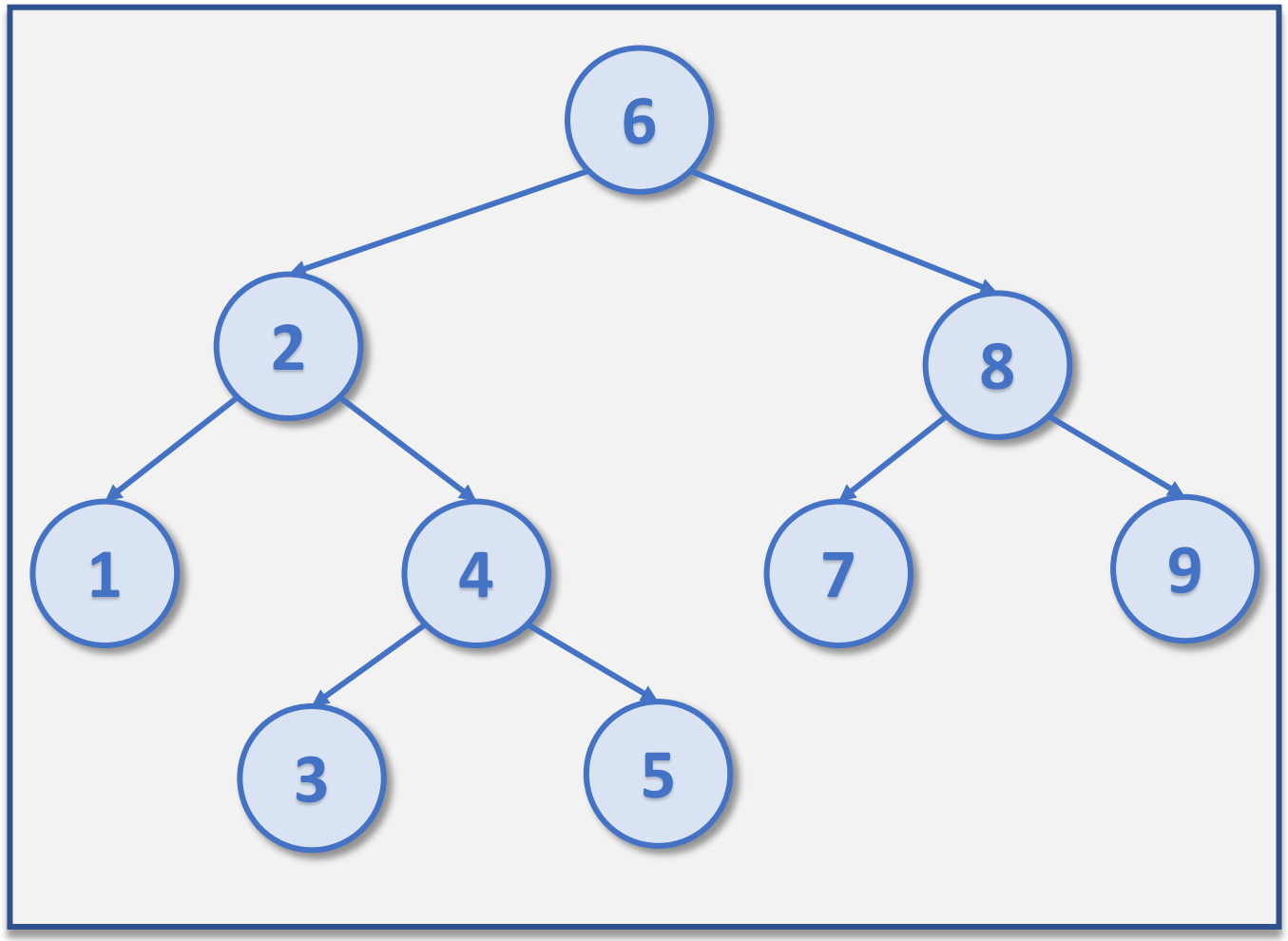
Árbol binario degenerado

- Tipo de árbol especial
- Hay un solo nodo hoja y cada nodo no hoja sólo tiene un hijo
- Un árbol degenerado es equivalente a una lista enlazada



Árbol binario de búsqueda (ABB)

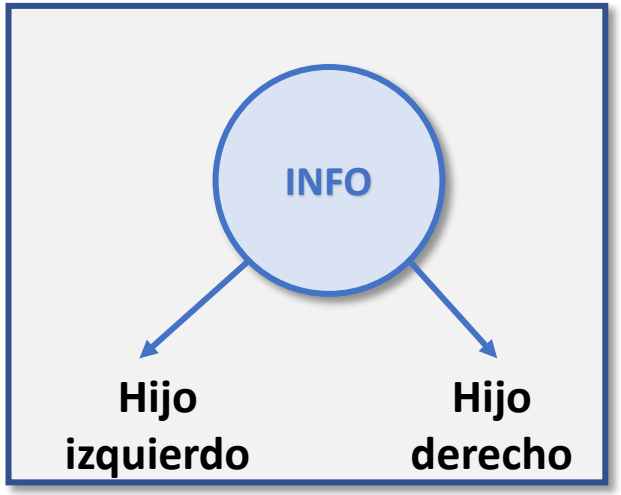
- Árbol ordenado para facilitar operaciones de búsqueda
- Para cada nodo se cumple los siguiente:
  - **Subárbol izquierdo**: contiene elementos con clave menor que la del nodo padre
  - **Subárbol derecho**: contiene elementos con clave mayor que la del nodo padre



Implementation de un ABB

La estructura de datos para definir un nodo

- Clase genérica que define un nodo de un árbol (ABB) con sus atributos y sus métodos
- Atributos que definen un nodo de un árbol
  - La **información** almacenada en el **nodo** que será de tipo genérico
  - La información sobre quien es su **hijo izquierdo**
  - La información sobre quien es su **hijo derecho**
- Métodos. Todos aquellos necesarios para gestionar los atributos del nodo



La estructura de datos para definir un árbol

- Clase genérica de un árbol de búsqueda binario (ABB) con sus atributos y sus métodos
- Atributos que definen un árbol ABB
  - El nodo raíz del árbol
- Métodos básicos (recursividad)
  - Añadir → addNode
  - Buscar → findNode
  - Borrar → removeNode
  - Mostrar el árbol → toString
- Cualquier otro método que resulte útil

ABB. Añadir una clave a un árbol

- Si el árbol está vacío crea un nuevo nodo con la información nueva y termina
- Si la clave a insertar ya existe termina
- **Método recursivo**
  - Si la clave del nodo a insertar **es menor** que la clave del nodo actual entonces
    Si el subárbol izquierdo es null entonces
    crear un nodo con la clave y asignarlo por la izquierda
    Sino volver a llamar al método insertar recursivo pero esta vez con el subárbol **izquierdo**
  - Si la clave del nodo a insertar **es mayor** que la clave del nodo actual entonces
    Si el subárbol derecho es null entonces
    crear un nodo con la clave y asignarlo por la derecha
    Sino volver a llamar al método insertar recursivo pero esta vez con el subárbol **derecho**
- **¿Complejidad del algoritmo?**

ABB. Añadir. Ejercicios

- **Ejercicio1**
  - Partiendo de un ABB vacío insertar la secuencia de claves: 5, 7, 9, 3, 1, 2, 6
  - ¿Cuál es la complejidad temporal de cada inserción?

- **Ejercicio2**
  - Partiendo de un ABB vacío insertar la secuencia de claves: 7,6, 5, 4, 3, 2, 1
  - ¿Cuál es la complejidad temporal de cada inserción?
  - Insertar el 8
  - ¿Cuál es la complejidad temporal de esta inserción?

ABB. Buscar una clave en un árbol

- **Casos básicos o de parada**
  - Si la clave del nodo a buscar es igual que la clave del nodo actual, la clave ya se ha encontrado y termina
  - Si el nodo actual es null significa que se ha alcanzado una hoja por lo tanto no se ha encontrado la clave a buscar y termina
- **Caso recursivo**
  - Si la clave del nodo a buscar **es menor** que la clave del nodo actual, volver a llamar al método buscar recursivo pero esta vez con el subárbol **izquierdo**
  - Si la clave del nodo a buscar **es mayor** que la clave del nodo actual, volver a llamar al método buscar recursivo pero esta vez con el subárbol **derecho**
- **¿Complejidad del algoritmo?**

ABB. Eliminar una clave del árbol

- **Casos básicos o de parada**
  - Si la clave del nodo a buscar no existe, termina
  - Si el nodo actual es null significa que el árbol esta vacío y termina
  - Si la clave del nodo actual es igual a la clase del nodo a eliminar entonces ***proceso de eliminación***
- **Caso recursivo**
  - Si la clave del nodo a buscar **es menor** que la clave del nodo actual, volver a llamar al método eliminar recursivo pero esta vez con el subárbol **izquierdo**
  - Si la clave del nodo a buscar **es mayor** que la clave del nodo actual, volver a llamar al método eliminar recursivo pero esta vez con el subárbol **derecho**
- **¿Complejidad del algoritmo?**

ABB. Proceso de eliminación. Caso I

- El nodo que contiene la clave a borrar **no tiene hijos** (derecho e izquierdo)
  - Devolver **null** para que se le asigne al padre (dejará de tener hijo)

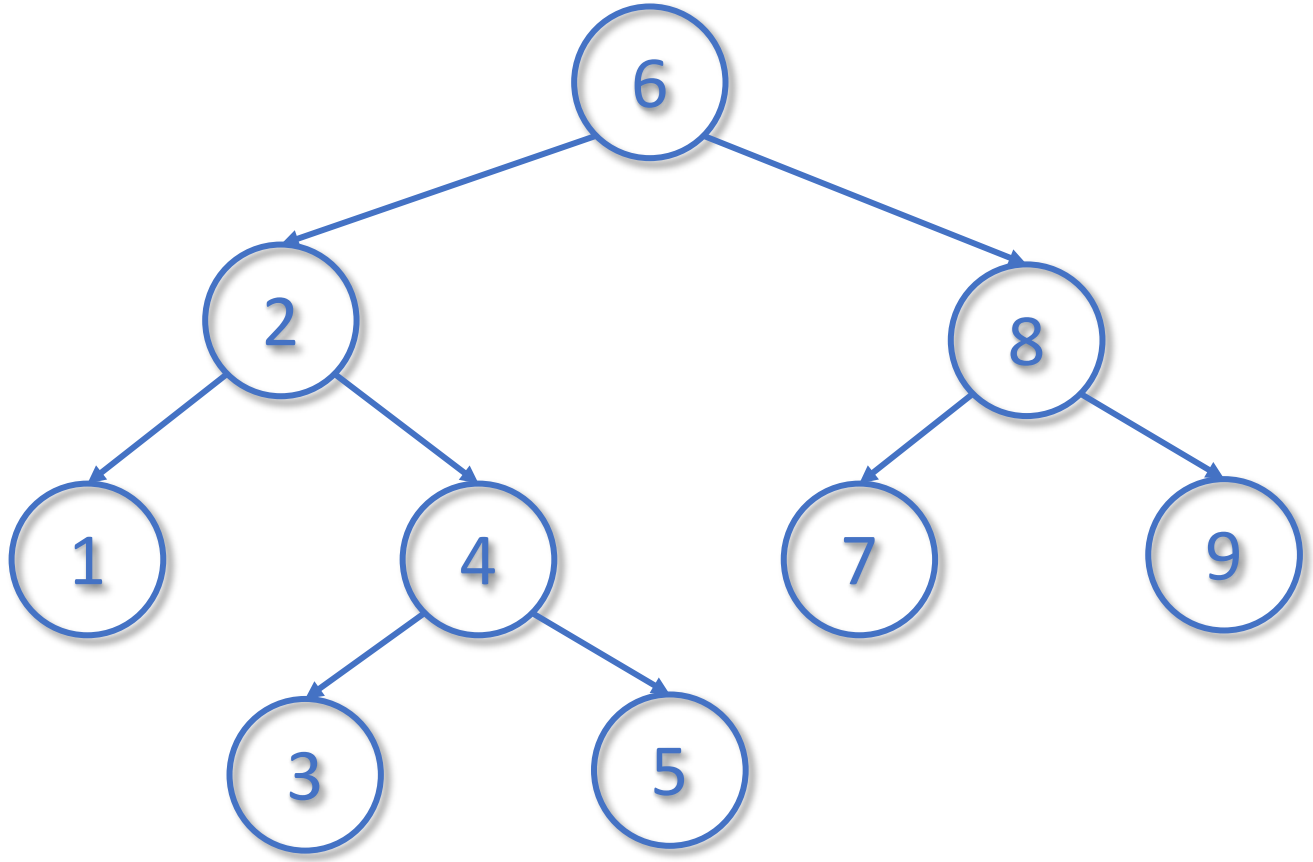


ABB. Proceso de eliminación. Caso I

- El nodo que contiene la clave a borrar **no tiene hijos** (derecho e izquierdo)
  - Devolver **null** para que se le asigne al padre (dejará de tener hijo)

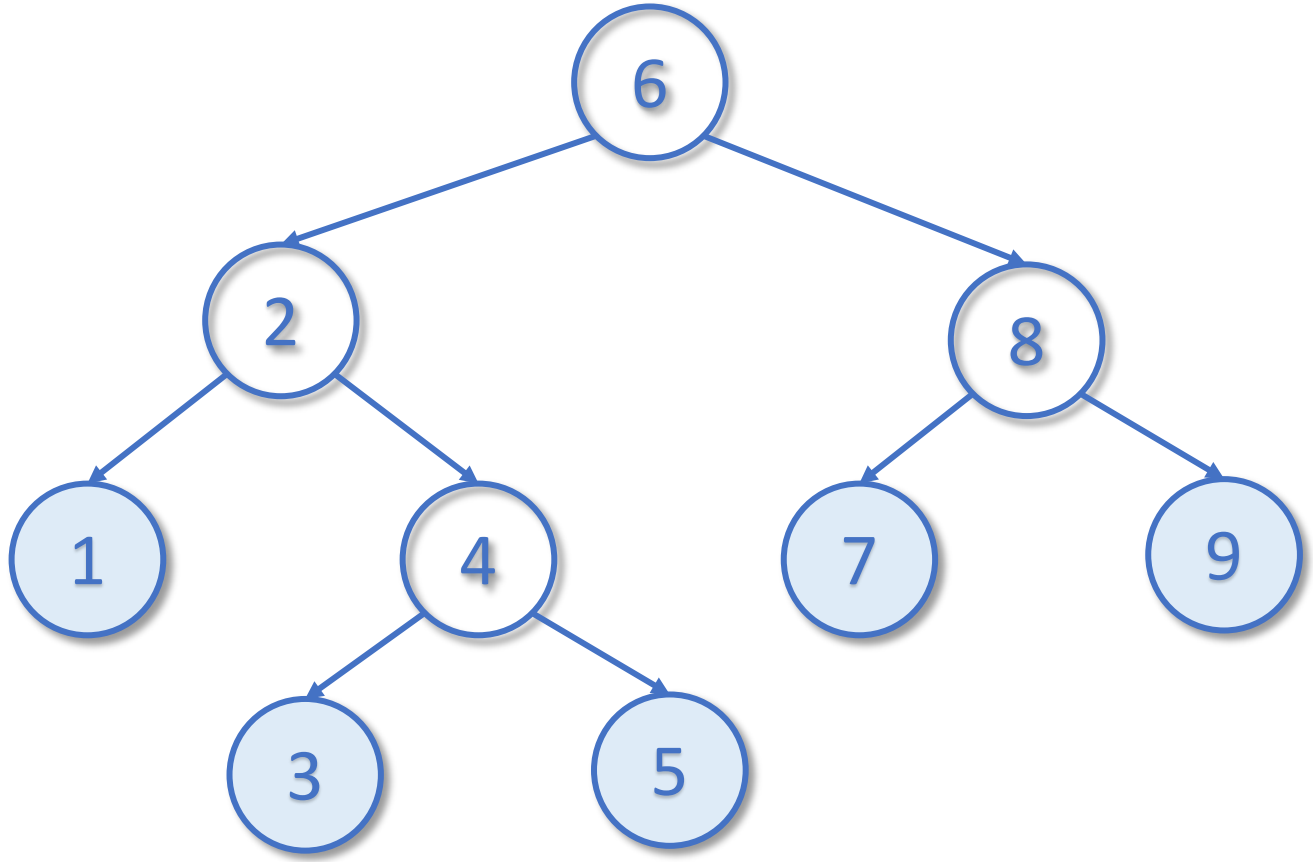


ABB. Proceso de eliminación. Caso I

- El nodo que contiene la clave a borrar **no tiene hijos** (derecho e izquierdo)
  - Devolver **null** para que se le asigne al padre (dejará de tener hijo)

- Ejemplo. **Borrar el 5**

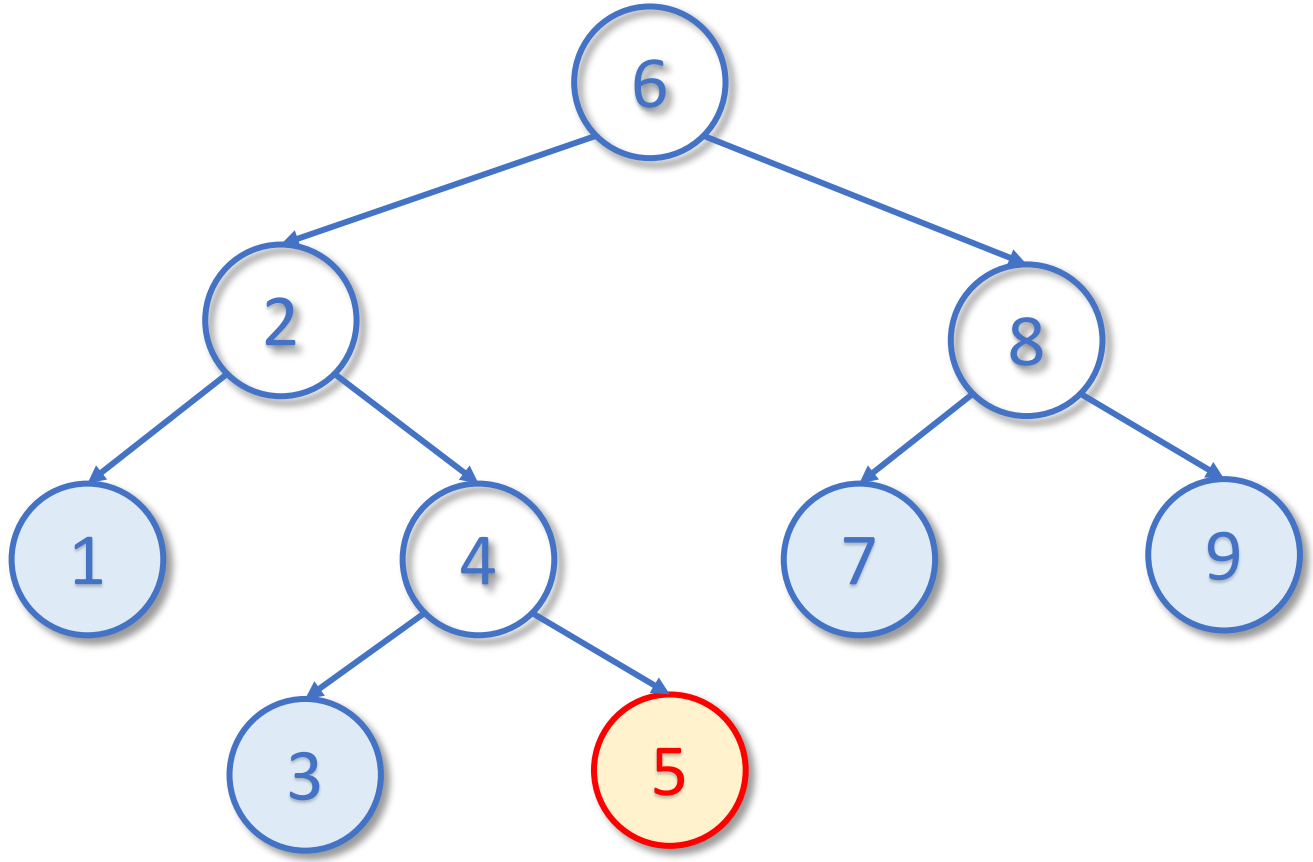


ABB. Proceso de eliminación. Caso I

- El nodo que contiene la clave a borrar **no tiene hijos** (derecho e izquierdo)
  - Devolver **null** para que se le asigne al padre (dejará de tener hijo)

- Ejemplo. **Borrar el 5**

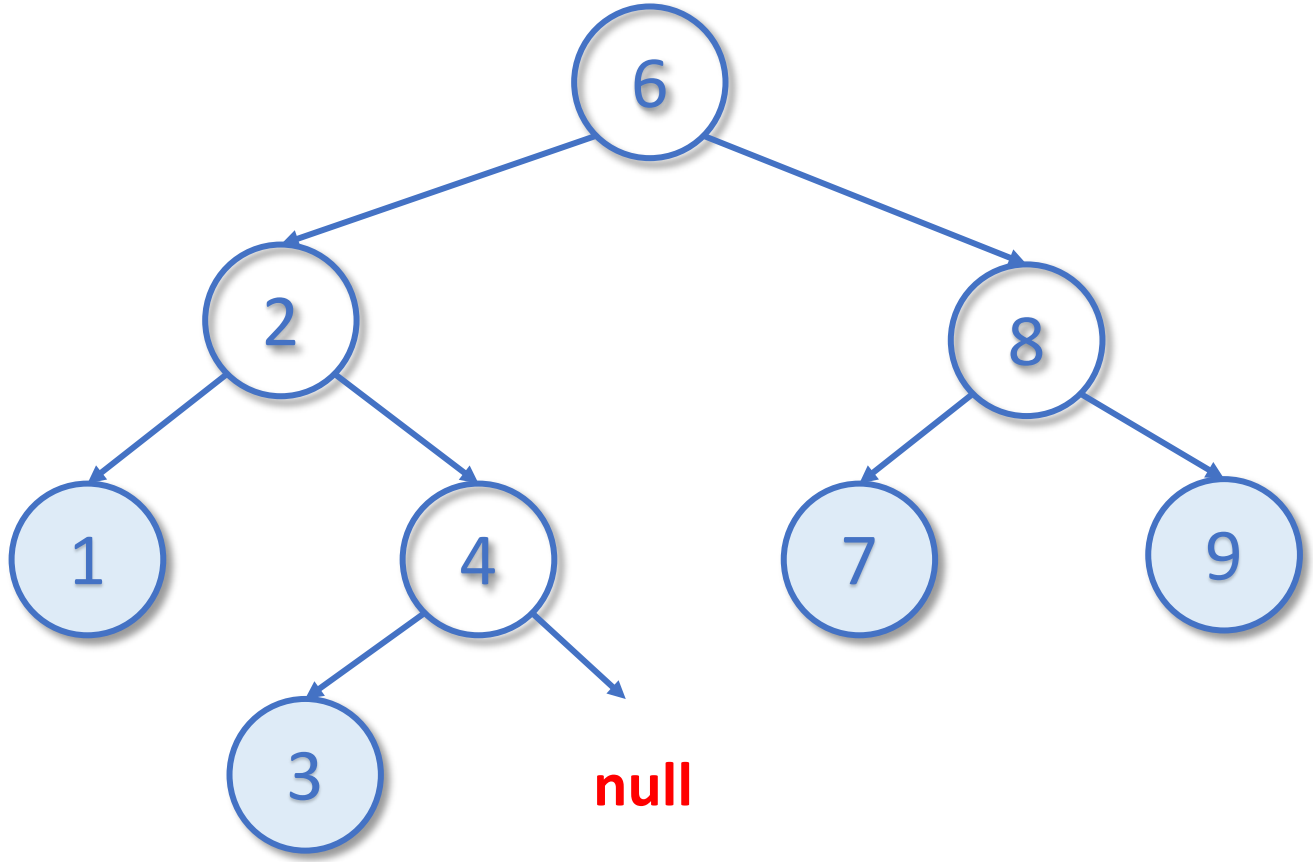


ABB. Proceso de eliminación. Caso I

- El nodo que contiene la clave a borrar **no tiene hijos** (derecho e izquierdo)
  - Devolver **null** para que se le asigne al padre (dejará de tener hijo)

- Ejemplo. **Borrar el 5**

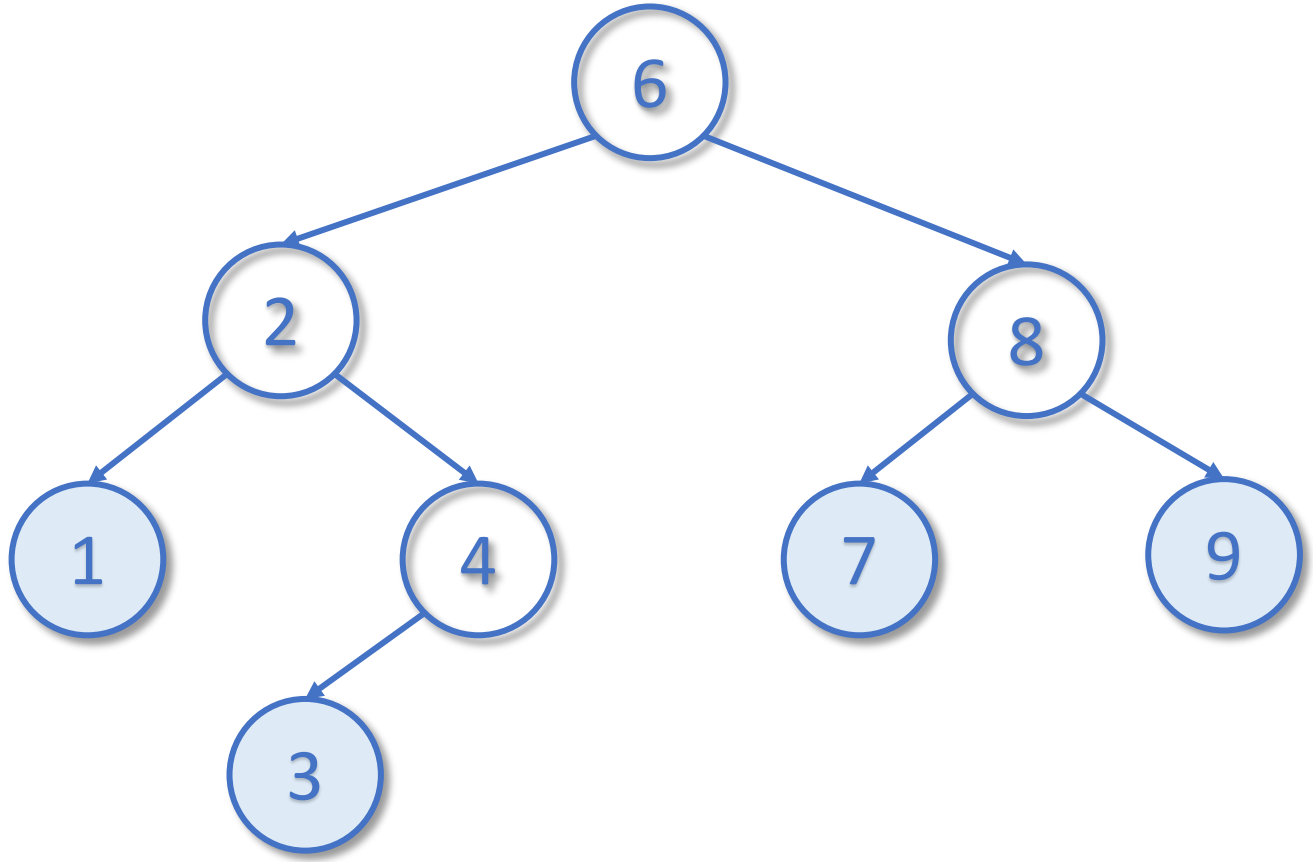


ABB. Proceso de eliminación. Caso II

- El nodo que contiene la clave a borrar **tiene un solo hijo** (derecho o izquierdo)
  - Devolver la referencia al único hijo que tiene (por la derecha o por la izquierda) para que se le asigne al padre

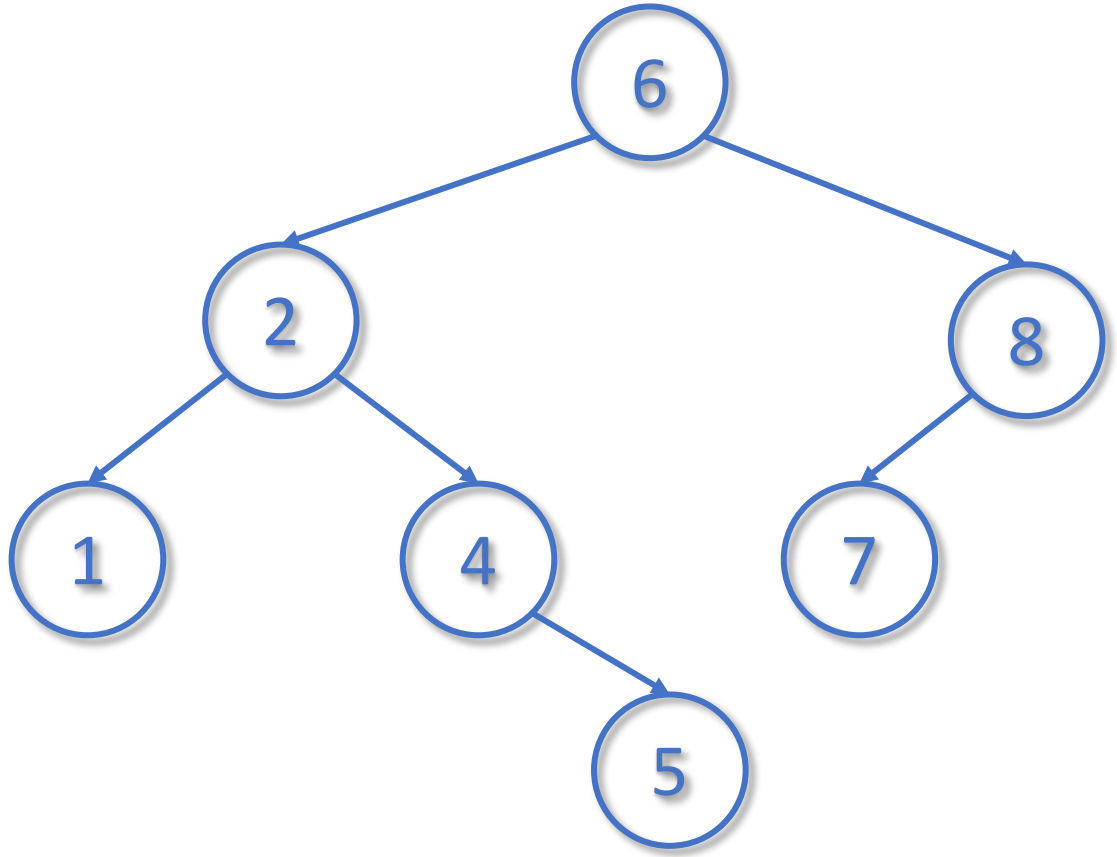


ABB. Proceso de eliminación. Caso II

- El nodo que contiene la clave a borrar **tiene un solo hijo** (derecho o izquierdo)
  - Devolver la referencia al único hijo que tiene (por la derecha o por la izquierda) para que se le asigne al padre

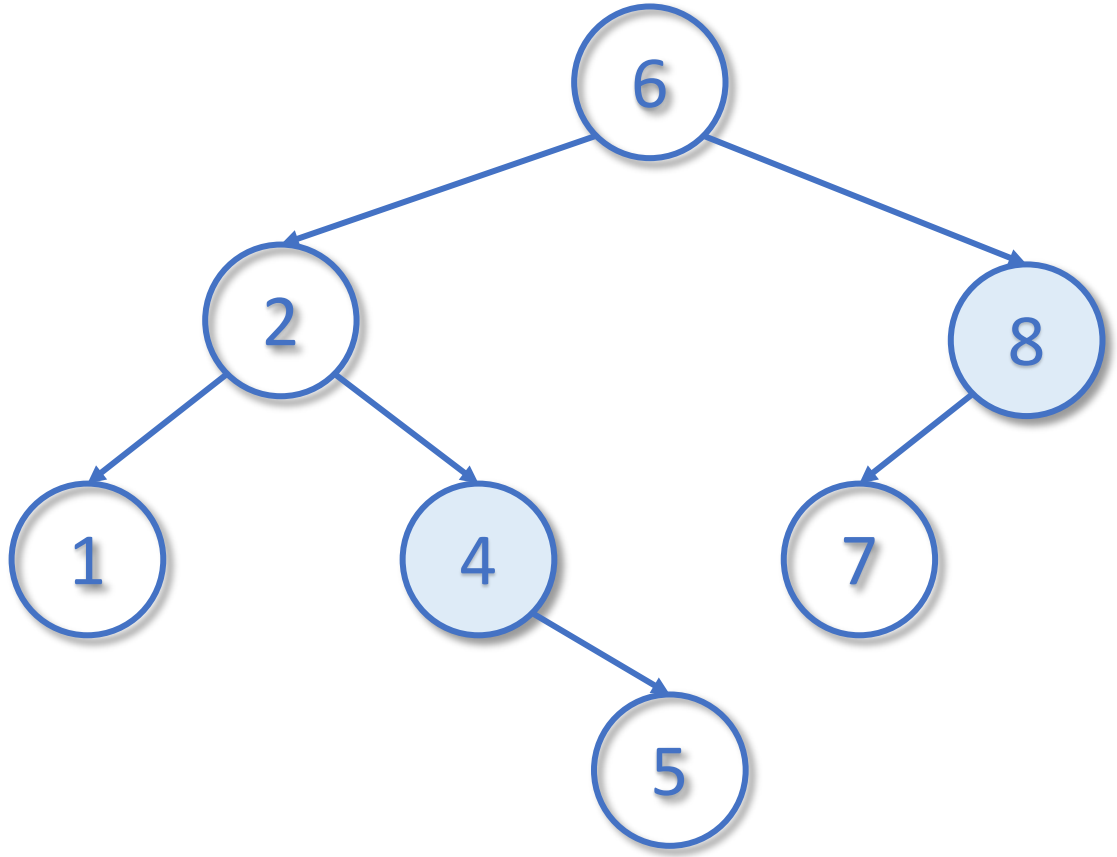


ABB. Proceso de eliminación. Caso II

- El nodo que contiene la clave a borrar **tiene un solo hijo** (derecho o izquierdo)
  - Devolver la referencia al único hijo que tiene (por la derecha o por la izquierda) para que se le asigne al padre

- Ejemplo. **Borrar el 4**

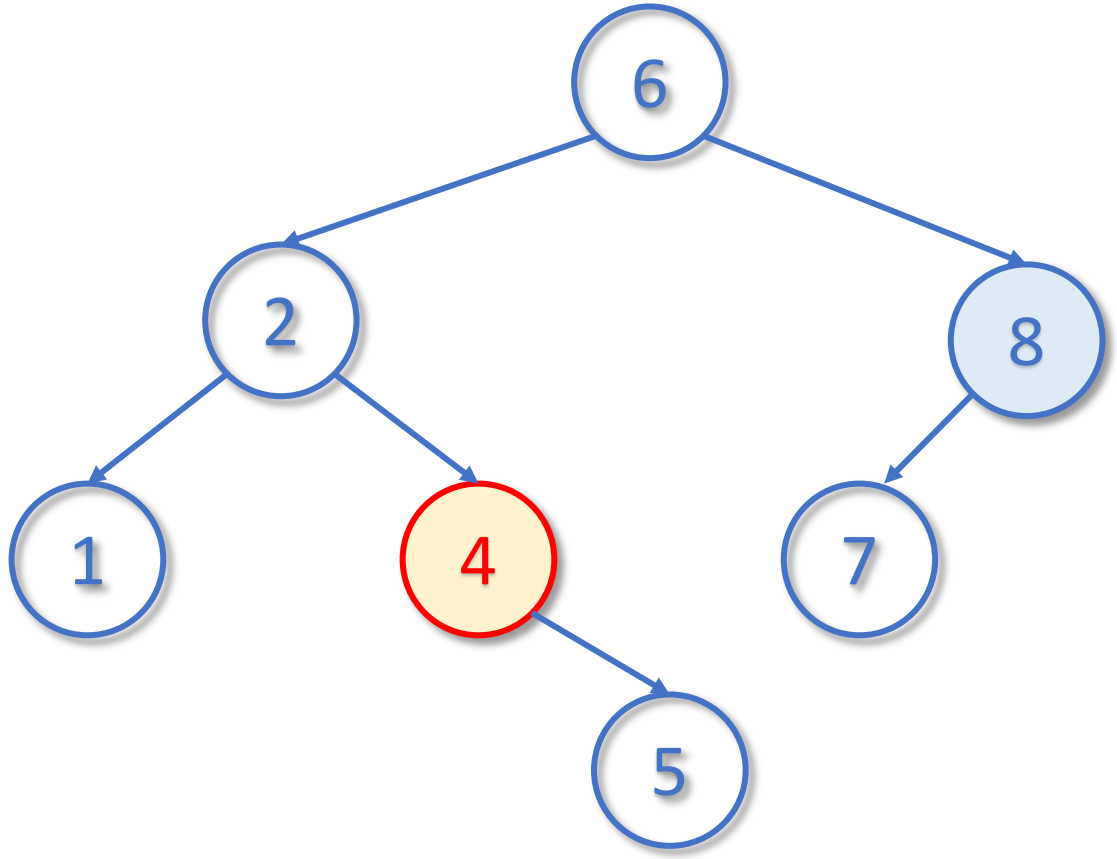


ABB. Proceso de eliminación. Caso II

- El nodo que contiene la clave a borrar **tiene un solo hijo** (derecho o izquierdo)
  - Devolver la referencia al único hijo que tiene (por la derecha o por la izquierda) para que se le asigne al padre

- Ejemplo. **<u>Borrar el 4</u>**

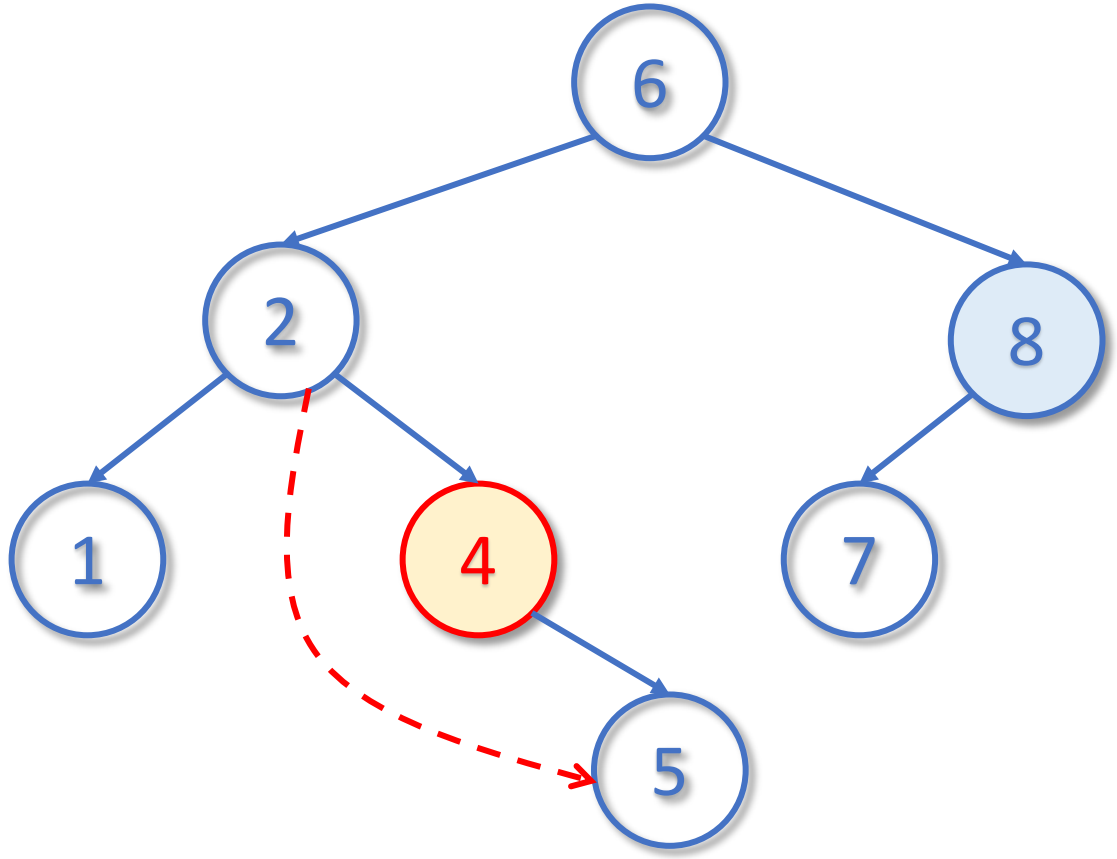
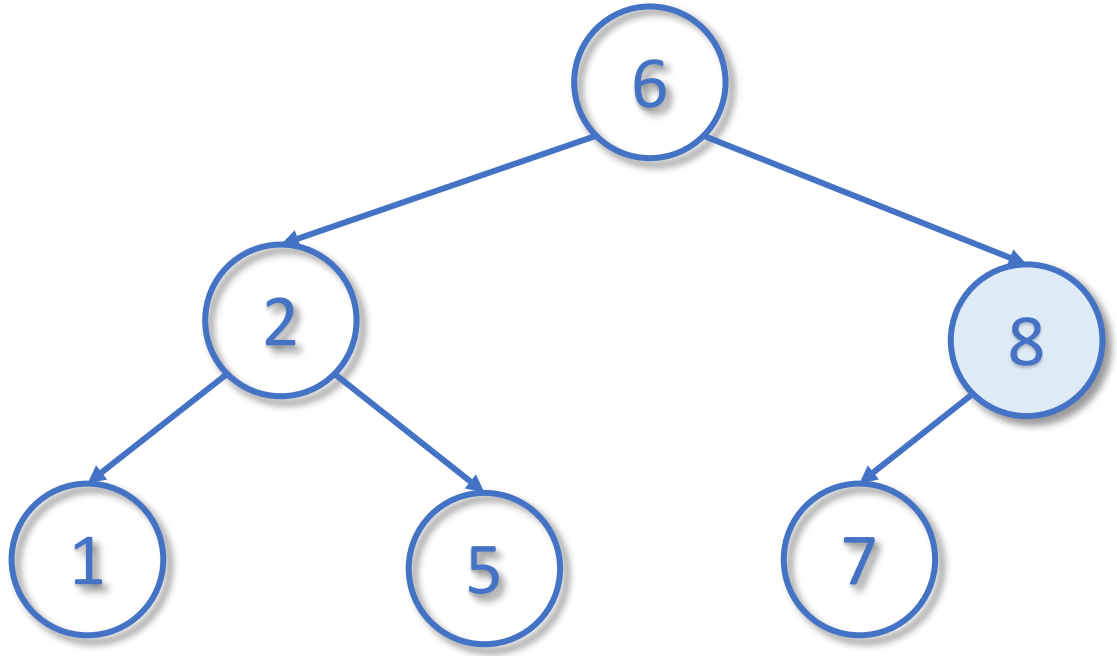


ABB. Proceso de eliminación. Caso II

- El nodo que contiene la clave a borrar **tiene un solo hijo** (derecho o izquierdo)
  - Devolver la referencia al único hijo que tiene (por la derecha o por la izquierda) para que se le asigne al padre



- Ejemplo. **Borrar el 4**

ABB. Proceso de eliminación. Caso III

- El nodo que contiene la clave a borrar **tiene dos hijos** (derecho e izquierdo)
  - Buscar el nodo que tenga la clave máxima del subárbol izquierdo del nodo a eliminar
  - Sustituir la clave del nodo a borra por ese máximo
  - Borrar el nodo cuya clave es el máximo encontrado
  - Devolver el subárbol modificado

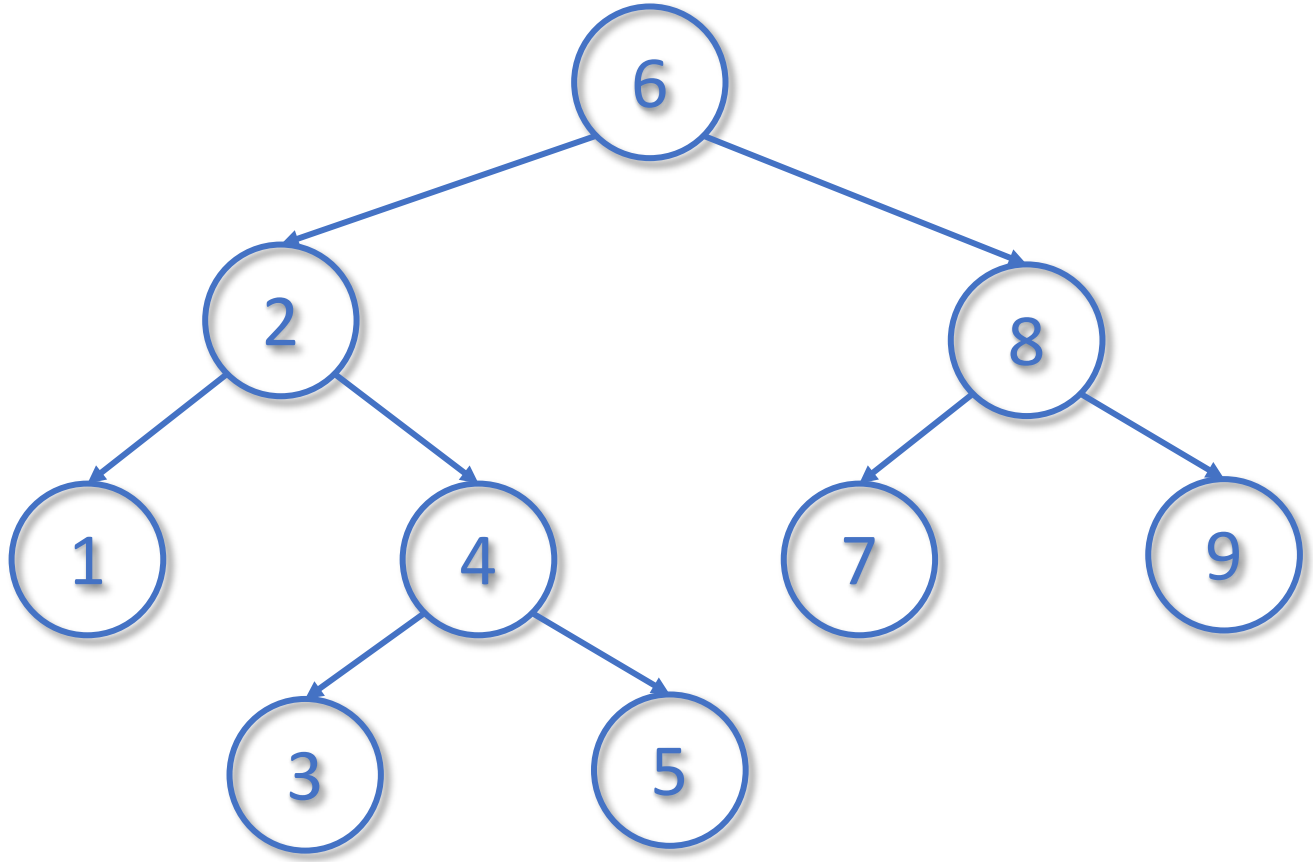


ABB. Proceso de eliminación. Caso III

- El nodo que contiene la clave a borrar **tiene dos hijos** (derecho e izquierdo)
  - Buscar el nodo que tenga la clave máxima del subárbol izquierdo del nodo a eliminar
  - Sustituir la clave del nodo a borra por ese máximo
  - Borrar el nodo cuya clave es el máximo encontrado
  - Devolver el subárbol modificado

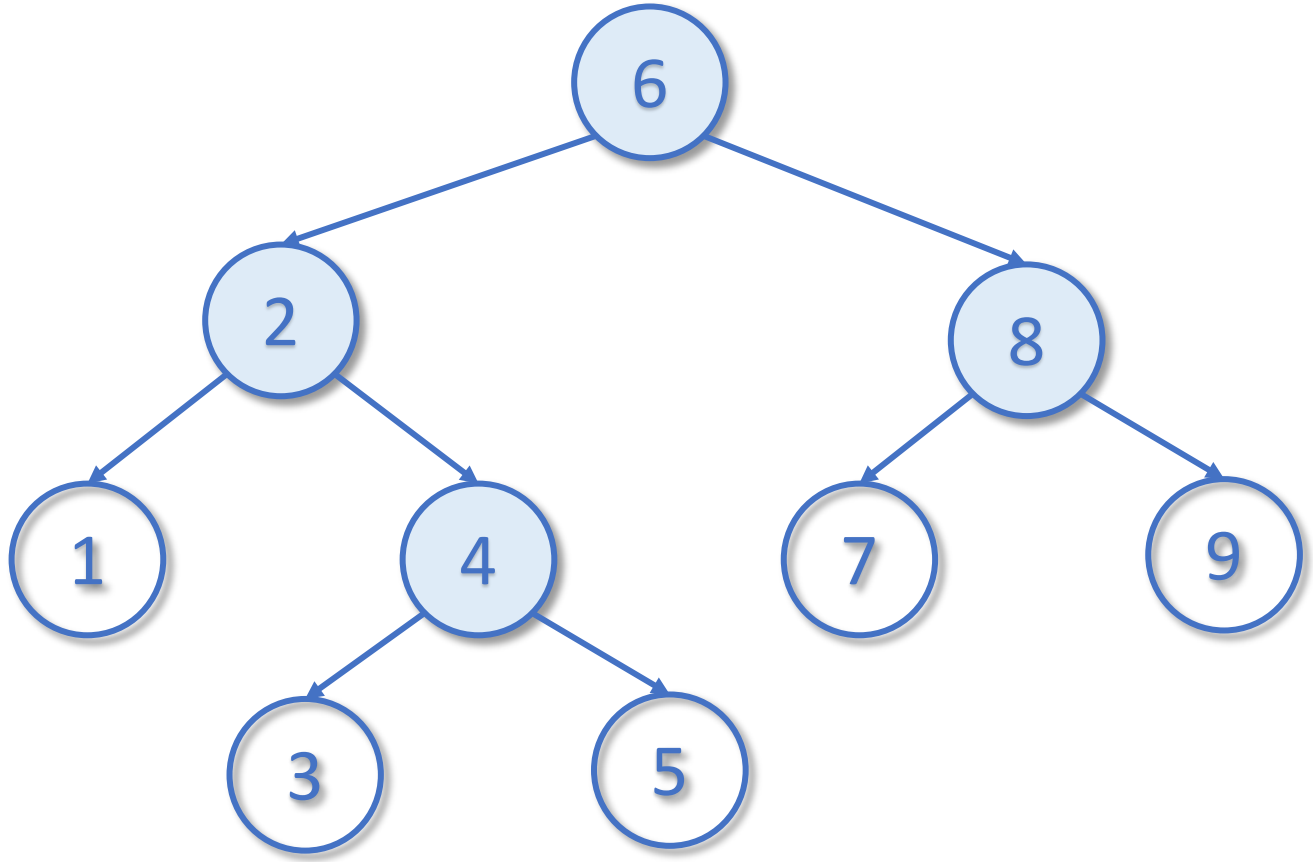


ABB. Proceso de eliminación. Caso III

- El nodo que contiene la clave a borrar **tiene dos hijos** (derecho e izquierdo)
  - Buscar el nodo que tenga la clave máxima del subárbol izquierdo del nodo a eliminar
  - Sustituir la clave del nodo a borra por ese máximo
  - Borrar el nodo cuya clave es el máximo encontrado
  - Devolver el subárbol modificado
- Ejemplos. **Borrar el 6**

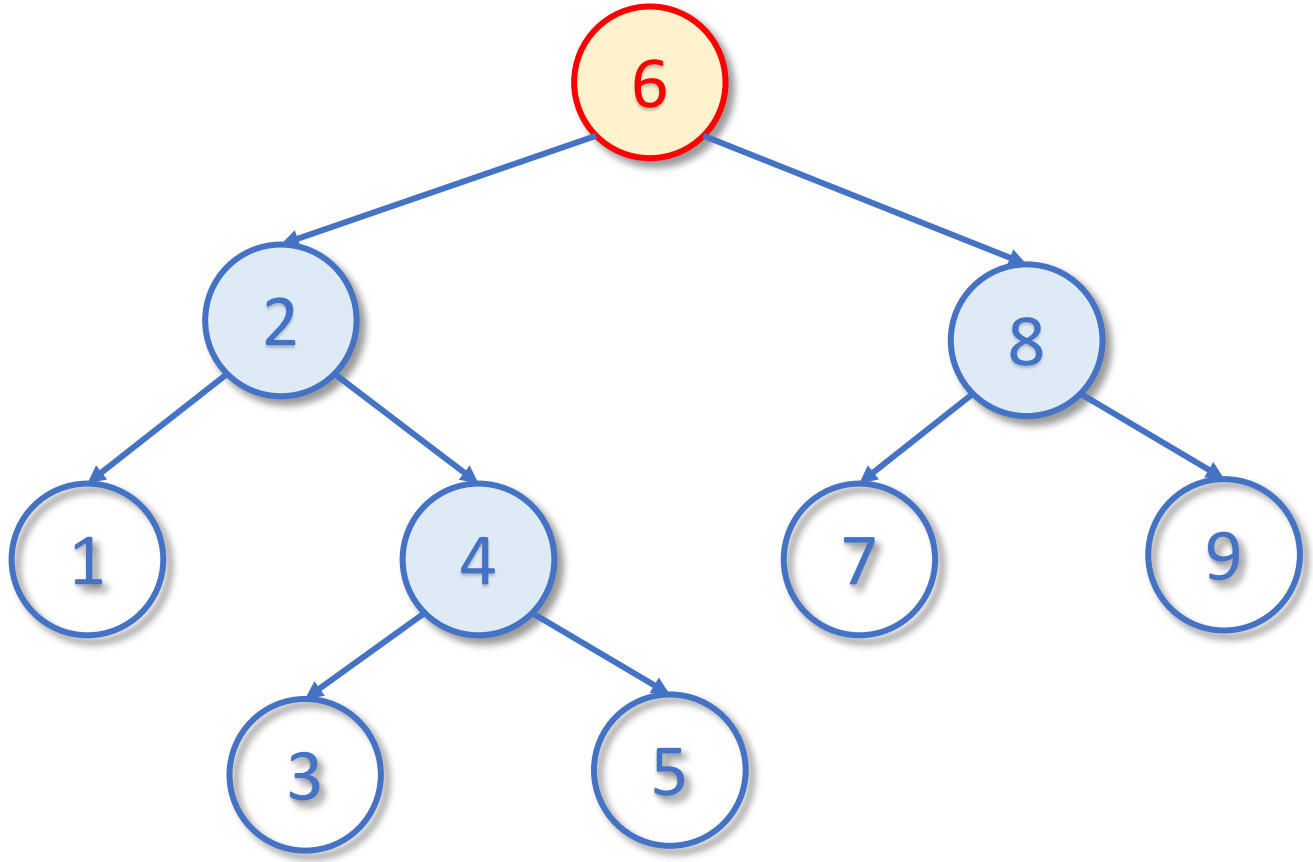


ABB. Proceso de eliminación. Caso III

- El nodo que contiene la clave a borrar **tiene dos hijos** (derecho e izquierdo)
  - Buscar el nodo que tenga la clave máxima del subárbol izquierdo del nodo a eliminar
  - Sustituir la clave del nodo a borra por ese máximo
  - Borrar el nodo cuya clave es el máximo encontrado
  - Devolver el subárbol modificado
- Ejemplos. **Borrar el 6**

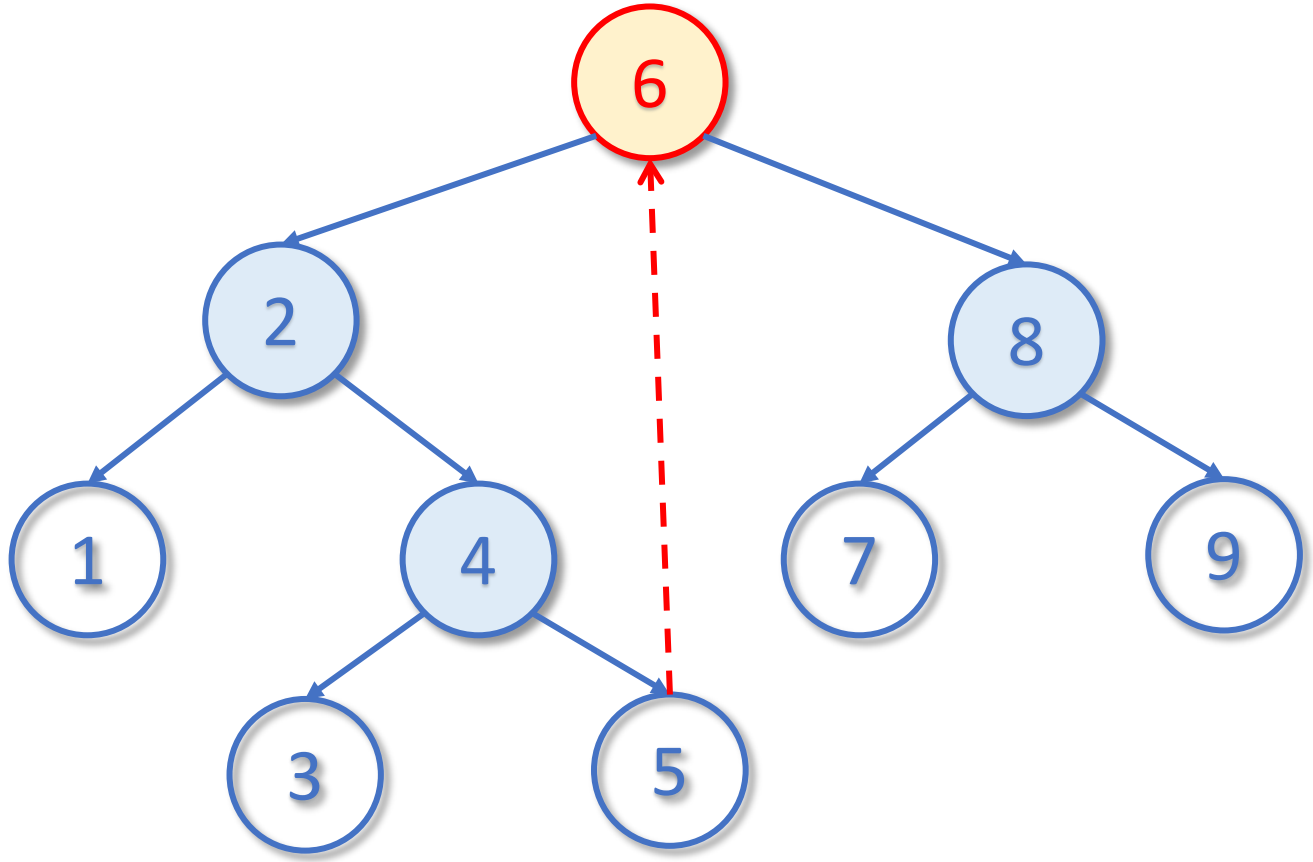


ABB. Proceso de eliminación. Caso III

- El nodo que contiene la clave a borrar **tiene dos hijos** (derecho e izquierdo)
  - Buscar el nodo que tenga la clave máxima del subárbol izquierdo del nodo a eliminar
  - Sustituir la clave del nodo a borra por ese máximo
  - Borrar el nodo cuya clave es el máximo encontrado
  - Devolver el subárbol modificado
- Ejemplos. **Borrar el 6**

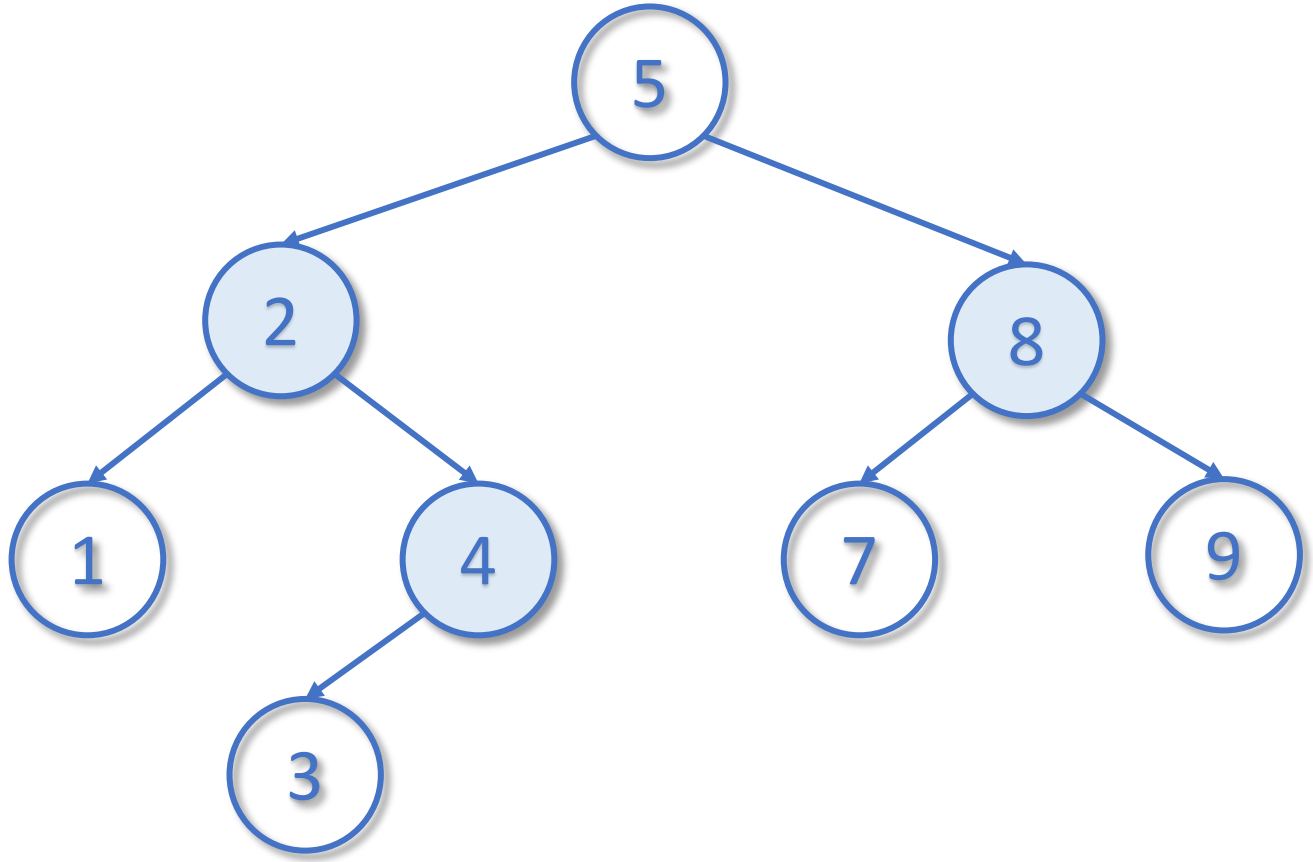


ABB. Buscar el máximo

- Versión iterativa
  - Partiendo del nodo a borrar
  - Mientras el nodo por la derecha sea distinto de null
    - Actualizar el nodo con el subárbol derecho
  - Devolver contenido del nodo que no tenga hijos por la derecha
- Versión recursiva
  - Partiendo del nodo a borrar
  - Si el nodo es null devolvemos null
  - Si el nodo no tiene hijo derecho devolvemos el contenido del nodo
  - Si el nodo tiene hijo por la derecha seguir buscando por la derecha

ABB. Recorridos

- Tres recorridos
  - Preorden (raíz, izquierdo, derecho)
    - Visitar la raíz
    - Explorar el subárbol izquierdo
    - Explorar el subárbol derecho
  - Inorden (izquierdo, raíz, derecho)
    - Explorar el subárbol izquierdo
    - Visitar la raíz
    - Explorar el subárbol derecho
  - PostOrden (izquierdo, derecho, raíz)
    - Explorar el subárbol izquierdo
    - Explorar el subárbol derecho
    - Visitar la raíz

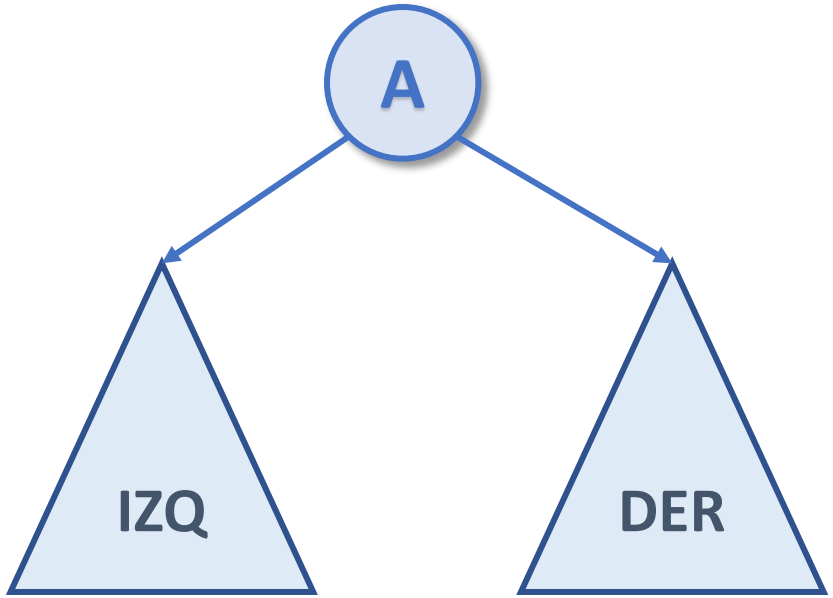


ABB. Recorridos. Implementación

- Método recursivo
- Caso básico
  - Si el nodo es null termina
- Caso recursivo
  - Si el nodo no es null
    - Per orden → trata el nodo, examina recursivamente el subárbol izquierdo, examina recursivamente el subárbol derecho
    - In orden → examina recursivamente el subárbol izquierdo, trata el nodo, examina recursivamente el subárbol derecho
    - Post orden → examina recursivamente el subárbol izquierdo, examina recursivamente el subárbol derecho, trata el nodo
- Complejidad → O(n)

ABB. Recorridos. Ejercicios

- Construir un ABB con las claves 50, 25, 75, 10, 40, 60, 90, 35, 45, 70, 42
- Una vez construido indicar:
  - Altura o profundidad
  - ¿Es completo?
  - ¿Esta lleno?
  - ¿Es degenerado?
  - Recorrido en preorden
  - Recorrido en inorden
  - Recorrido en postorden

ABB. Eficiencia

- Depende de su altura que va de [$\log_2 n$, n]
  - Altura mínima → $\log_2 n$
  - Altura máxima (árbol degenerado) → n
- Resumiendo

MÉTODO	COMPLEJIDAD	
	CASO MEJOR	CASO PEOR
Insertar	O(1)	O(n)
Buscar	O(1)	O(n)
Borrar	O(1)	O(n)
Recorridos	O(n)	O(n)

Objetivo

Minimizar la altura del árbol, evitando árboles degenerados

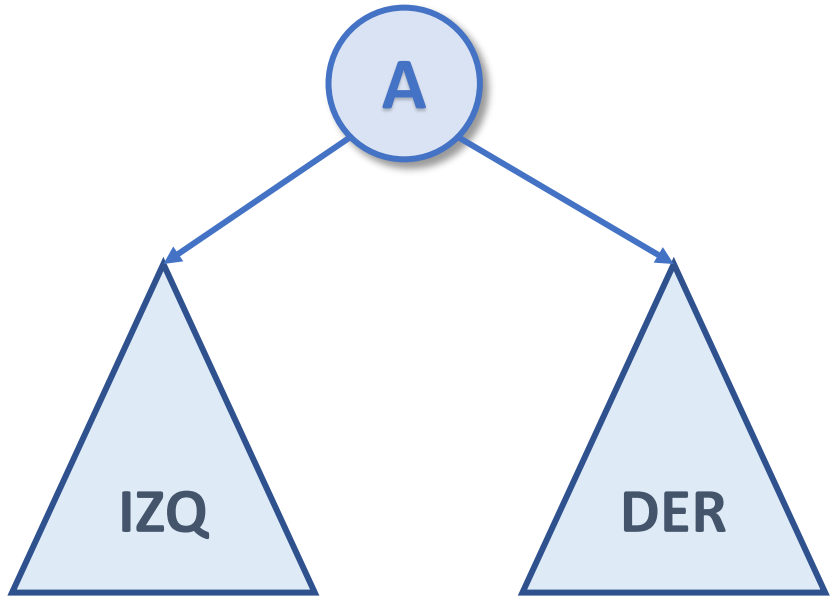
Otros tipos de árboles

- Árboles de altura mínima
- Árboles perfectamente equilibrados (APE)
- Árboles de Adelson-Velskii y Landis (AVL)
- Árboles de Fibonacci

Árboles de altura mínima

- Son aquellos que tiene la altura mínima para que quepan todos los nodos del árbol

- Recordamos que la altura de un árbol se calcula con la siguiente formula
  - $h = \log_2(nodos+1)$

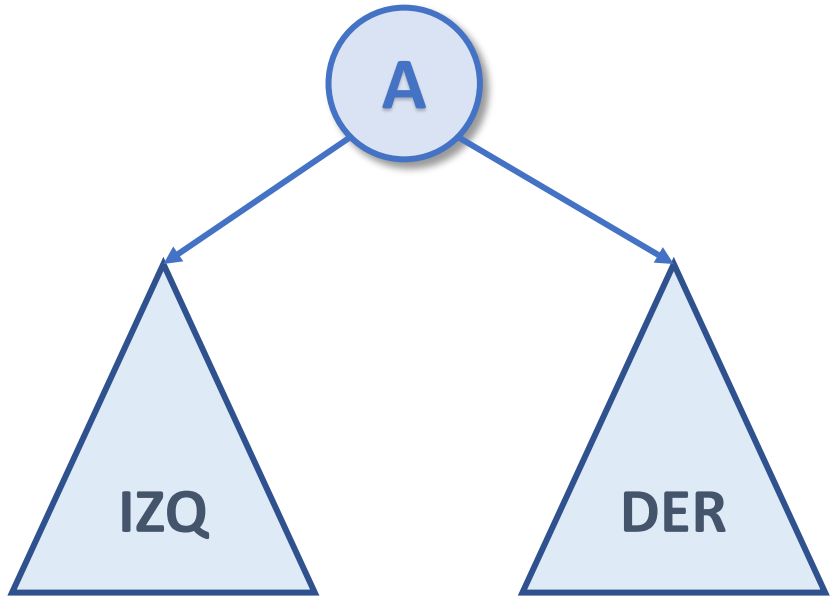


Árboles perfectamente equilibrados (APE)

- Para cualquier nodo del árbol se cumple que la diferencia de nodos entre el subárbol izquierdo y el subárbol derecho es menor o igual a uno en valor absoluto

- |#Izq - #der| <= 1
  - #izq → número de nodos del subárbol izquierdo
  - #der → número de nodos del subárbol derecho

- Garantiza la altura mínima de un árbol



Problema de los árboles APE

- Cada vez que se borra o insertar un elemento puede ser necesario reconstruir el árbol por lo que la complejidad en el peor y mejor de los casos es O(n)
- Tan sólo en las búsquedas conseguimos una complejidad logarítmica debido a la ordenación de los nodos
- Eficiente si hay que hacer muchas búsquedas
- Solución
  - Diseñar un árbol capaz de tener complejidad temporal $\log_2(n)$ en el peor de los casos para las tres operaciones básicas