El conteo de votos

- Se quiere hacer una votación para supremo líder del planeta
- Todos los habitantes del planeta son candidatos
- Cada persona tiene asignado un número, llamado ID
- El voto consiste en anotar el ID de tu candidato en la papeleta
- La persona cuyo ID aparezca en más papeletas gana

¿Cuál es el costo de contar los votos?

Los desafíos del conteo de votos

No sabemos cuántos candidatos aparecerán en los votos

¿Cómo podemos llevar la cuenta solo de los ID que han recibido votos?

Hay que considerar la eficiencia de contar un nuevo voto

Nuestra primera estructura de datos: El diccionario

Es una estructura de datos que permite las siguientes operaciones:

- Asociar un **valor** a una **clave**, o actualizarlo
- Obtener el **valor** asociado a una **clave**

Y para ciertos casos de uso:

- Eliminar de la estructura una **clave** y su **valor** asociado

El diccionario

Queremos un diccionario para contar los votos

… que use solo lo necesario de memoria

¿Cómo podemos implementarlo?

El árbol binario de búsqueda (ABB)

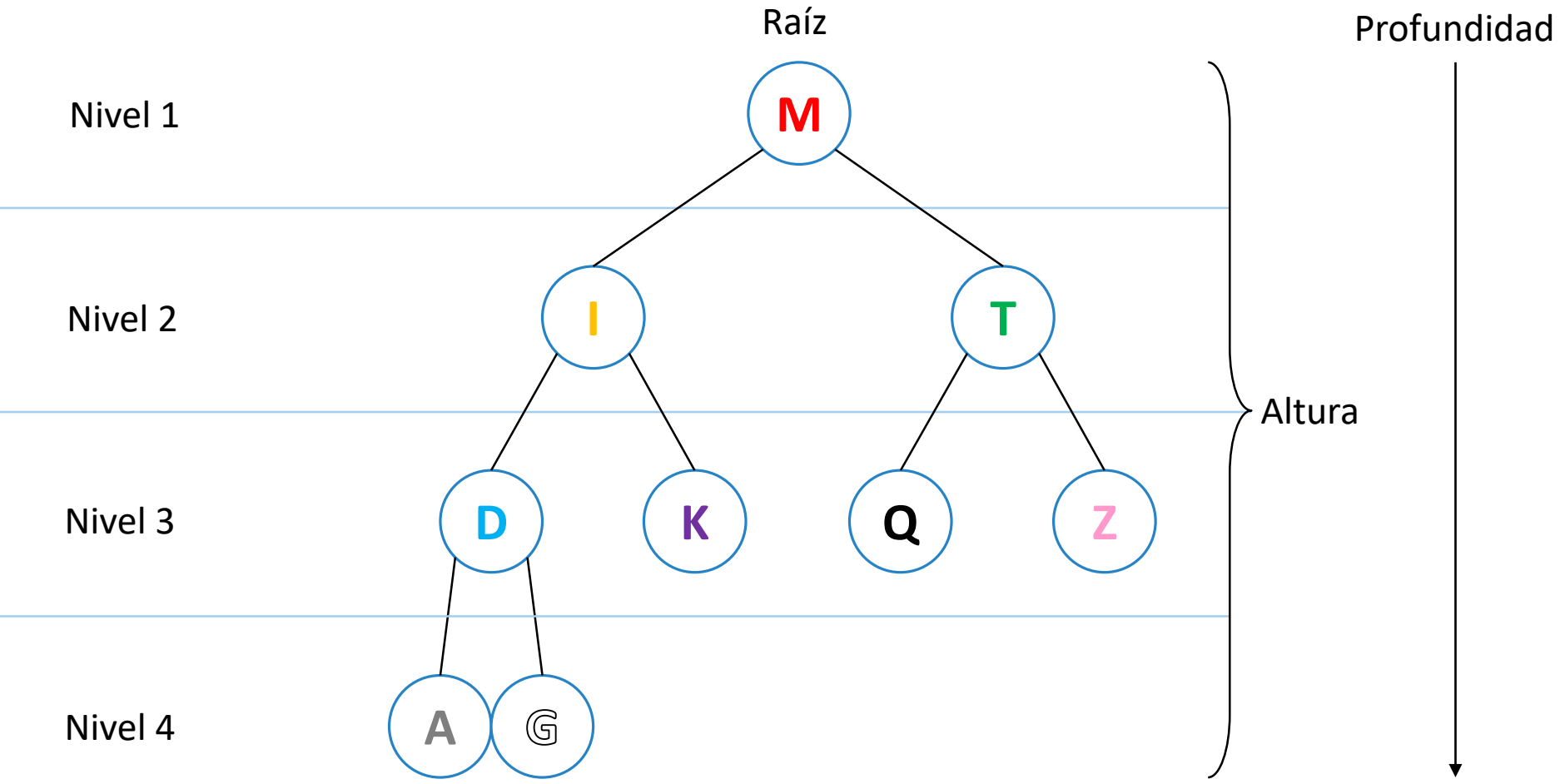
Es un **árbol binario**, con uno de sus elementos como raíz

Los demás datos están divididos en los (con claves) mayores y los (con claves) menores que la raíz

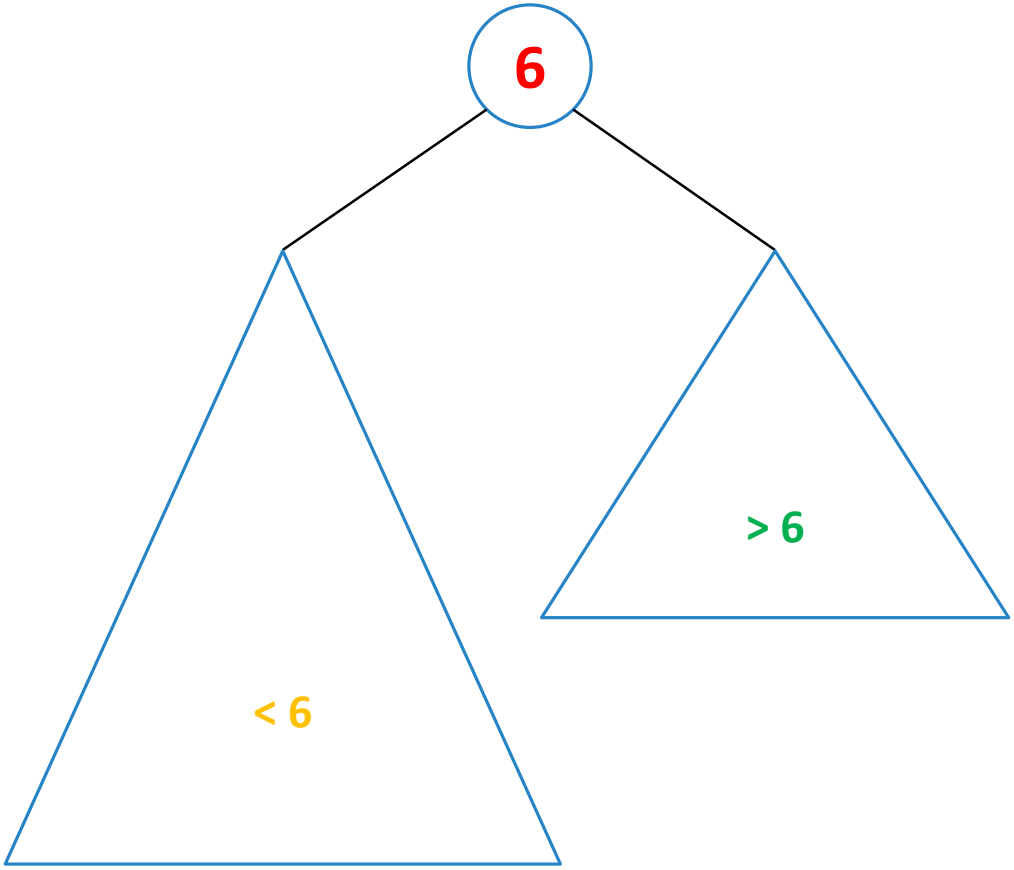
Cada uno de estos grupos está a su vez organizado como **ABB**

Los menores cuelgan del hijo izquierdo de la raíz, y los mayores, del hijo derecho

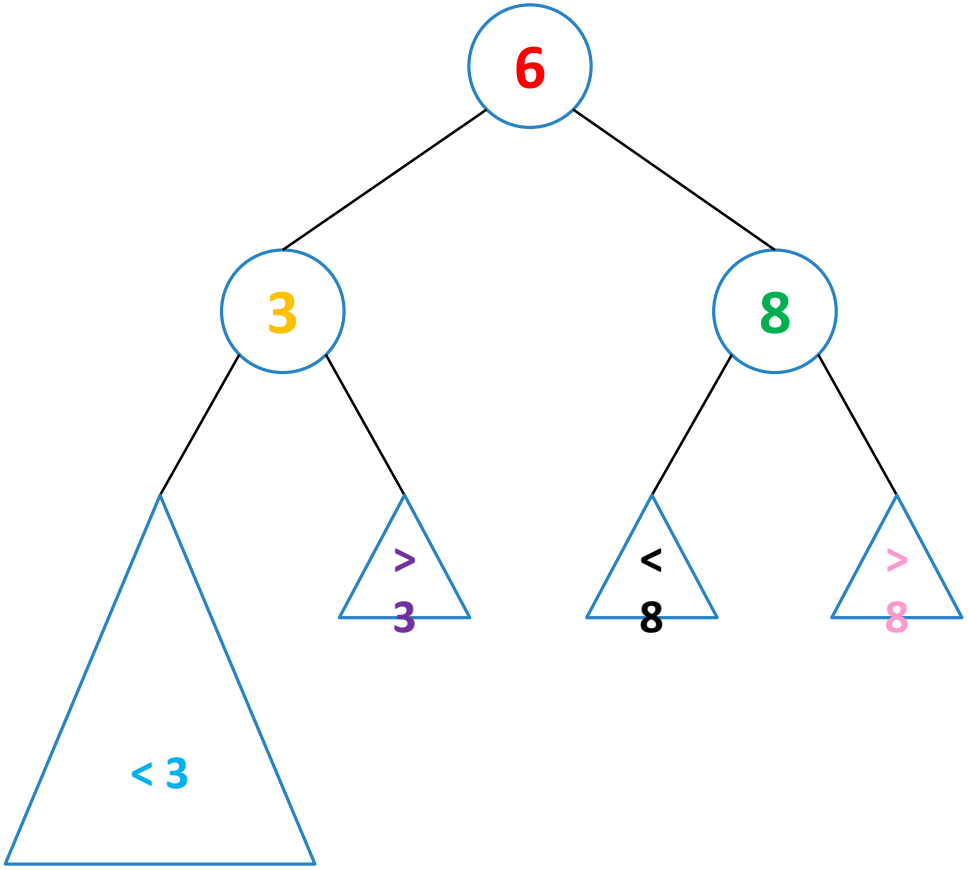
Anatomía de un árbol binario



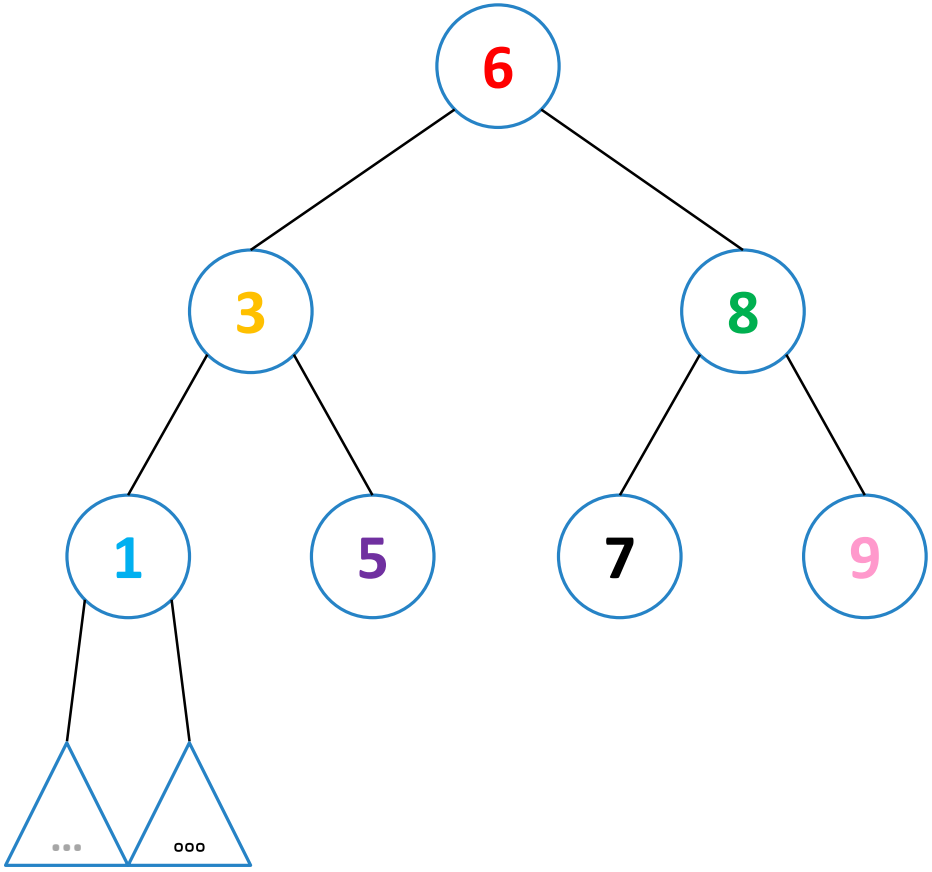
El árbol binario de búsqueda …



… está compuesto por árboles binarios de búsqueda



… y así hasta las hojas



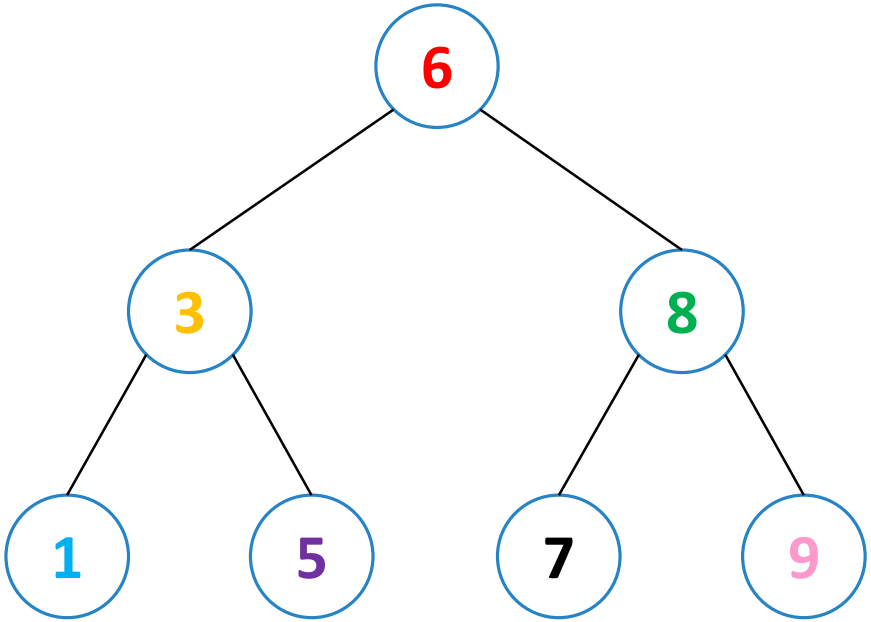
Operaciones del ABB

¿Cómo se busca un elemento en el árbol?

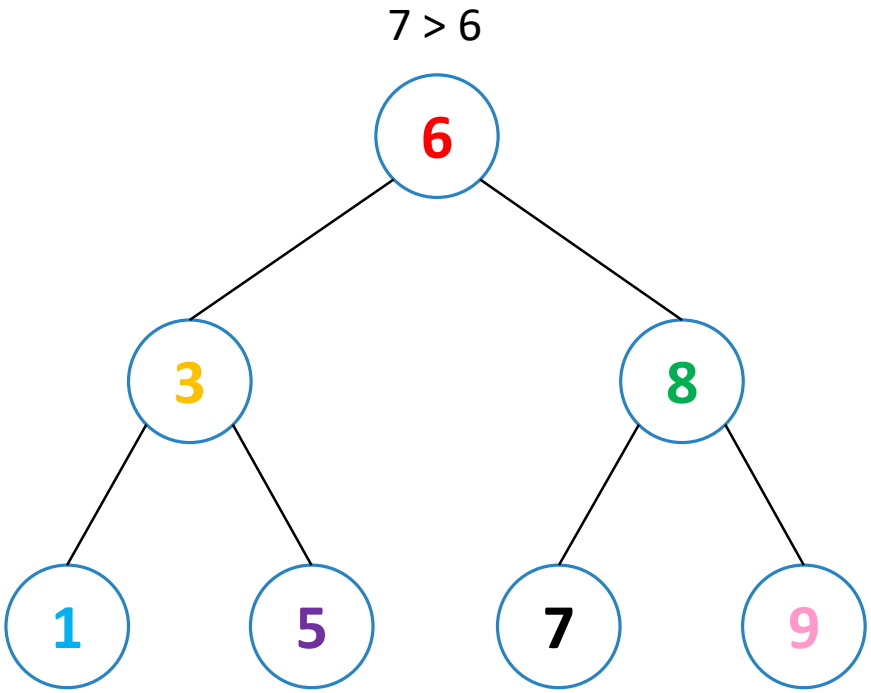
¿Qué pasa si queremos insertar un nuevo dato?

Tratemos de aprovechar que la estructura es recursiva

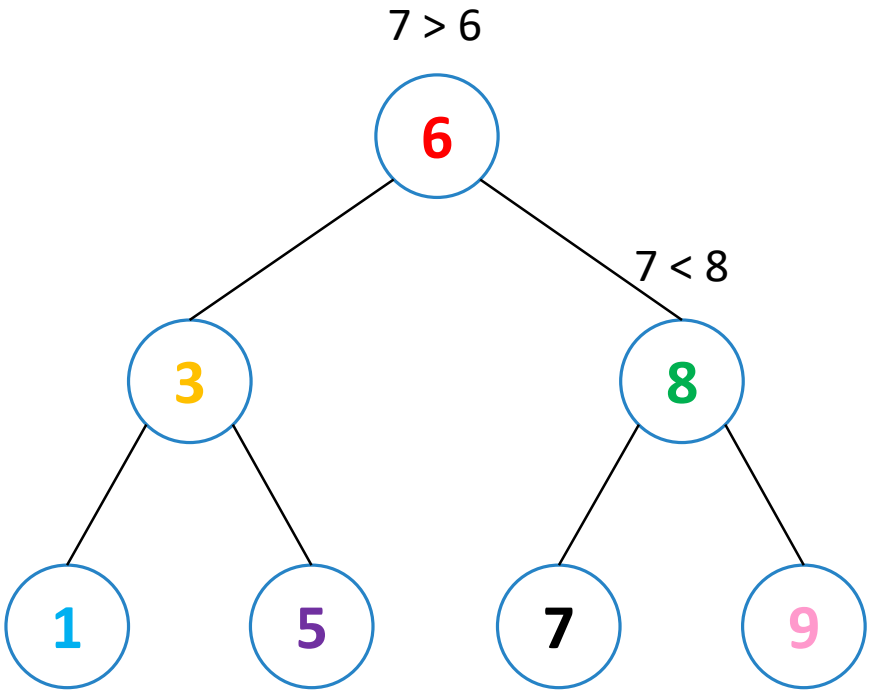
Busquemos el 7



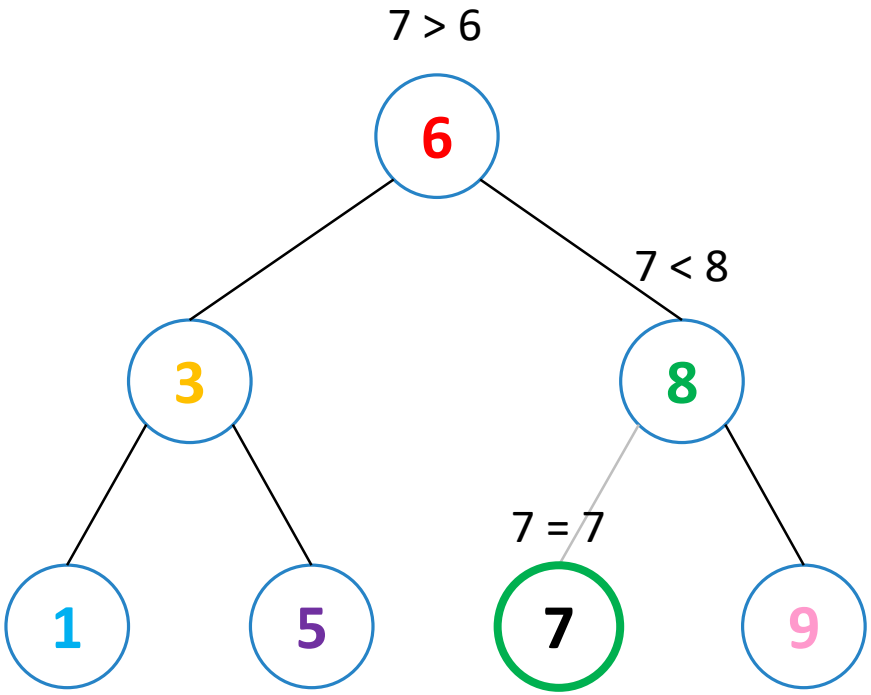
Busquemos el 7



Busquemos el 7

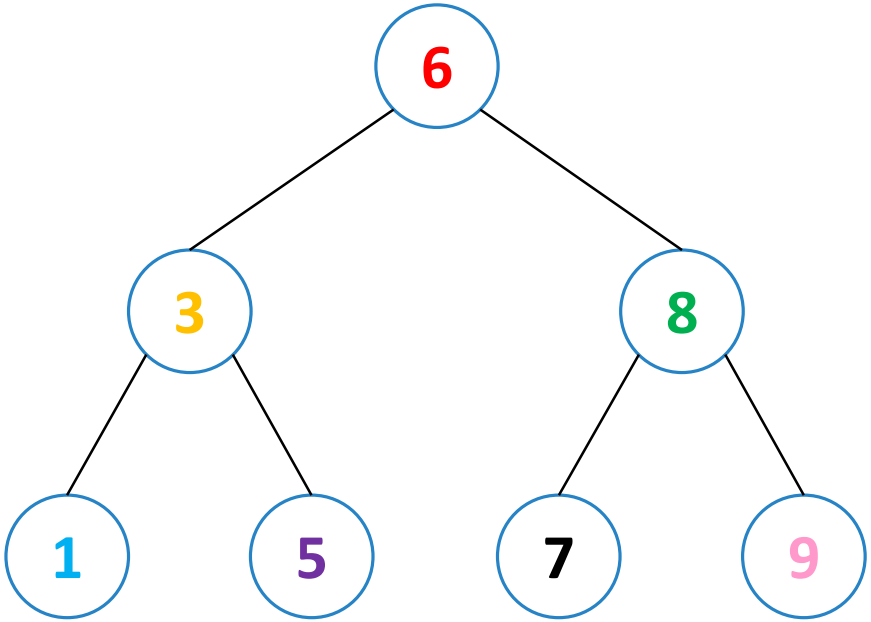


Busquemos el 7

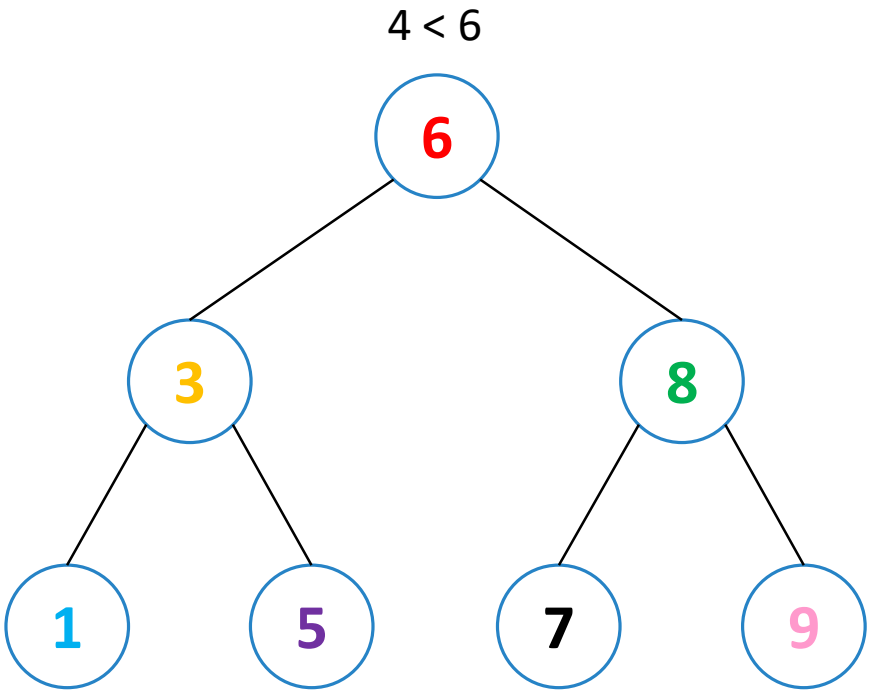


```
search(A, k):
    if A = ∅  o  A.key = k:
        return A
    else if k < A.key:
        search(A.left , k)
    else:
        search(A.right, k)
```

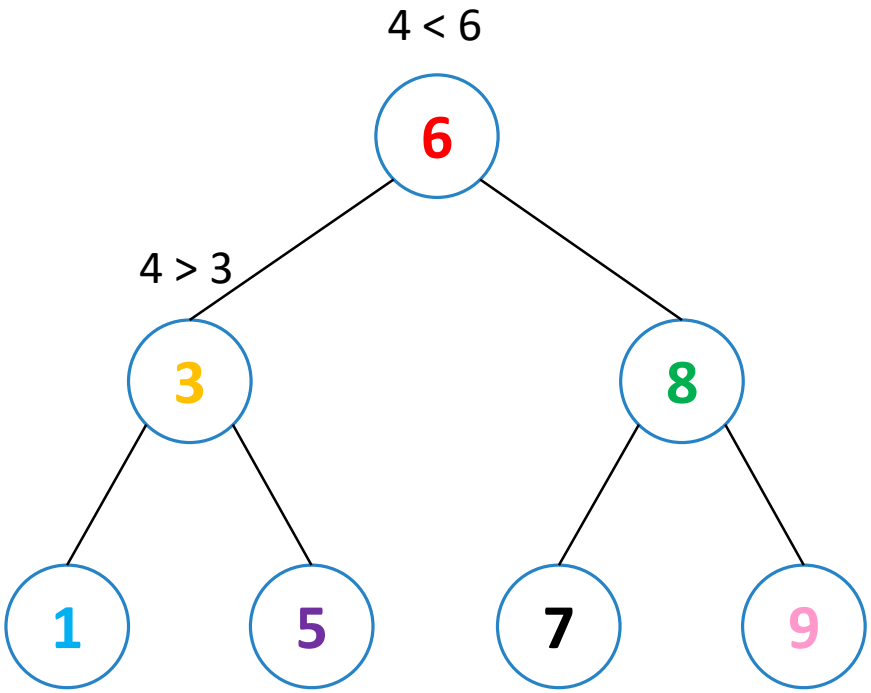
Insertemos el 4



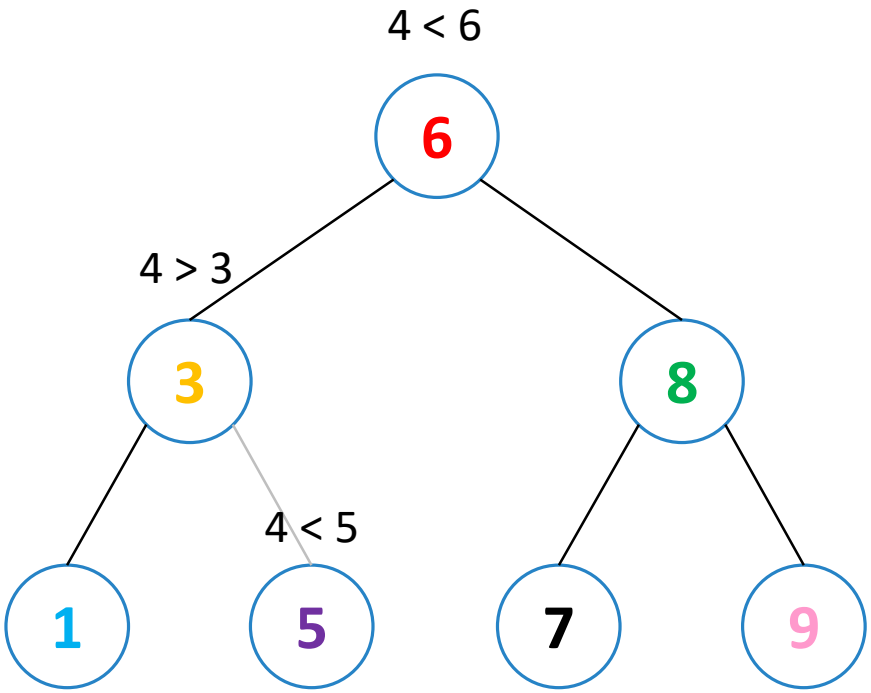
Insertemos el 4



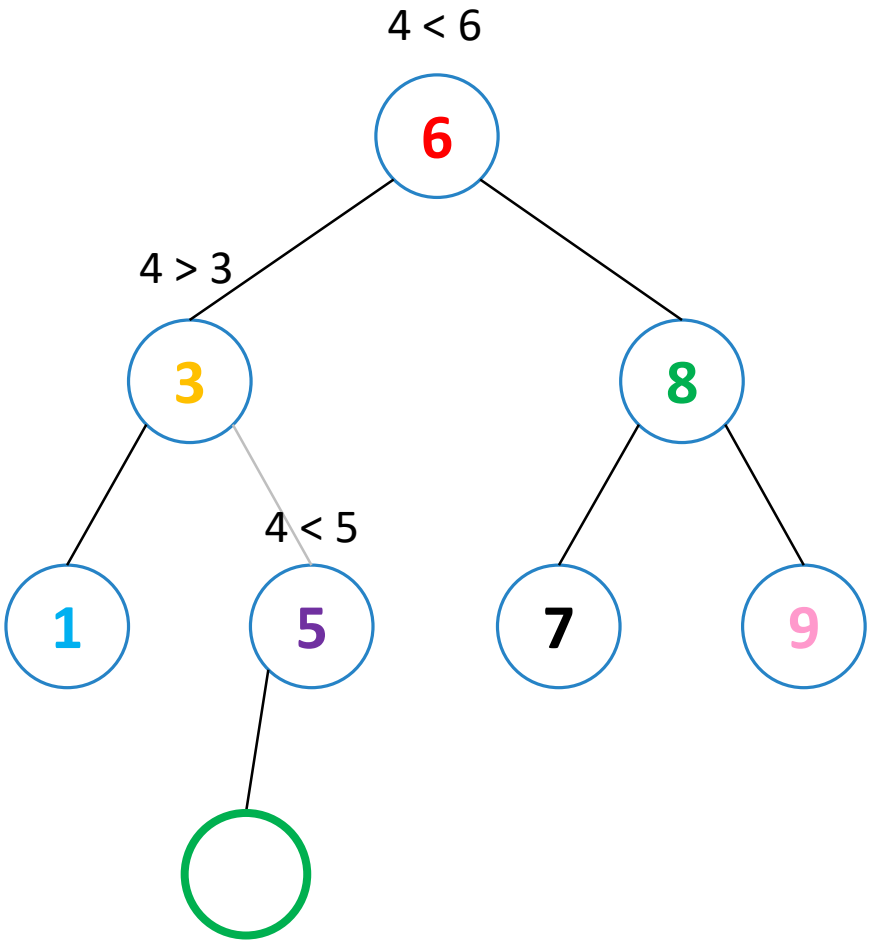
Insertemos el 4



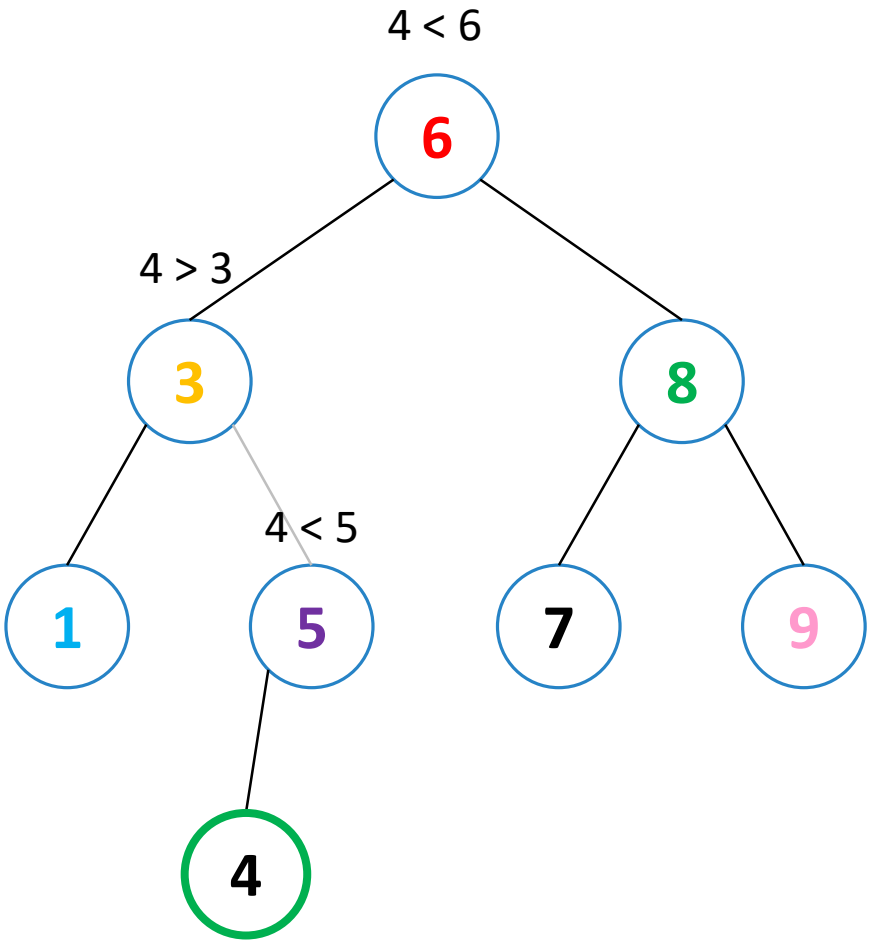
Insertemos el 4



Insertemos el 4



Insertemos el 4



$$\textit{insert}(A, k):$$

$$B \leftarrow \textit{search}(A, k)$$

$$B.key \leftarrow k$$

Eliminación

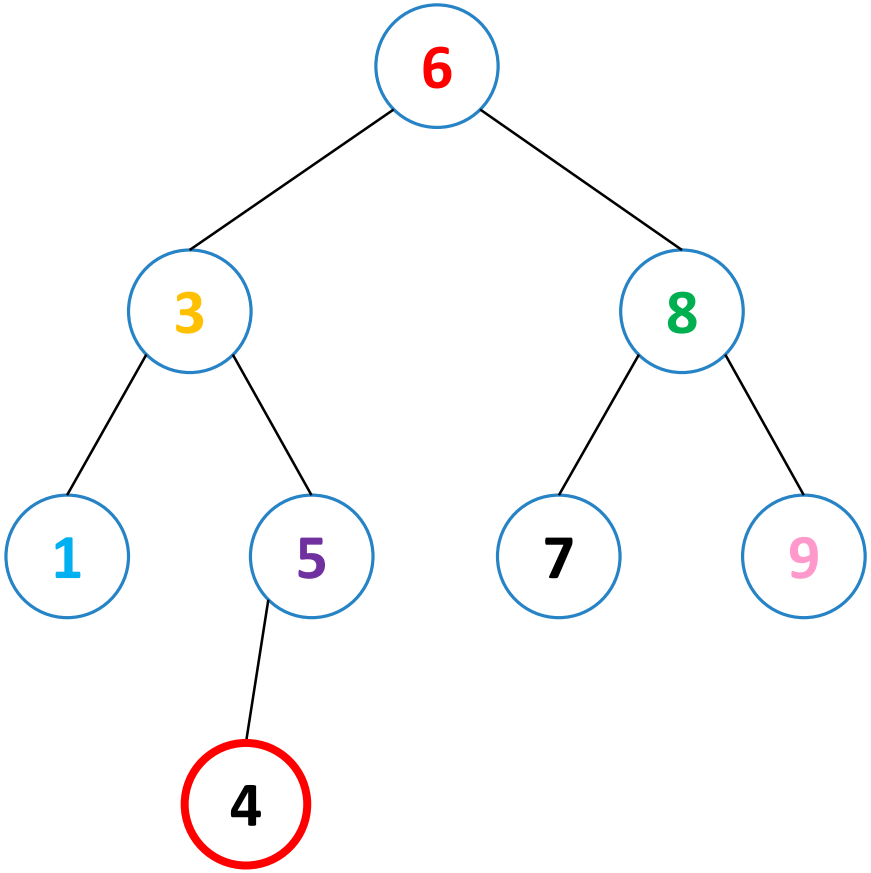
Se ha ingresado un dato erróneo al árbol

Si el dato quedó en una hoja, o tiene un solo hijo, eliminarlo es trivial

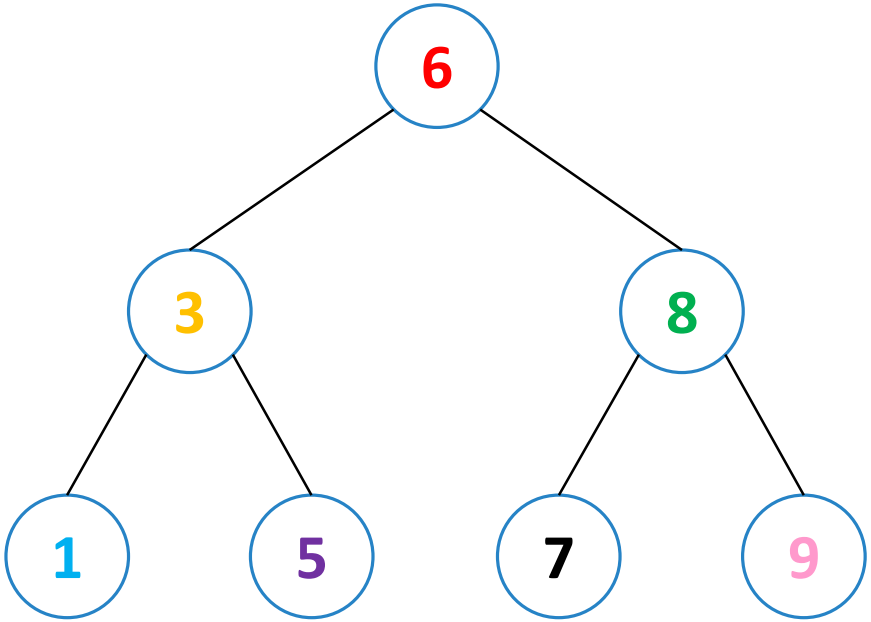
Si no, ¿cómo podemos eliminarlo sin romper la estructura?

¿Podremos reemplazarlo por otro nodo del árbol? ¿Cuál?

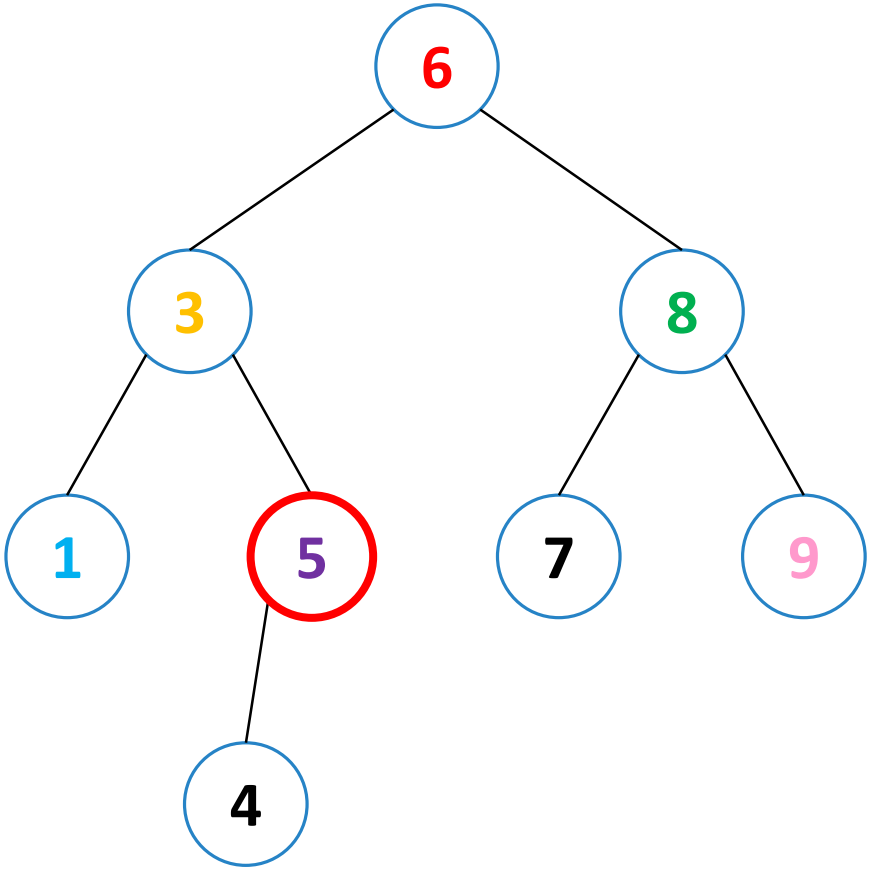
Eliminemos el 4



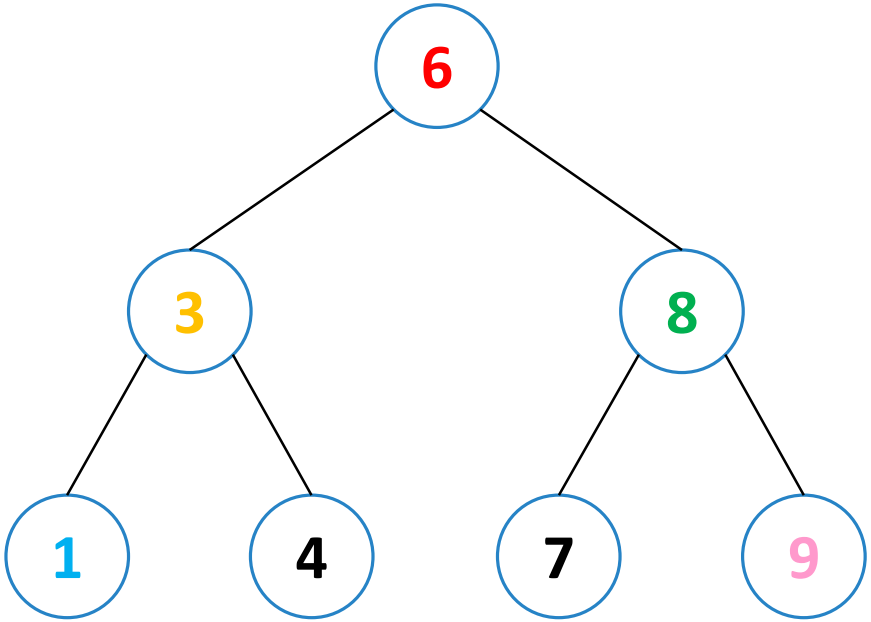
Eliminemos el 4



Eliminemos el 5



Eliminemos el 5



Antecesor y Sucesor

Si los nodos estuvieran ordenados en una lista según su key:

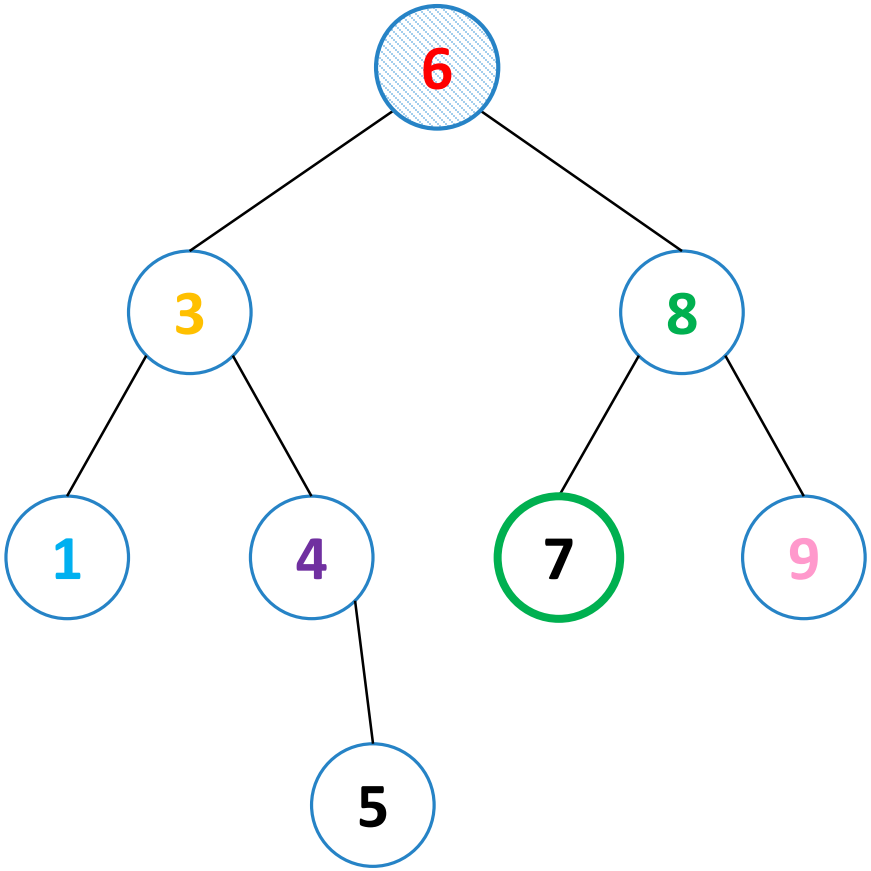
- El **sucesor** de un nodo es el siguiente de la lista
- El **antecesor** de un nodo es el anterior en la lista

¿Cómo podemos encontrar estos elementos dentro del árbol?

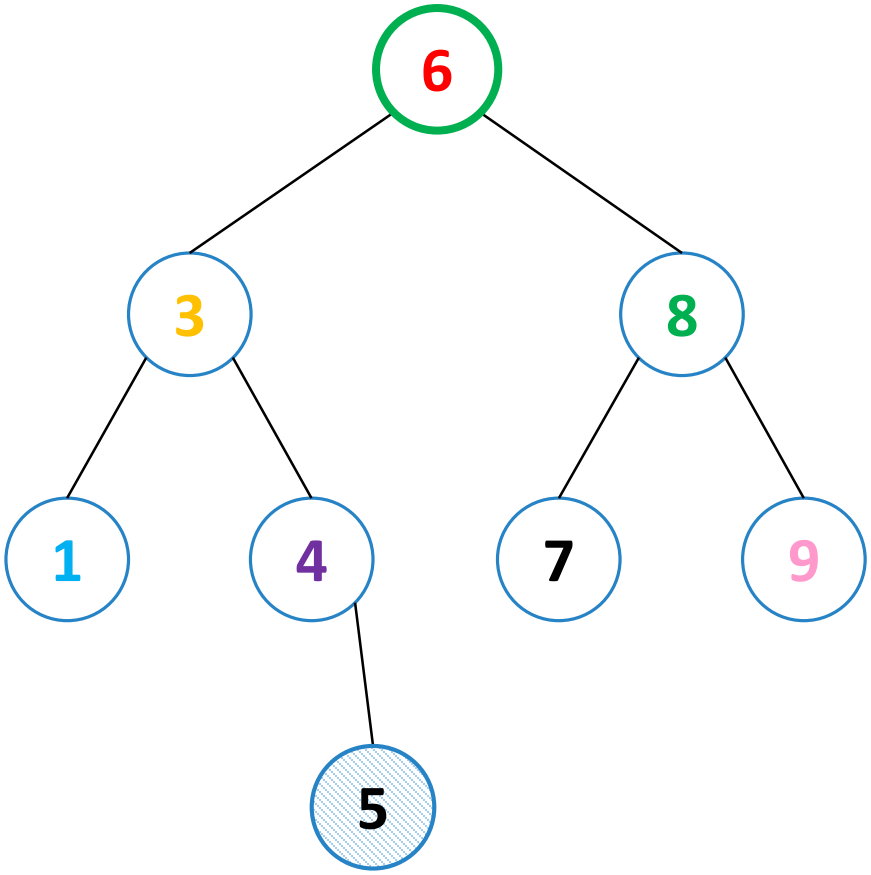
```
min(A):
    if A.left = ∅:
        return A
    else:
        return min(A.left)
```

```
succesor(A):
    if A.right ≠ ∅:
        return min(A.right)
    B ← A.parent
    while A = B.right:
        A ← B
        B ← A.parent
    return B
```

Busquemos el sucesor del 6



Busquemos el sucesor del 5



$$delete(A, k):$$

$$D \leftarrow search(A, k)$$

$$if\ D\ es\ hoja, \quad D \leftarrow \emptyset$$

$$else\ if\ D\ tiene\ un\ solo\ hijo\ H, \quad D \leftarrow H$$

$$else:$$

$$R \leftarrow succesor(D)$$

$$D \leftarrow R$$

$$R \leftarrow \emptyset$$

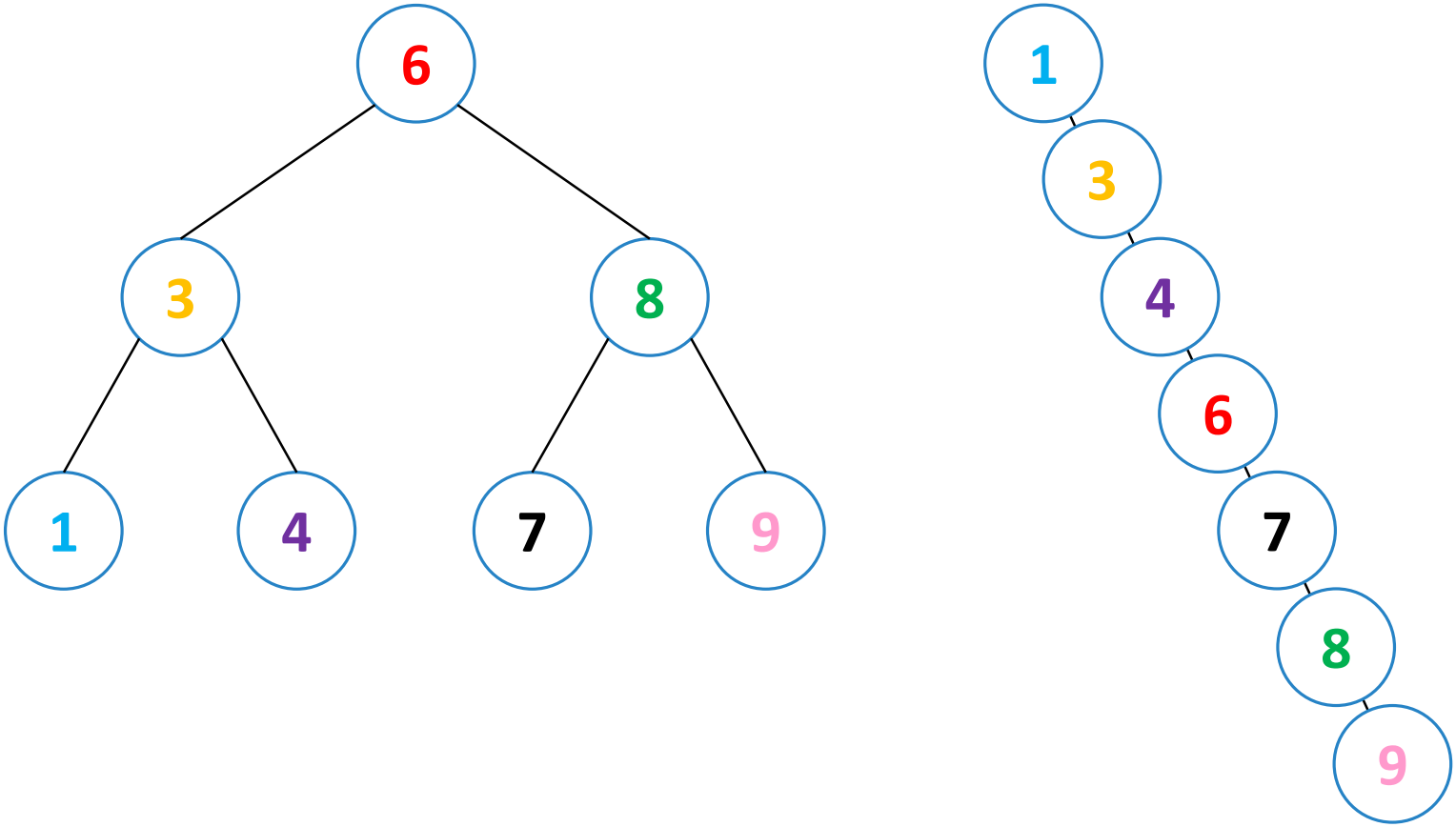
Raíces y raíces

¿Hay algunas raíces más convenientes que otras?

¿Qué pasa con el árbol si no queda un dato conveniente como raíz?

¿Cómo varía la complejidad de las operaciones?

Mismos datos, distinto árbol



¡Todo depende del orden de inserción (y posiblemente eliminación)!