Primera alternativa: iteradores

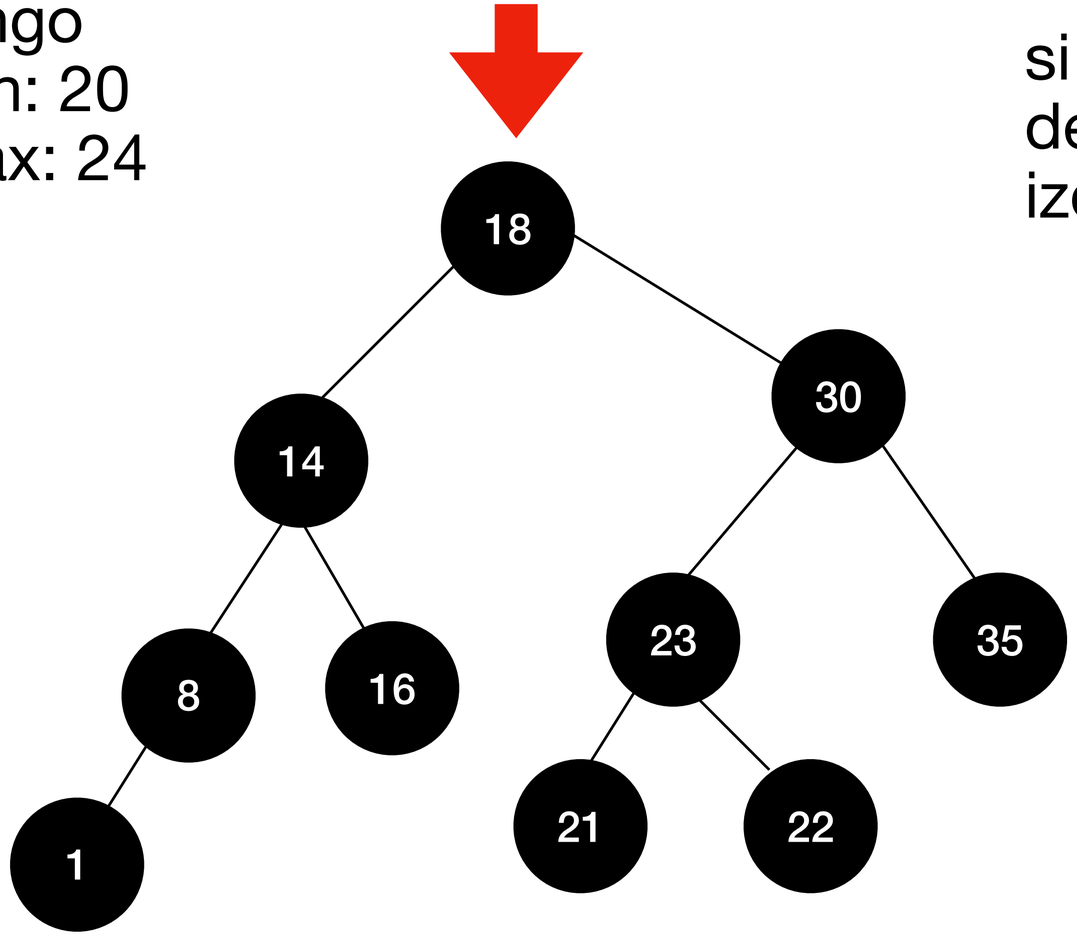
- Los iteradores permiten recorrer los elementos de cualquier estructura utilizando la misma interfaz
- El iterador de un abb/avl devuelve los elementos **en orden**.
- El costo total de recorrer todos los elementos es O(n).

Primera alternativa: iteradores

- ESCRIBIR EL ALGORITMO
-

Segunda alternativa: algoritmo recursivo

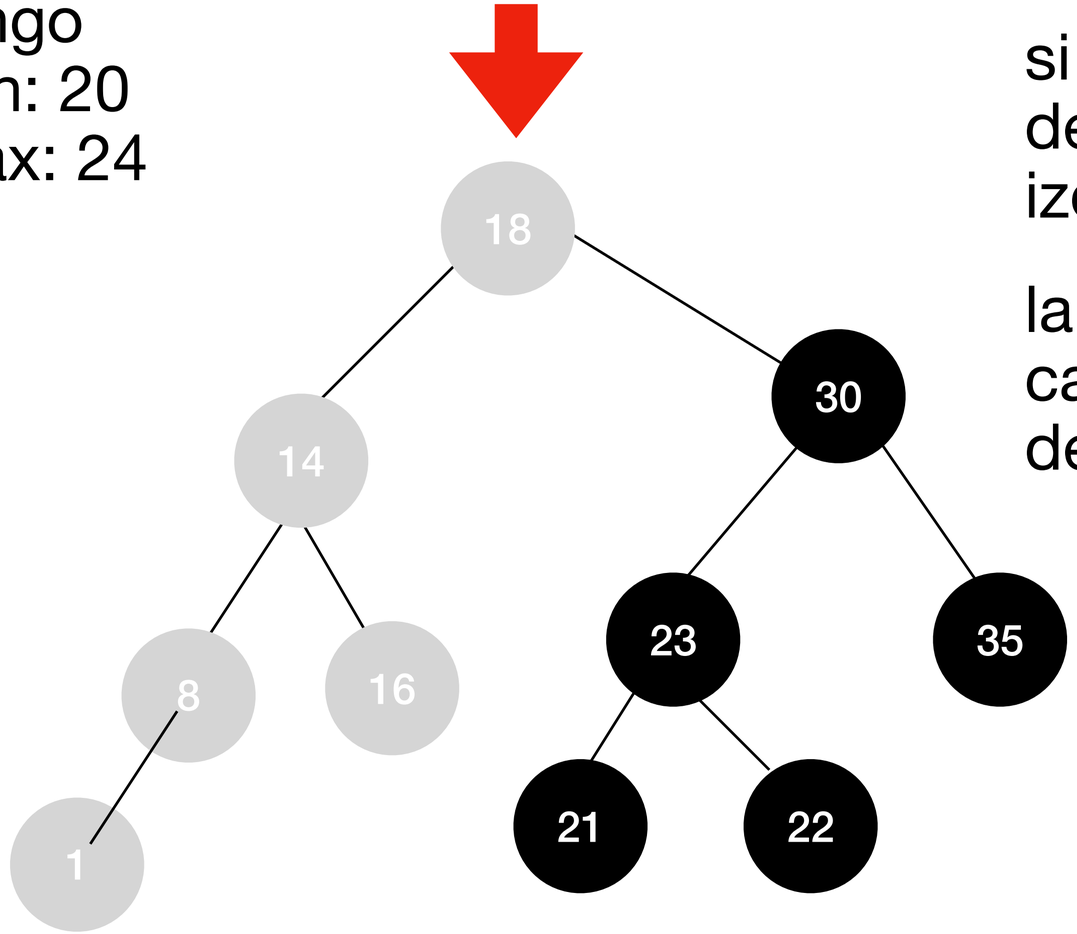
rango
min: 20
max: 24



si es menor al mínimo, puedo descartar todo el subarbol izquierdo

Segunda alternativa: algoritmo recursivo

rango
min: 20
max: 24



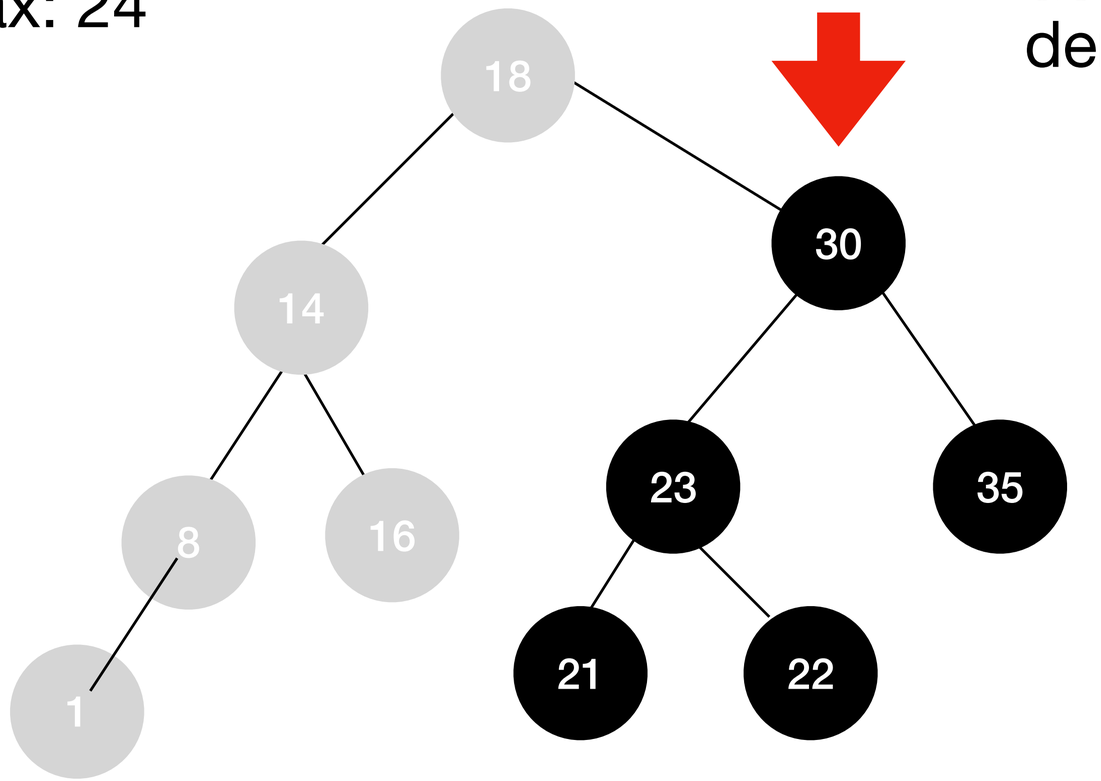
si es menor al mínimo, puedo descartar todo el subarbol izquierdo

la cantidad de nodos en rango será la cantidad de nodos en rango de la derecha

```
if nodo.dato > max
  return cantNodosEnRango(nodo.der,
  min, max)
```

Segunda alternativa: algoritmo recursivo

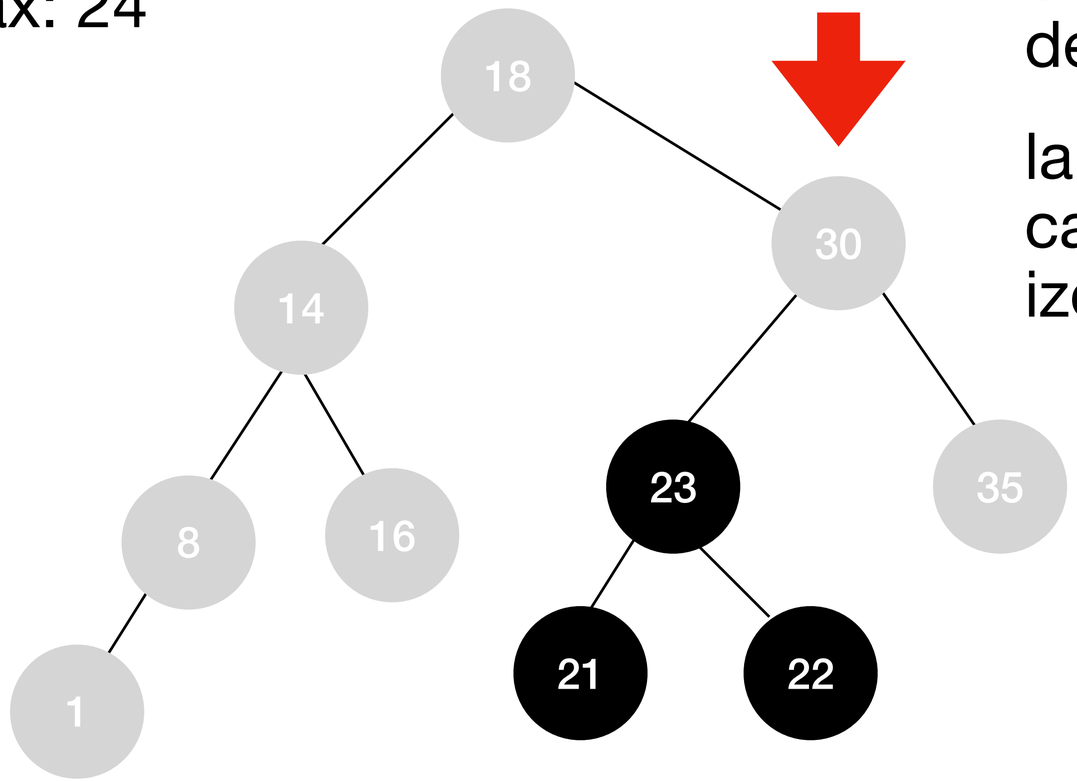
rango
min: 20
max: 24



si es mayor al máximo, puedo descartar todo el subarbol derecho

Segunda alternativa: algoritmo recursivo

rango
min: 20
max: 24



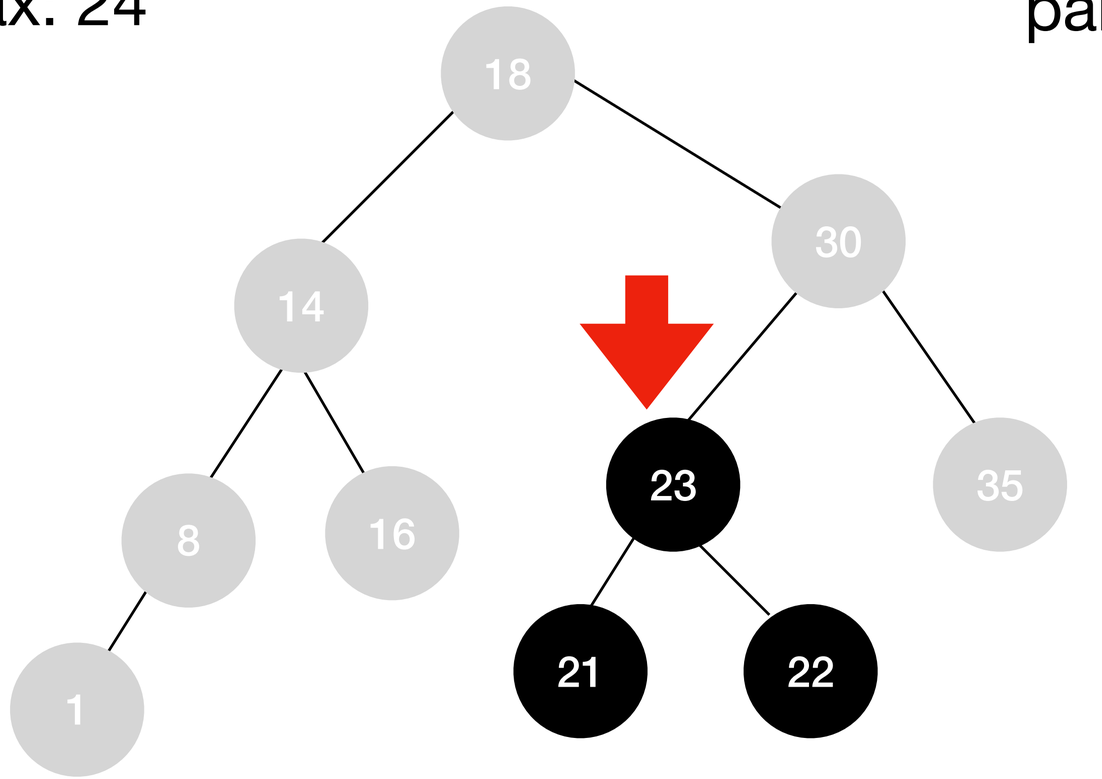
si es mayor al máximo, puedo descartar todo el subarbol derecho

la cantidad de nodos en rango será la cantidad de nodos en rango de la izquierda

```
if nodo.dato < min
  return cantNodosEnRango(nodo.izq,
  min, max)
```

Segunda alternativa: algoritmo recursivo

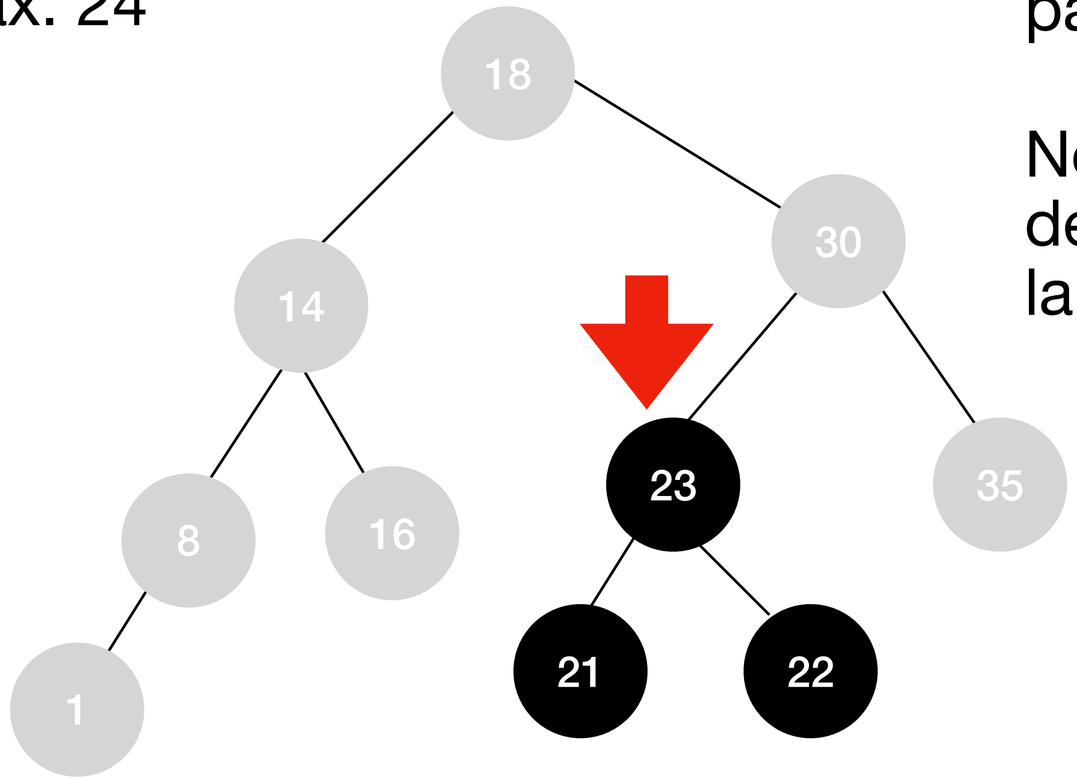
rango
min: 20
max: 24



si está en rango, tengo que revisar para los dos lados

Segunda alternativa: algoritmo recursivo

rango
min: 20
max: 24



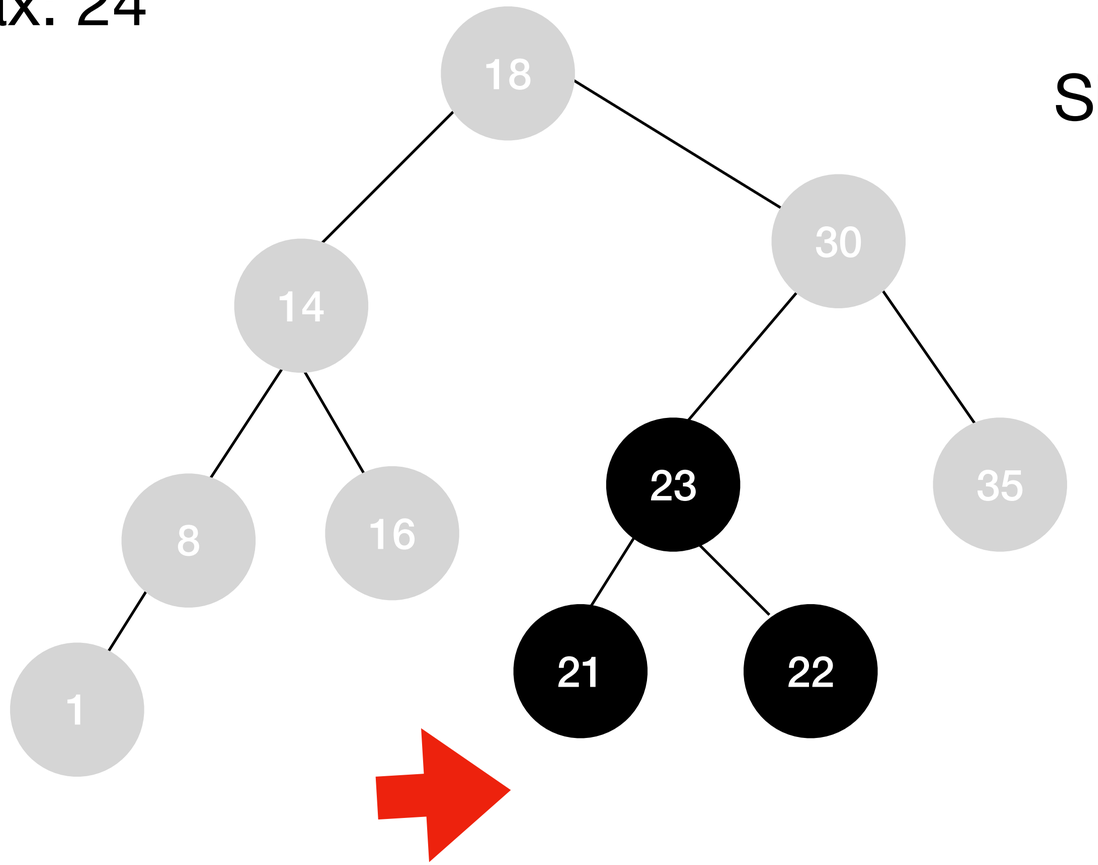
si está en rango, tengo que revisar para los dos lados

NodosEnRango será NodosEnRango de la izquierda más NodosEnRango de la derecha más uno

```
if min <= nodo.dato < max
  return cantNodosEnRango(nodo.izq,
  min, max) +
  cantNodosEnRango(nodo.der, min,
  max) + 1
```

Segunda alternativa: algoritmo recursivo

rango
min: 20
max: 24



falta algo?

Si el nodo es nil, devuelvo 0

```
if nodo == nil
  return 0
```

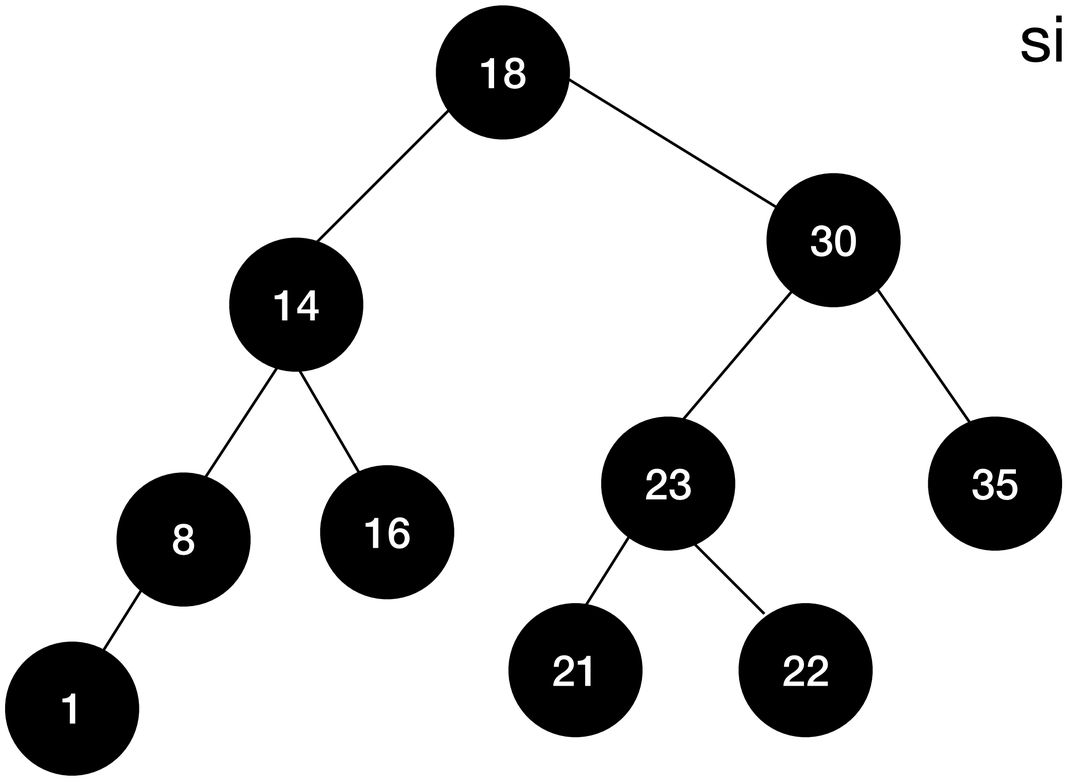
Segunda alternativa: algoritmo recursivo

```
proc cantNodosEnRango(in n: Nodo, in min: int, in max: int): int
        if nodo == nil
          return 0

        if nodo.dato >= max
          return cantNodosEnRango(nodo.der,
          min, max)

        if nodo.dato < min
          return cantNodosEnRango(nodo.izq,
          min, max)

        if min <= nodo.dato < max
          return cantNodosEnRango(nodo.izq,
          min, max) +
          cantNodosEnRango(nodo.der, min,
          max) + 1
```

Segunda alternativa: algoritmo recursivo

- Cuál es la complejidad?
- Para eso veamos cuál es el peor caso…
    - Tener que recorrer todo el árbol! Eso pasa si el rango abarca todos los nodos
    - Como no hay ciclos y siempre nos movemos hacia abajo, cada nodo lo recorreremos a lo sumo una sóla vez
- Y cuántos nodos tiene el árbol? n
- Luego, la complejidad de peor caso es O(n)

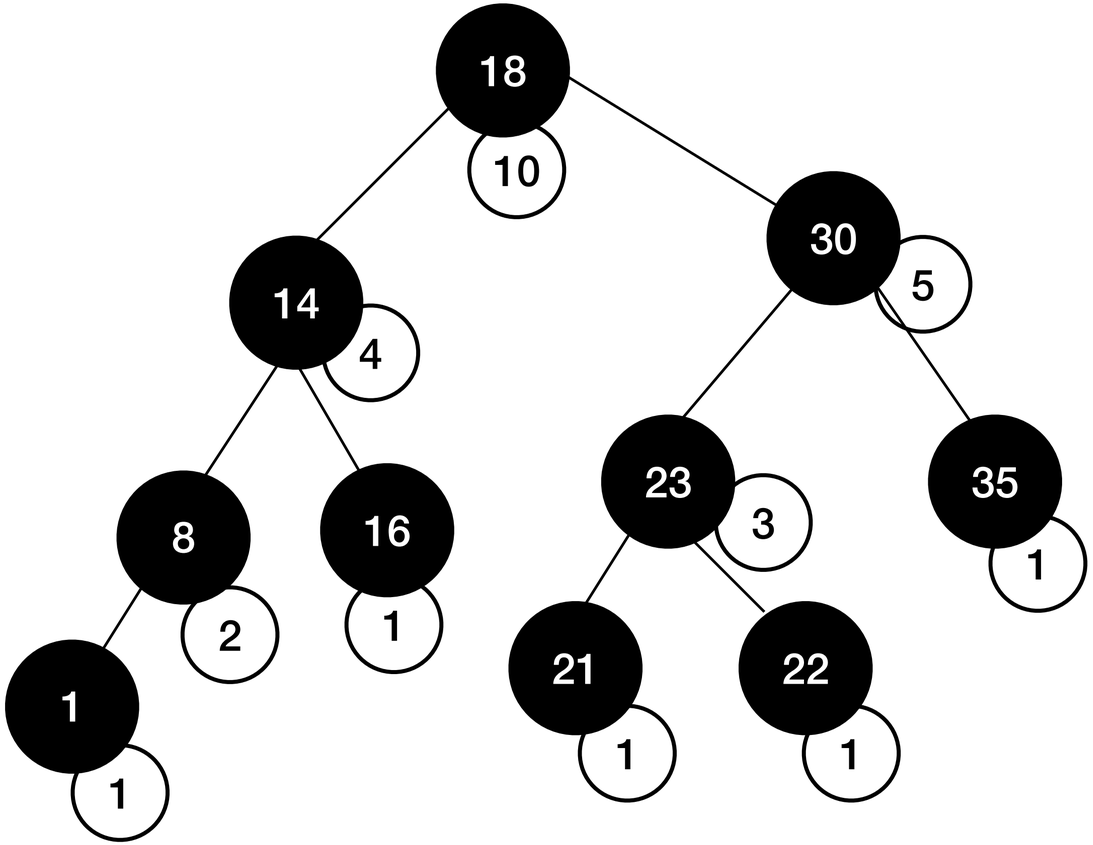
Tercera alternativa: guardar cosas en los nodos



si pudiera saber cuántos nodos hay menores que un valor, puedo hacer lo siguiente:

- busco min (o el primer nodo mayor) y me fijo cuántos nodos más chicos hay (cmin)
- busco max (o el primer nodo mayor) y me fijo cuántos nodos más chicos hay (cmax)
- la cant. de nodos en rango es cmax-cmin

Tercera alternativa: guardar cosas en los nodos



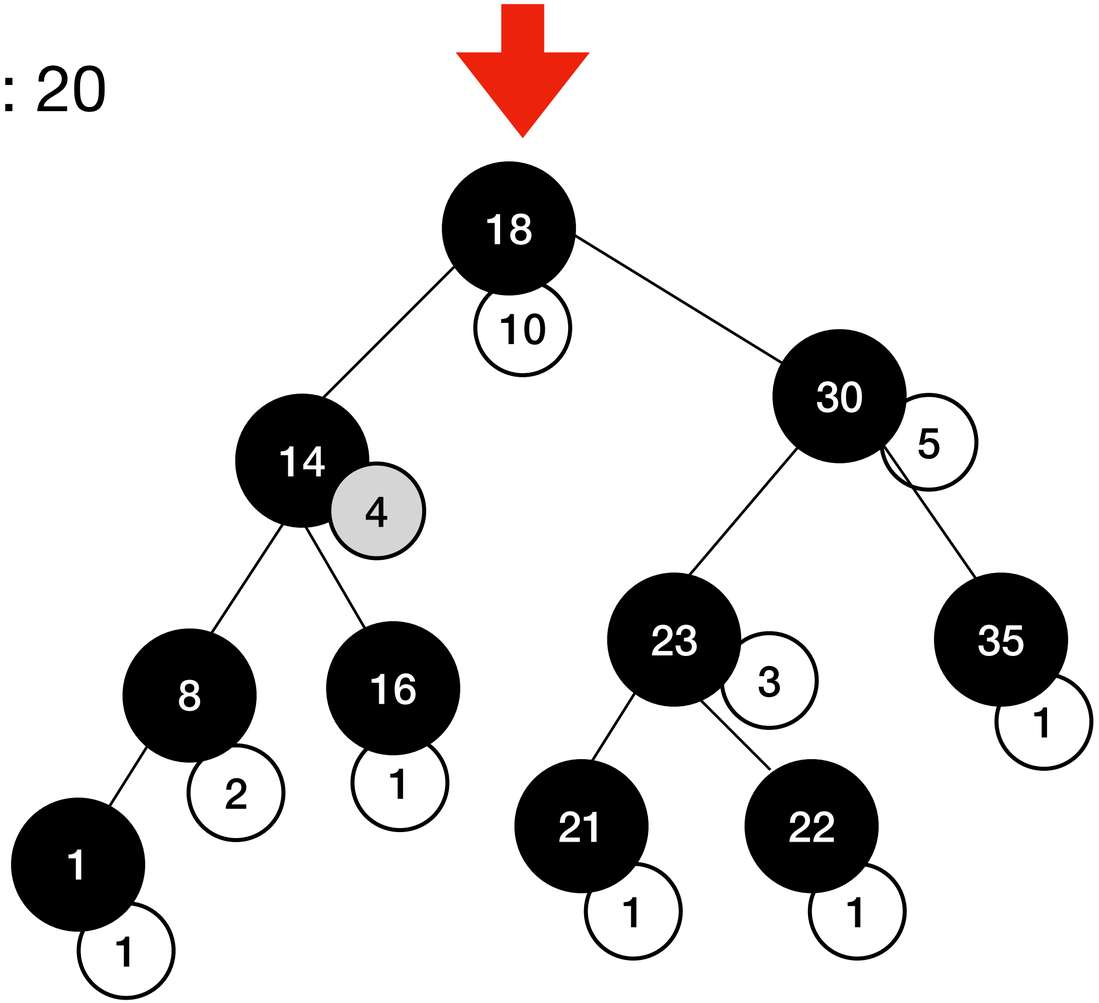
cómo sé cuántos nodos menores a *val* hay?

guardo en cada nodo el tamaño del subarbol (tam)

ahora busco val

Tercera alternativa: guardar cosas en los nodos

val: 20



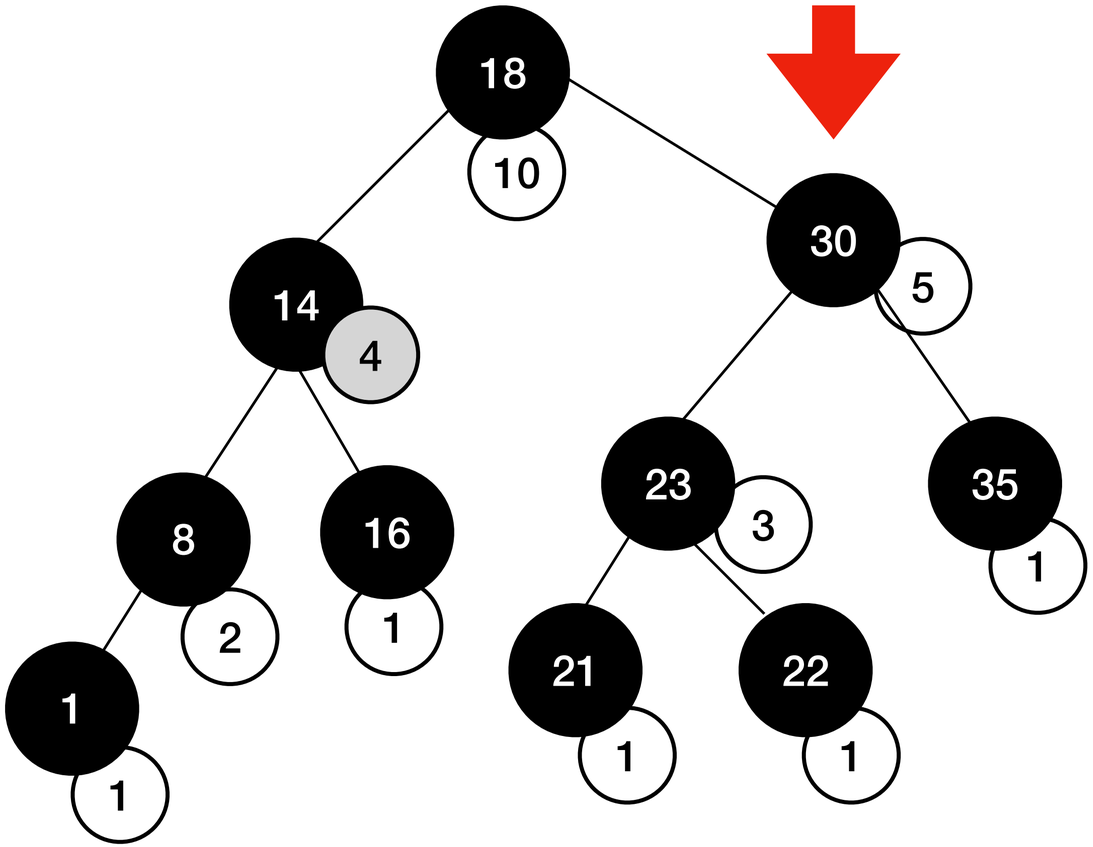
cómo sé cuántos nodos menores a *val* hay?

si el nodo actual es menor o igual, todo el subarbol izquierdo es menor

```
if nodo.dato <= val
  if nodo.izq == nil
    return 1 + cantNodosMenores(nodo.der)
  else
    return 1 + nodo.izq.tam +
  cantNodosMenores(nodo.der, val)
```

Tercera alternativa: guardar cosas en los nodos

val: 20



cómo sé cuántos nodos menores a *val* hay?

si el nodo actual es mayor, todo el subarbol izquierdo es mayor (pero no me importa)

```
if nodo.dato > val
  return cantNodosMenores(nodo.der, val)
```

Tercera alternativa: guardar cosas en los nodos

```
proc cantNodosMenores(in n: Nodo, in val: int): int

   if nodo == nil
     return 0

   if nodo.dato <= val
     if nodo.izq == nil
       return 1 + cantNodosMenores(nodo.der)
     else
       return 1 + nodo.izq.tam + cantNodosMenores(nodo.der, val)

   if nodo.dato > val
     return cantNodosMenores(nodo.der, val)



proc cantNodosEnRango(in n: Nodo, in min: int, in max: int): int

   return cantNodosMenores(n, max) - cantNodosMenores(n, min)
```

Tercera alternativa: guardar cosas en los nodos

- Cuál es la complejidad?
- La misma de buscar un nodo! O(log n)

- Y cuanto me cuesta actualizar tam?
- O(1)!
  - Al insertar un nodo en el árbol, sumo uno a todo el camino hasta el nodo
  - Al borrar un nodo, resto uno en todo el camino hasta el nodo