Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

Moodle de la asignatura

Guía docente

Objetivos

- Descripción de las estructuras de datos mapa y diccionario ordenado
- Aproximaciones basadas en árboles binarios de búsqueda (ABB)
- Diseño e implementación de los TAD ABB y diccionario.

Índice

- Árboles binarios de búsqueda
- Mapas y Diccionarios ordenados
- Equilibrado de árboles
  - Árboles AVL
  - Árboles Rojo-Negro

Índice

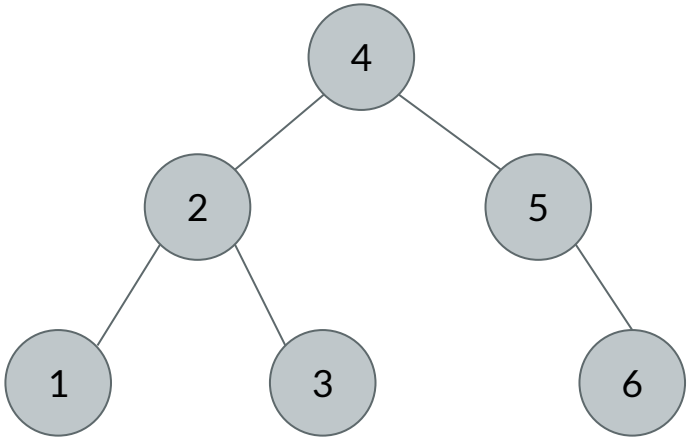
- **Árboles binarios de búsqueda**
- Mapas y Diccionarios ordenados
- Equilibrado de árboles
  - Árboles AVL
  - Árboles Rojo-Negro

Árbol binario de búsqueda

- Un ABB es un árbol binario en el que todos sus nodos satisfacen:
  - Sean u, v y w tres nodos tales que u está en el sub-árbol izquierdo de v y w en el sub-árbol derecho. Entonces:

    value(u) < value(v) < value(w)

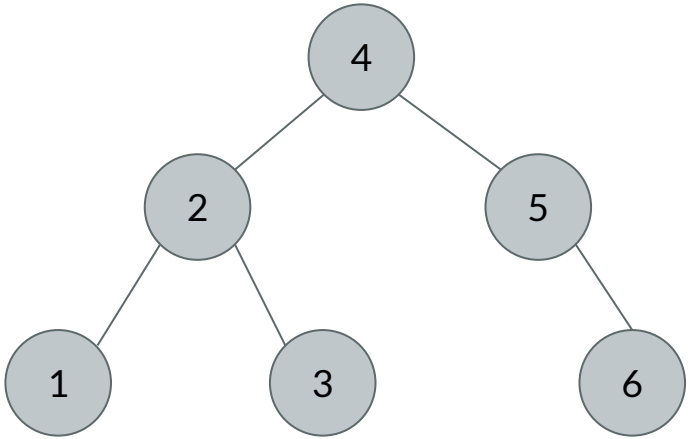
- Por lo tanto, un recorrido en inorden visita los elementos en orden ascendente.



Árbol binario de búsqueda

- La búsqueda de un elemento v desde el nodo n de un ABB se puede expresar de manera recursiva.
  - La complejidad del algoritmo es O(h) donde h es la altura.

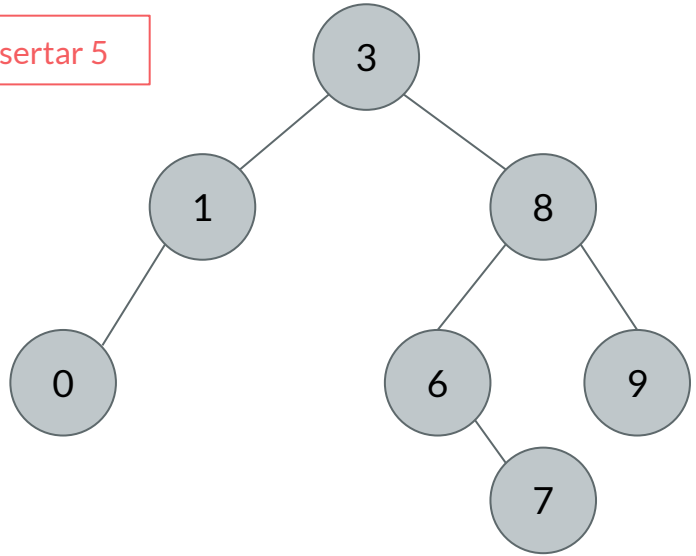
```
TreeSearch(value, node)
  if isLeaf(node)
    return value == node.value
  if value < node.value
    return TreeSearch(value, node.left)
  else if value > node.value
    return TreeSearch(value, node.right)
  return true;
```



Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

Insertar 5



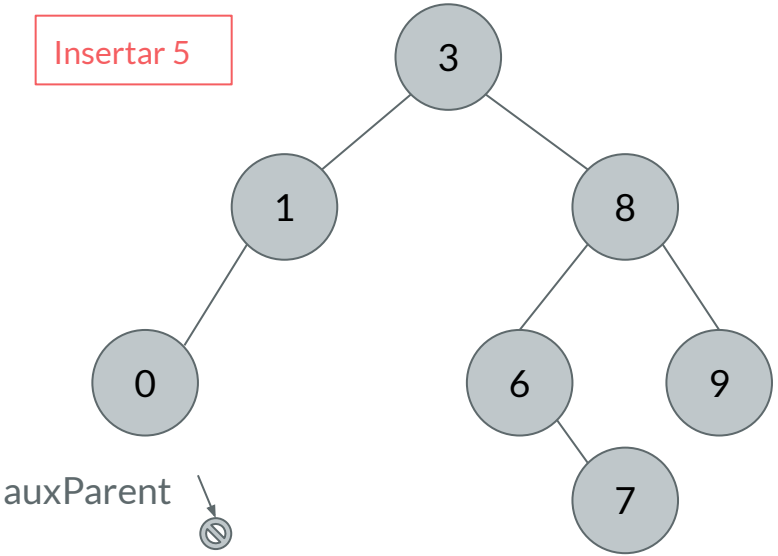

```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

Insertar 5

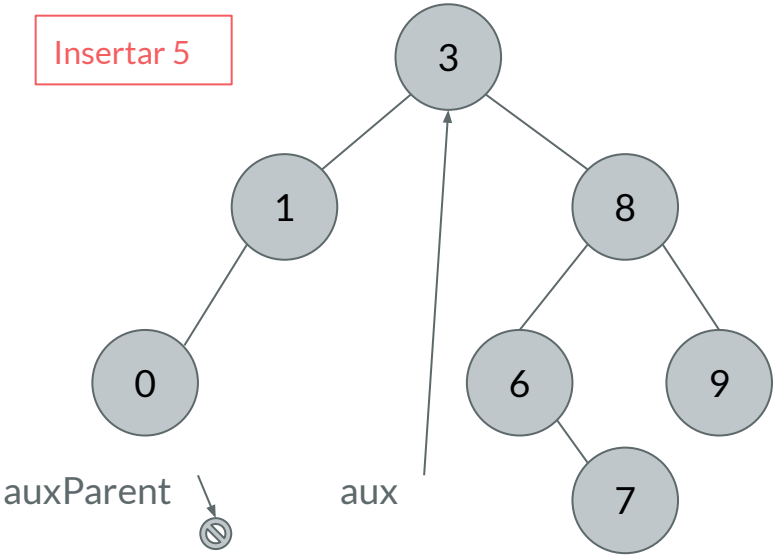



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

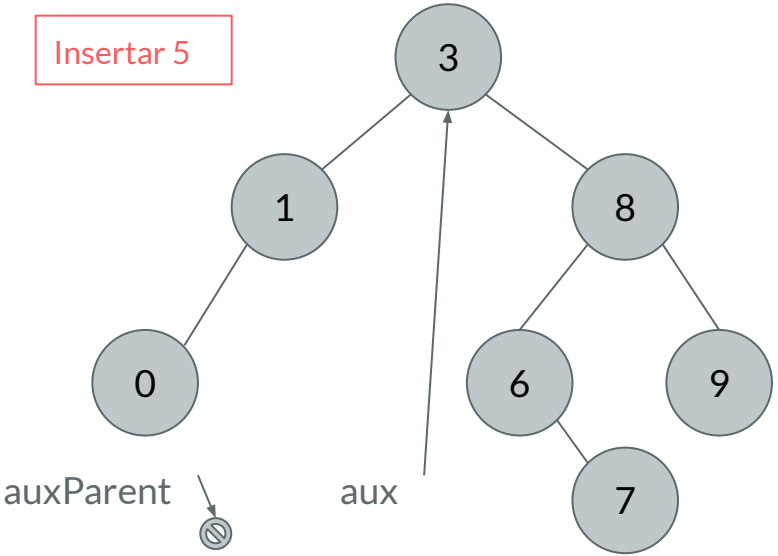



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

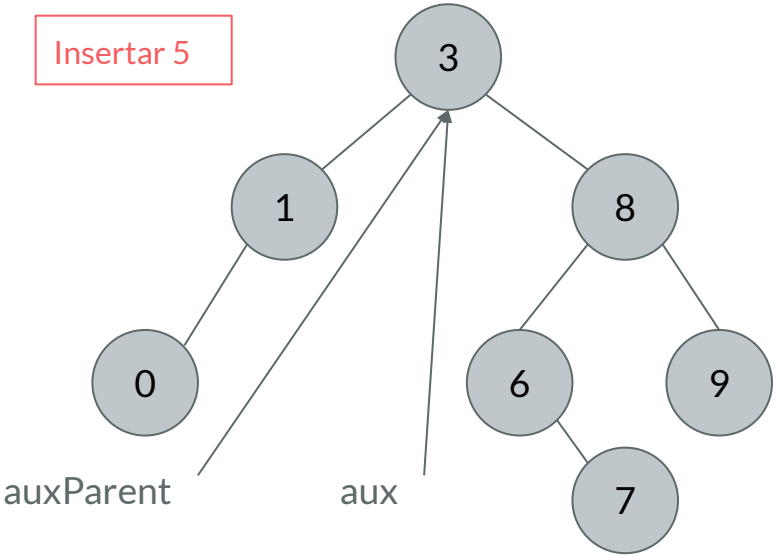



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

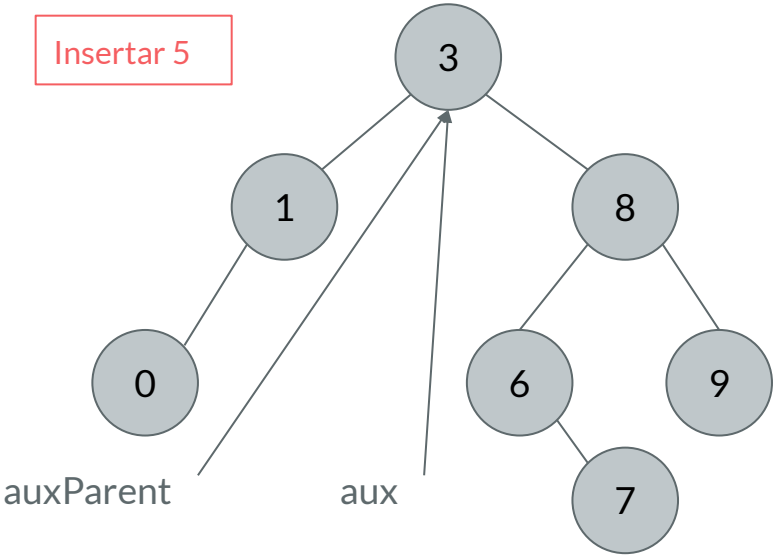



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

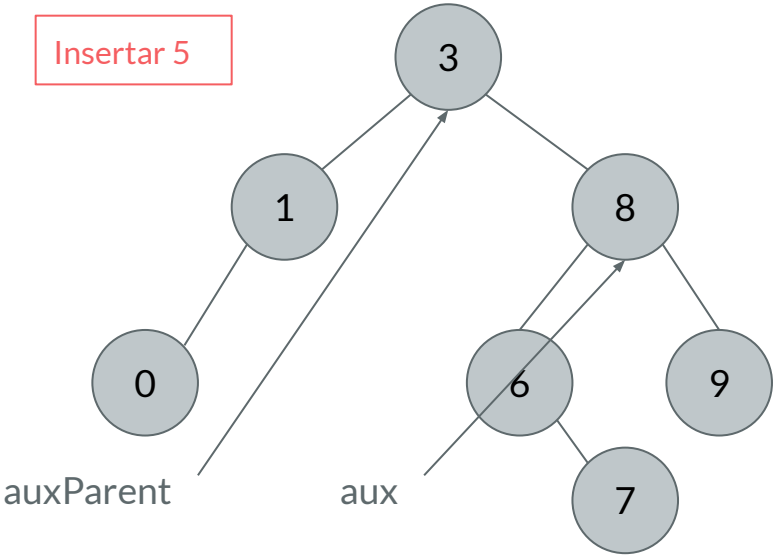



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

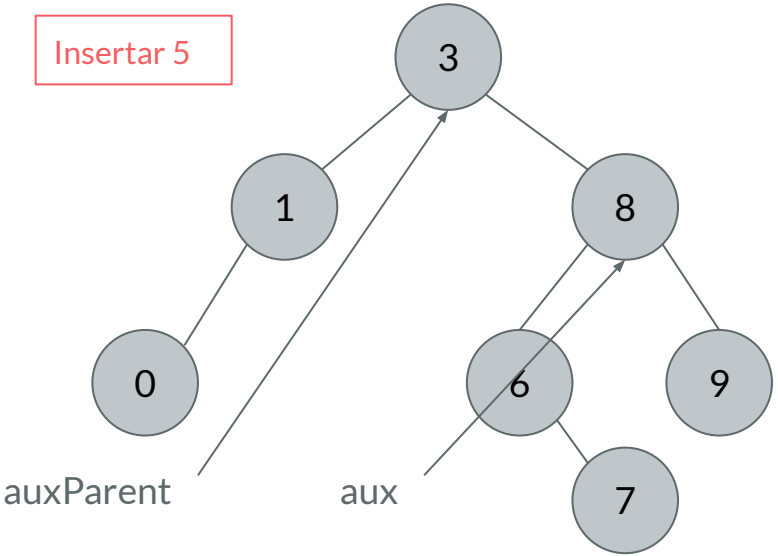



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

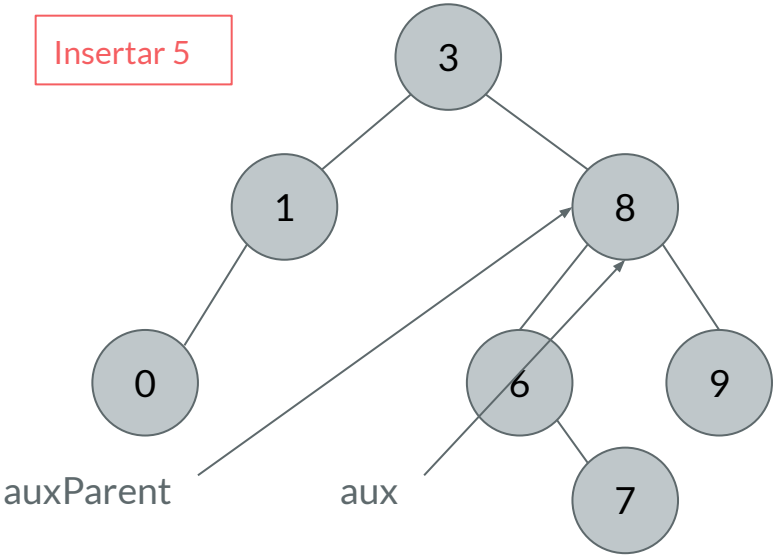



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

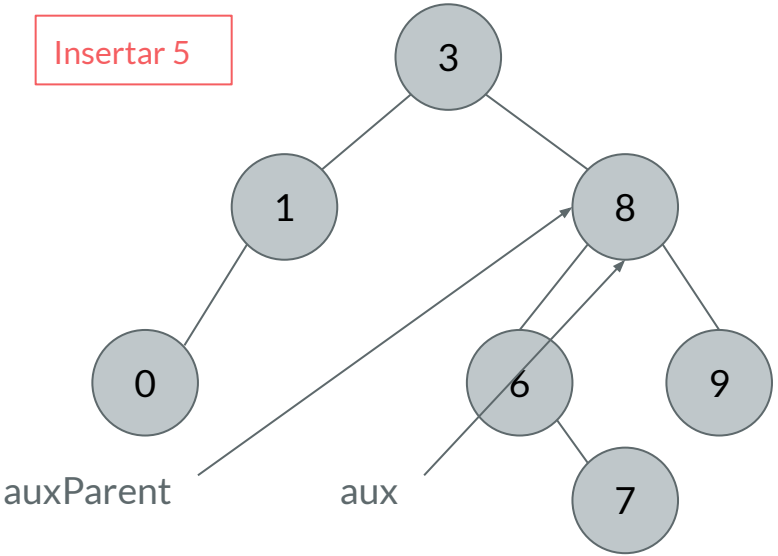



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

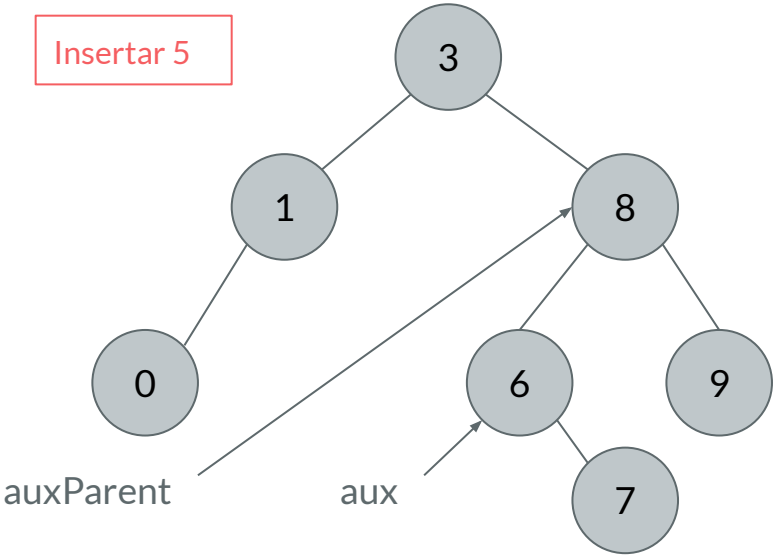



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

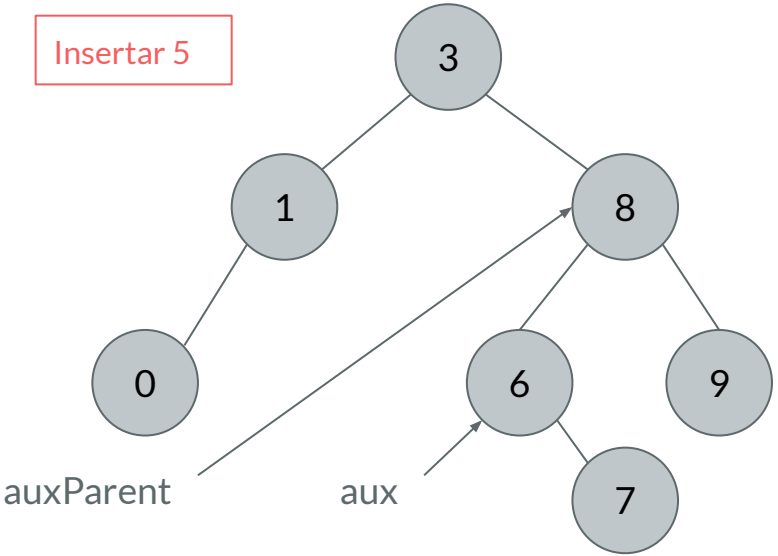



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

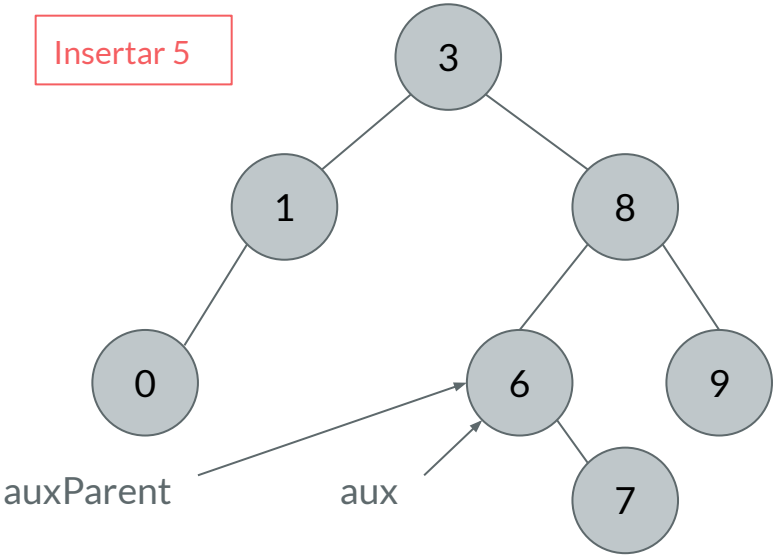



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

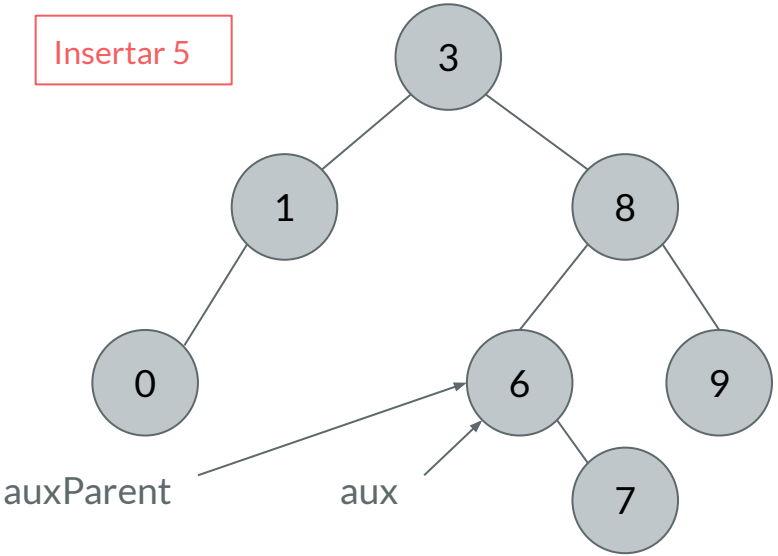



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

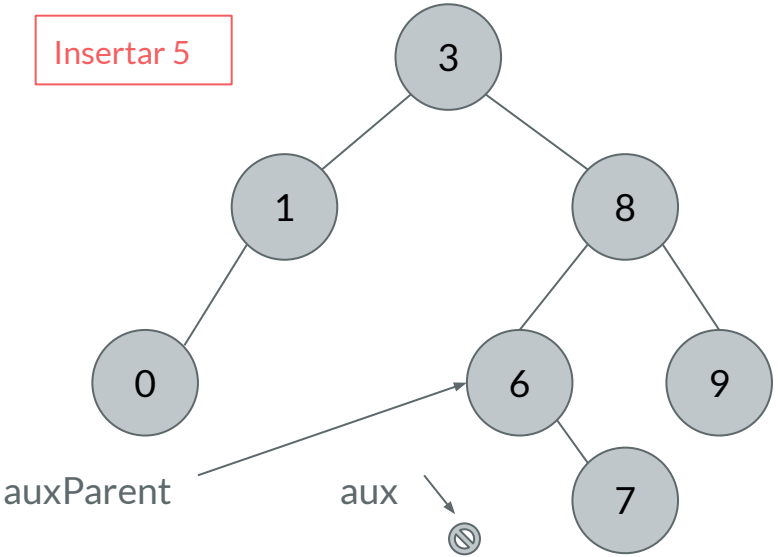



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura



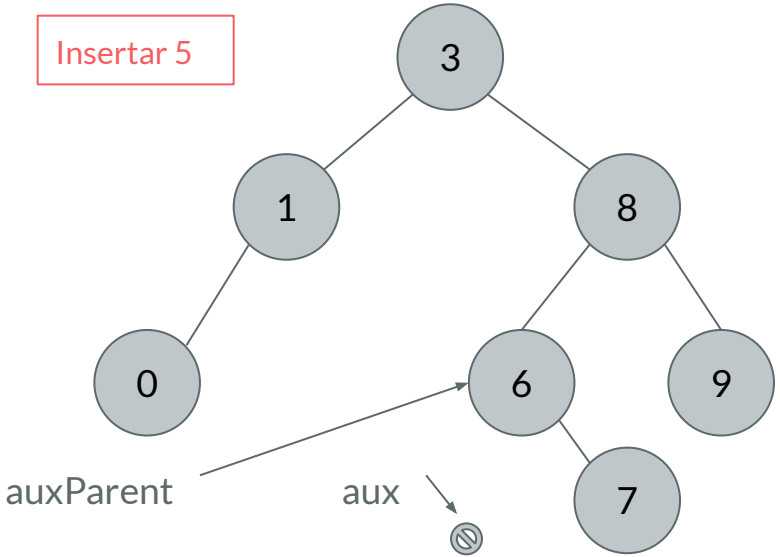

```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

Insertar 5

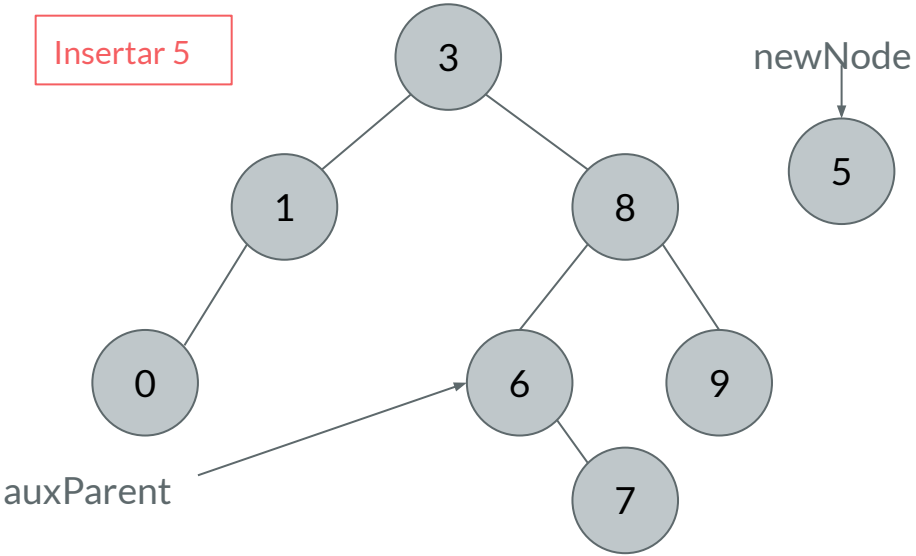



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

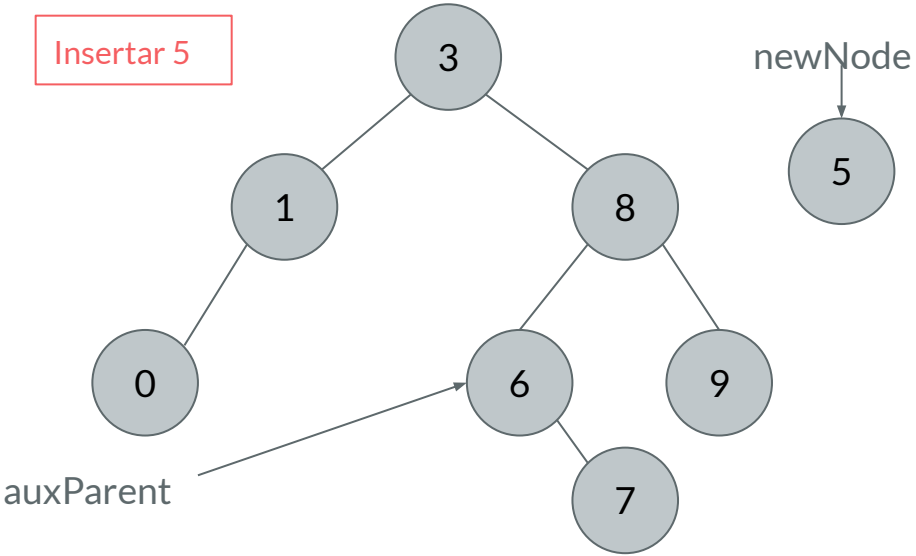



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
   auxParent = aux
   if value < aux.value
      aux = aux.left
   else
      aux = aux.right

Node newNode(value)
if auxParent == null
   root = newNode
else if value < auxParent.value
   auxParent.left = newNode
else
   auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

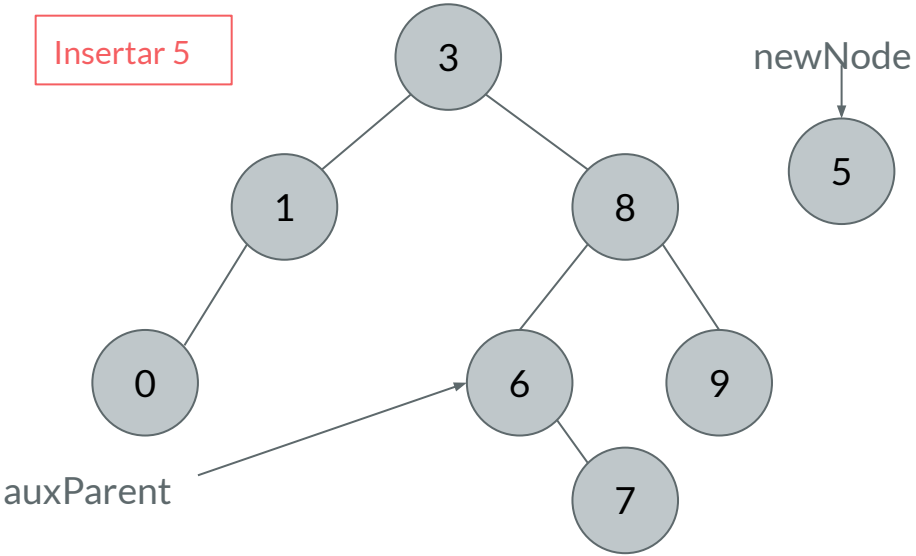



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
   auxParent = aux
   if value < aux.value
      aux = aux.left
   else
      aux = aux.right

Node newNode(value)
if auxParent == null
   root = newNode
else if value < auxParent.value
   auxParent.left = newNode
else
   auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

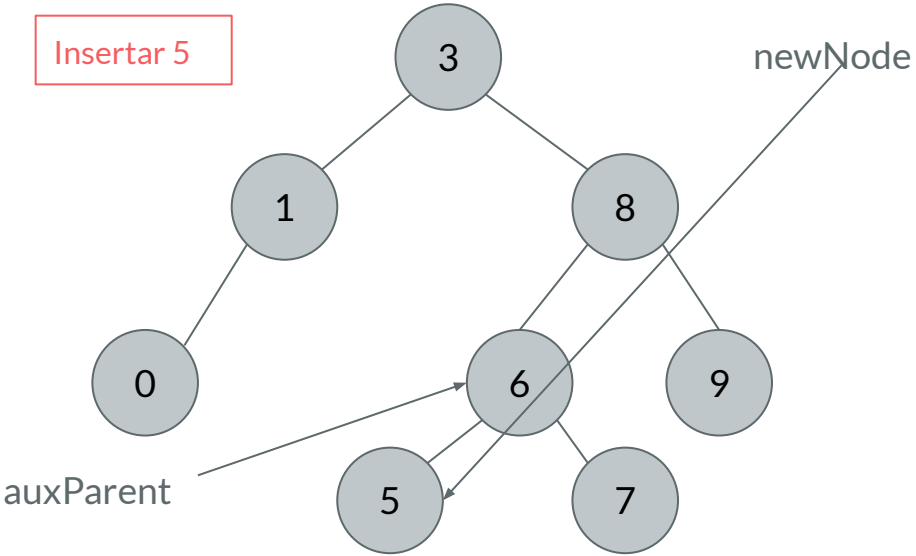



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura

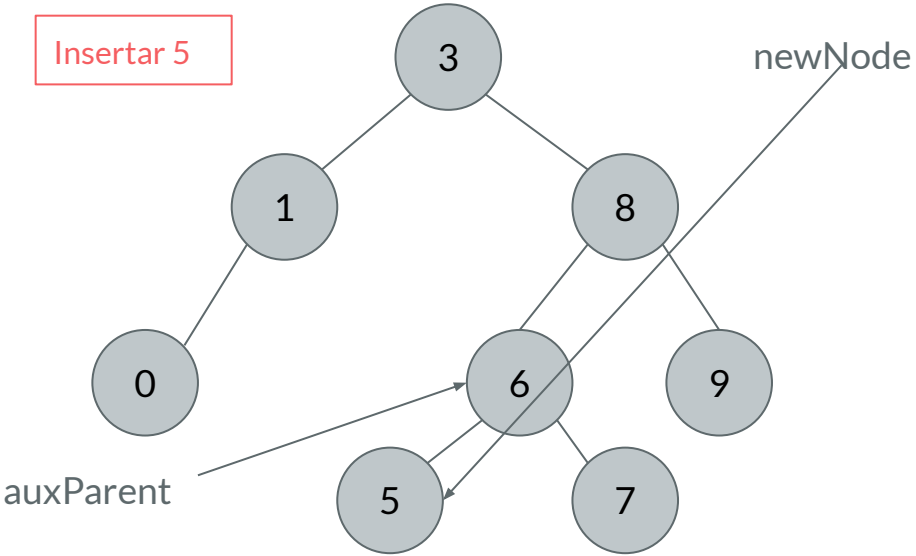



```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo recursivo o iterativo
  - La complejidad del algoritmo es O(h), donde h es la altura



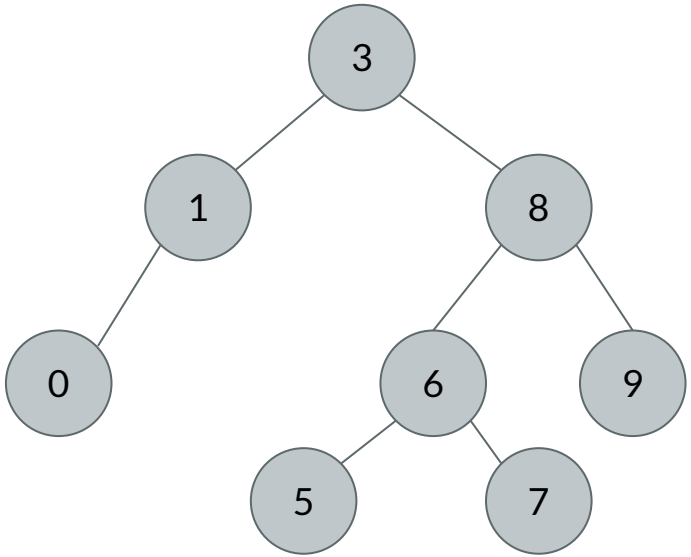

```
Insert(value)
Node auxParent = null
Node aux = root
while aux != null
  auxParent = aux
  if value < aux.value
    aux = aux.left
  else
    aux = aux.right

Node newNode(value)
if auxParent == null
  root = newNode
else if value < auxParent.value
  auxParent.left = newNode
else
  auxParent.right = newNode
newNode.parent = auxParent.
```

Árbol binario de búsqueda

- El mínimo de un subárbol de un ABB corresponde a su nodo más a la izquierda.

- Obsérvese que, gracias a la estructura del árbol binario de búsqueda, no es necesario comparar valores



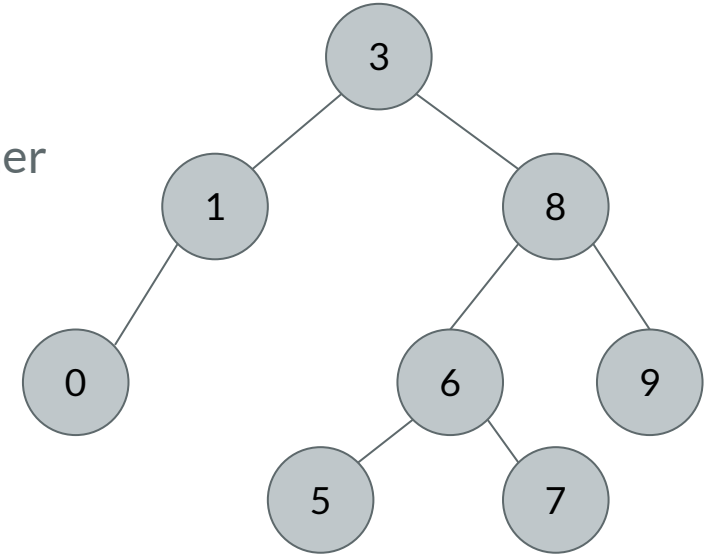

```
Minimum(node)
while node.left
   node = node.left
return node
```

Árbol binario de búsqueda

- Si un nodo tiene hijo derecho, el sucesor es el mínimo del subárbol derecho.

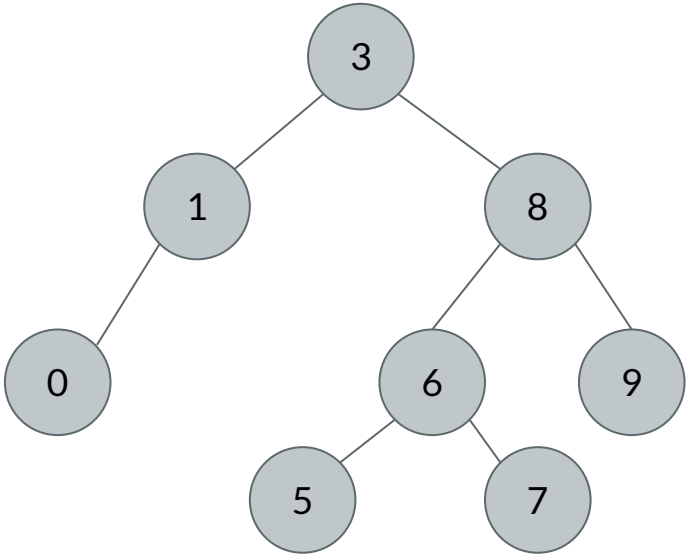
- Si no tiene hijo derecho, su sucesor será el primer ancestro mayor que él.

```
Successor(node)
if node.right
   return Minimum(node.right)
Node aux = node.parent
while aux and node == aux.right
   node = aux
   aux = aux.parent
return aux
```



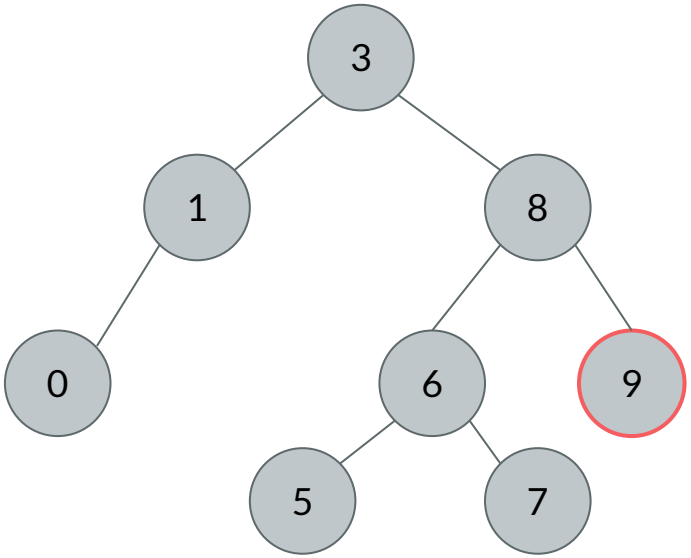
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



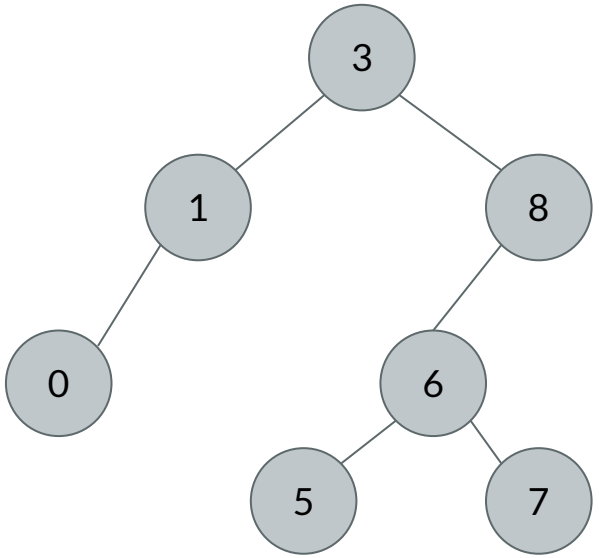
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



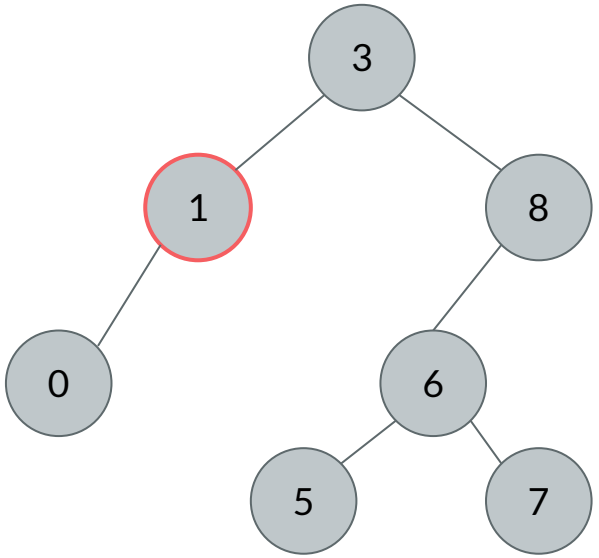
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



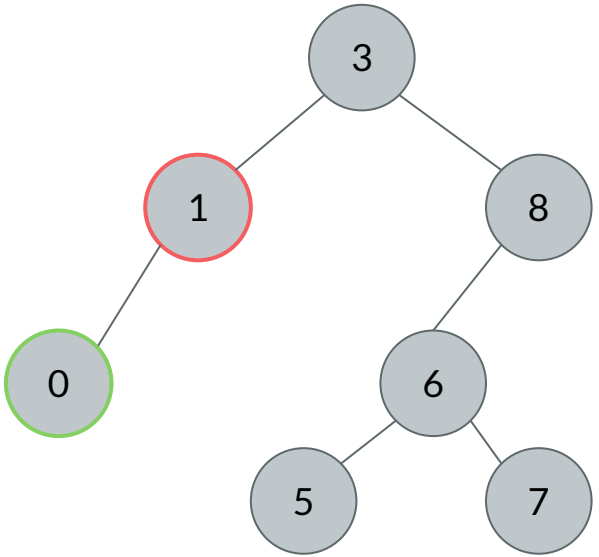
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



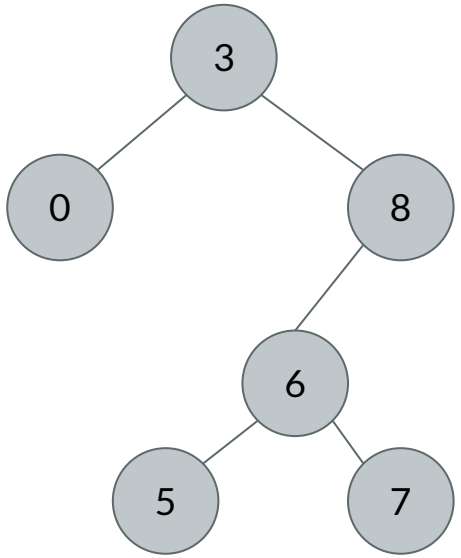
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



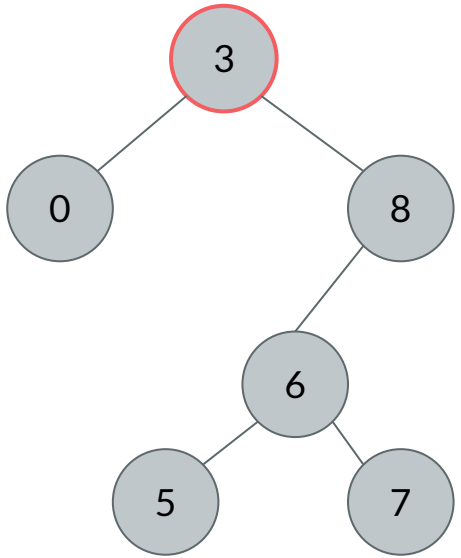
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



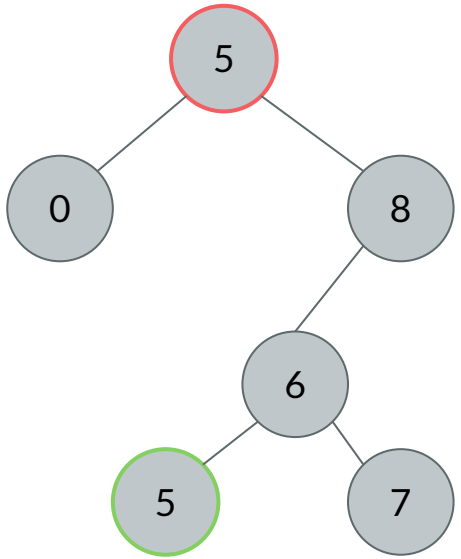
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



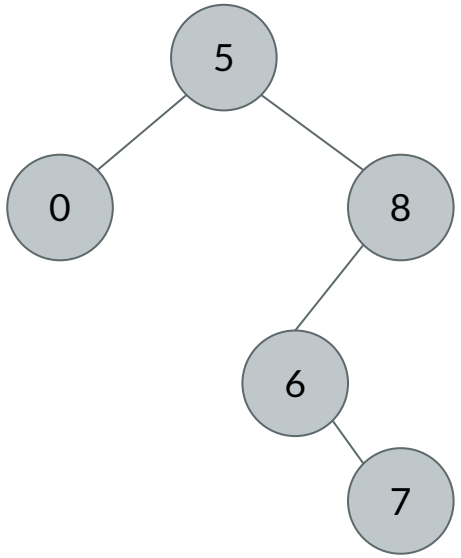
Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



Árbol binario de búsqueda

- El borrado tiene 3 casos:
  - El nodo a borrar es una hoja
    - Se elimina el nodo directamente
  - El nodo a borrar solo tiene un hijo
    - Se reemplaza el nodo por su hijo
  - El nodo a borrar tiene dos hijos
    - Se pone al nodo el valor de su sucesor
    - Se borra el sucesor del subárbol derecho.



Índice

- Árboles binarios de búsqueda

- **Mapas y Diccionarios ordenados**

- Equilibrado de árboles
  - Árboles AVL
  - Árboles Rojo-Negro

Mapas y Diccionarios ordenados

- Almacenan pares clave-valor permitiendo la búsqueda eficiente por clave.
- En un mapa no se permiten repeticiones en un diccionario si.
- Los elementos tiene un orden, se debe proporcionar el criterio de ordenación.
- En c++ se implementan como set, map, multiset y multimap (no confundir con las versiones “unordered_”).

Mapas y Diccionarios ordenados

- La búsqueda de un elemento es O(logN).

- Pero permiten la búsqueda por rango.

- El criterio de elección entre las versiones ordenadas (map, ...) y desordenadas (unordered_map, ...):
  - Si solo se necesita acceso a un elemento y se hace habitualmente: unordered_map
  - Si necesito recorrer en un orden, preguntar por un rango, acceder al siguiente, etc.: map

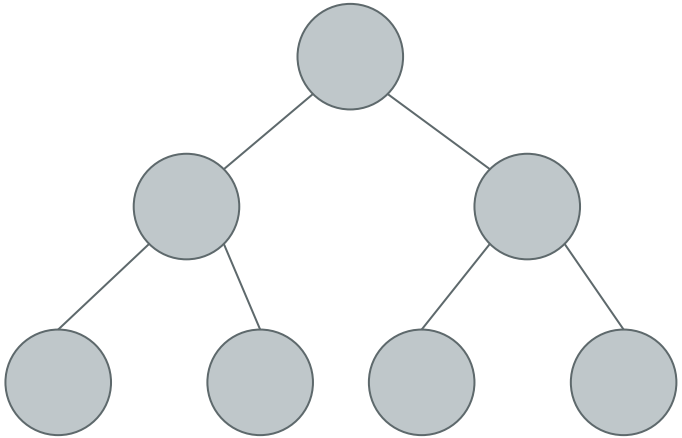
Índice

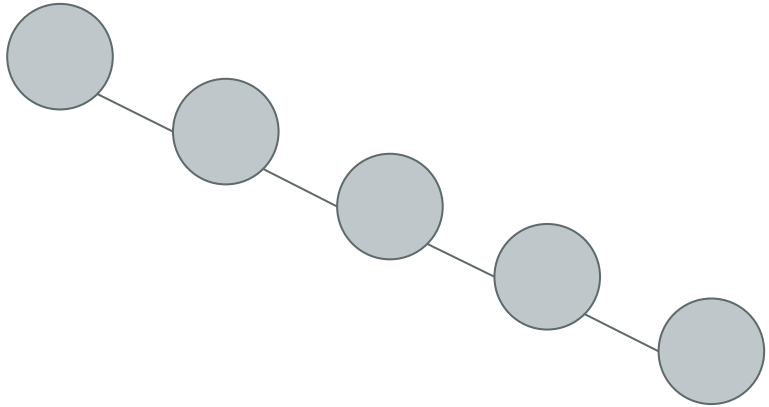
- Árboles binarios de búsqueda
- Mapas y Diccionarios ordenados
- Equilibrado de árboles
  - Árboles AVL
  - Árboles Rojo-Negro

Equilibrado de ABB

- Si bien la complejidad de las operaciones de un árbol binario no cambian, la velocidad depende enormemente del equilibrio del árbol.

- Hay muchos criterios de equilibrio suelen estar relacionados con la altura.





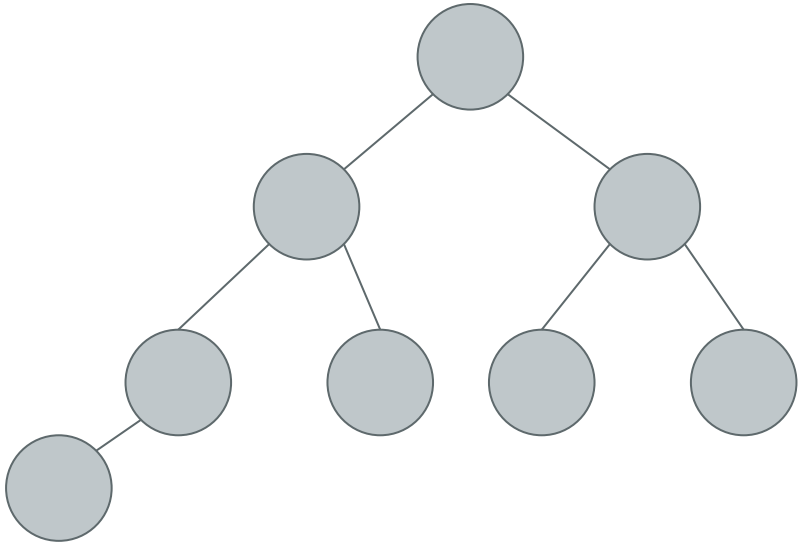
Equilibrado de ABB

- Los árboles equilibrados o balanceados surgen para mejorar el rendimiento de operaciones que involucren una búsqueda pues mantienen la altura logarítmica

- Se estudiarán dos tipos de árboles equilibrados:
  - Árboles AVL (Adelson-Velskii y Landis)
  - Árboles Rojo Negro

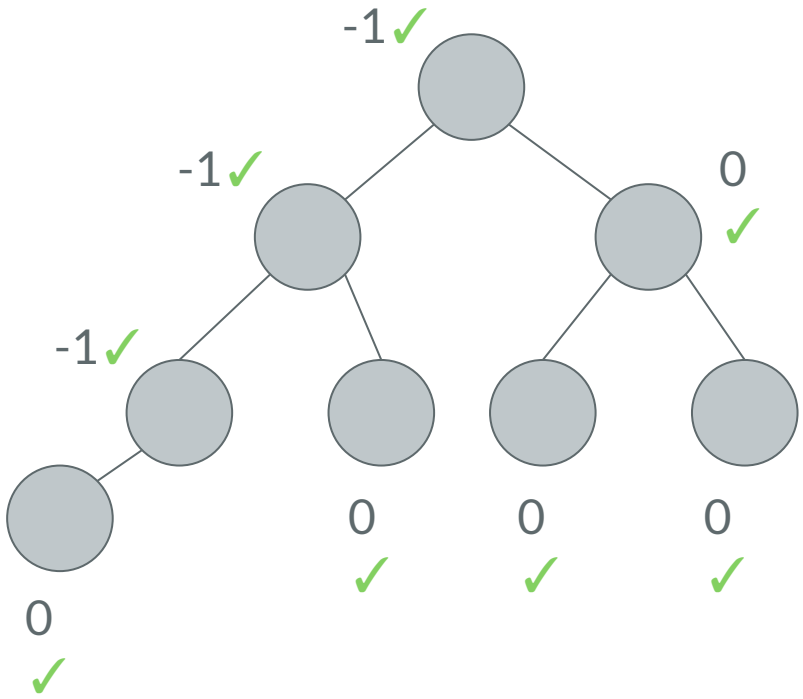
Árboles AVL

- Georgy Adelson-Velsky y Evgenii Landis tuvieron la idea de equilibrar un árbol de manera que el subárbol izquierdo y el sub-árbol derecho de cualquier nodo tuviesen, aproximadamente, la misma altura.



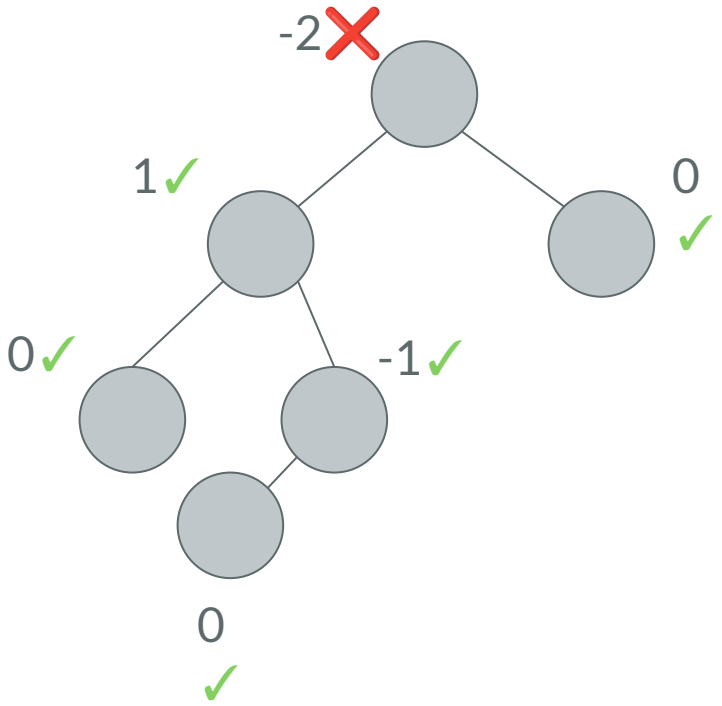
Árboles AVL

- Un árbol AVL es un árbol binario de búsqueda con una condición de equilibrio.
  - Para cada nodo, la altura de los 2 subárboles no difiere en más de una unidad.
  - El Factor de equilibrio o balance de un nodo se define como la diferencia de altura de sus dos subárboles (derecho - izquierdo).
  - Cada nodo de un AVL puede tener un balance de -1, 0 ó 1.
  - Para que el cálculo del factor sea eficiente, en cada nodo hay que mantener su altura.



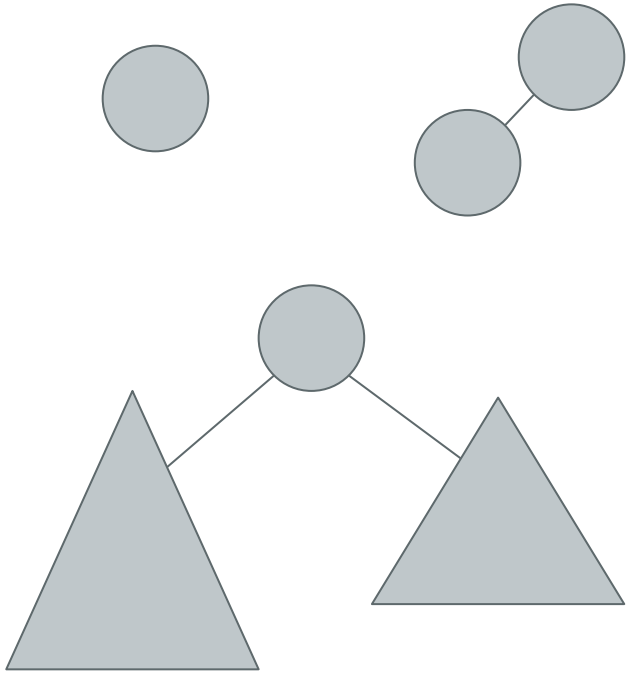
Árboles AVL

- Un árbol AVL es un árbol binario de búsqueda con una condición de equilibrio.
  - Para cada nodo, la altura de los 2 subárboles no difiere en más de una unidad.
  - El Factor de equilibrio o balance de un nodo se define como la diferencia de altura de sus dos subárboles (derecho - izquierdo).
  - Cada nodo de un AVL puede tener un balance de -1, 0 ó 1.
  - Para que el cálculo del factor sea eficiente, en cada nodo hay que mantener su altura.



Árboles AVL

- La altura de un ABB-AVL con n nodos es O(logN).
  - Si n(h) es el número mínimo de nodos de un ABB-AVL con altura h:
    - n(1) = 1
    - n(2) = 2
    - n(h) = n(h-1) + n(h-2) + 1
  - Al ser mínimo, la altura de los subárboles de un nodo X de altura h siempre se diferencian en 1. Por eso se suma n(h-1) + n(h-2) añadiendo 1 por por el nodo X.
  - Esta fórmula corresponde con la serie de Fibonacci cuyo crecimiento es exponencial.

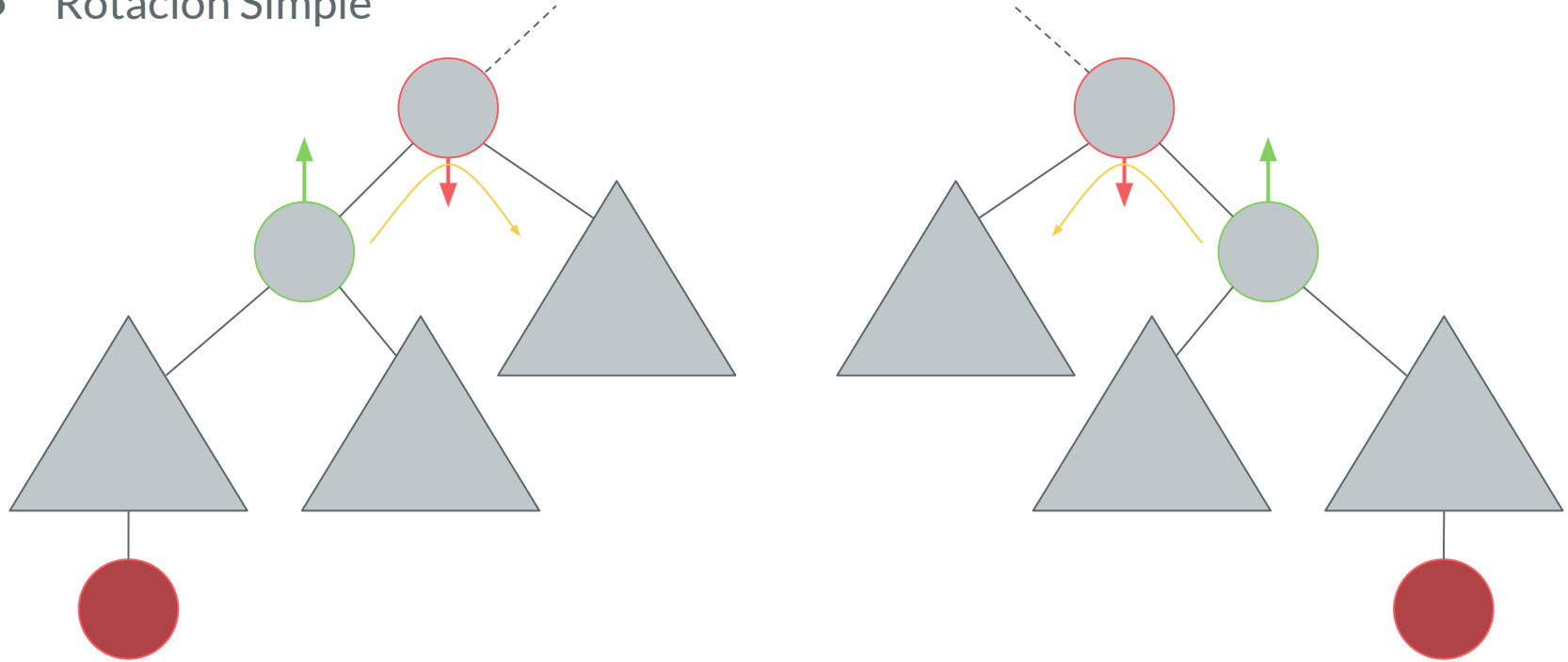


Árbol AVL

- Inserción:
  - Se inserta en ABB normal
  - Se comprueba la condición de equilibrio
  - Si no está en equilibrio se restaura aplicando rotaciones:
    - Rotación simple: izq-izq ó dcha-dcha
    - Rotación doble: izq-dcha ó dcha-izq

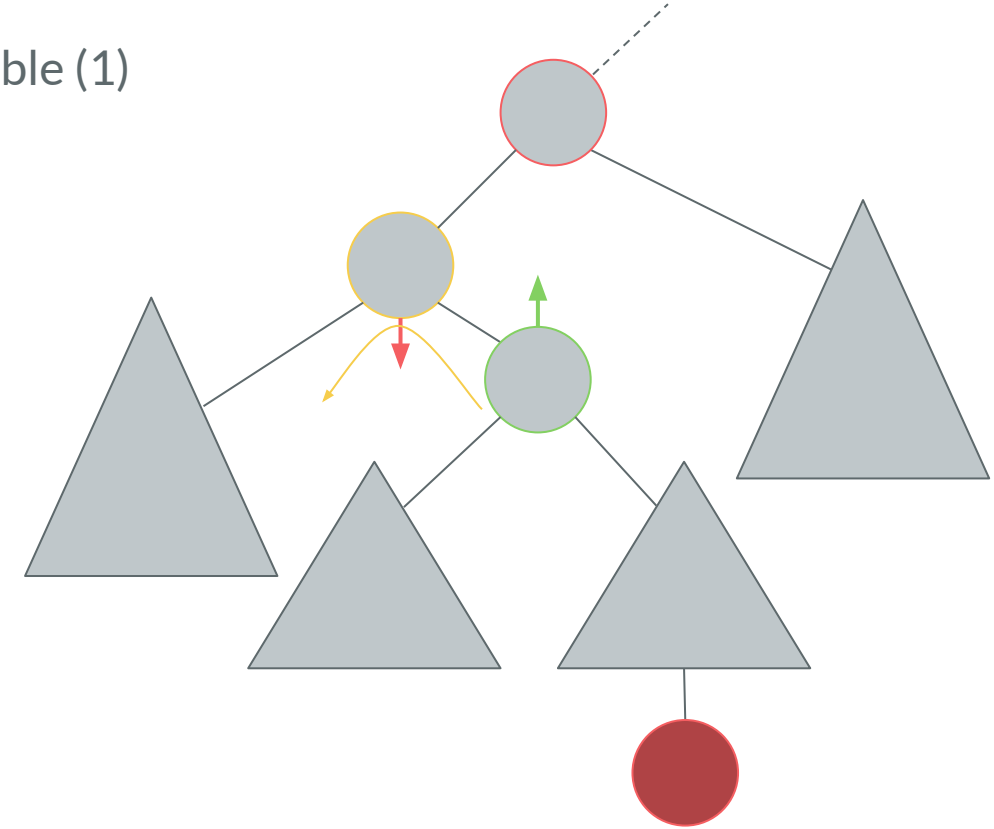
Árboles AVL

- Rotación Simple



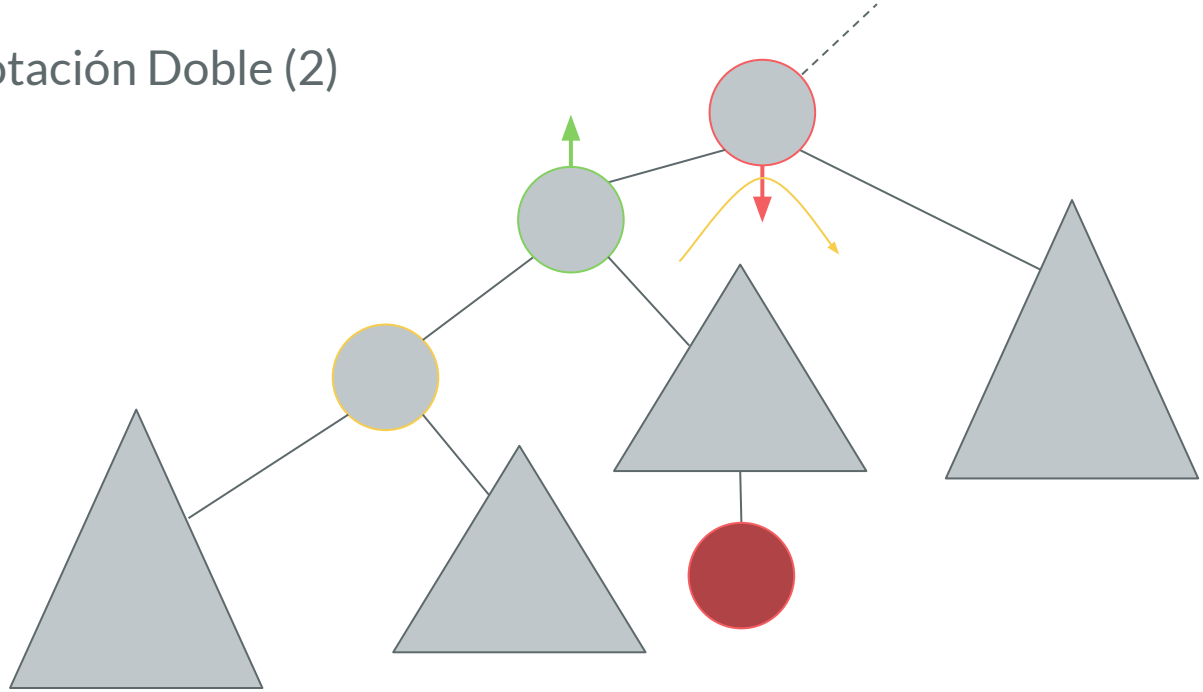
Árboles AVL

- Rotación Doble (1)



Árboles AVL

- Rotación Doble (2)



Árboles Rojo-Negro

- Sería deseable que los árboles tuviesen la misma profundidad partiendo de cualquier hoja.

- Ya que esto no siempre es posible, se plantea crear un árbol donde eso se cumpla excepto por unos pocos nodos que pintaremos de rojo.

- Por simplicidad de los algoritmos supondremos que los punteros a Nulo son hojas negras.

Árboles Rojo-Negro

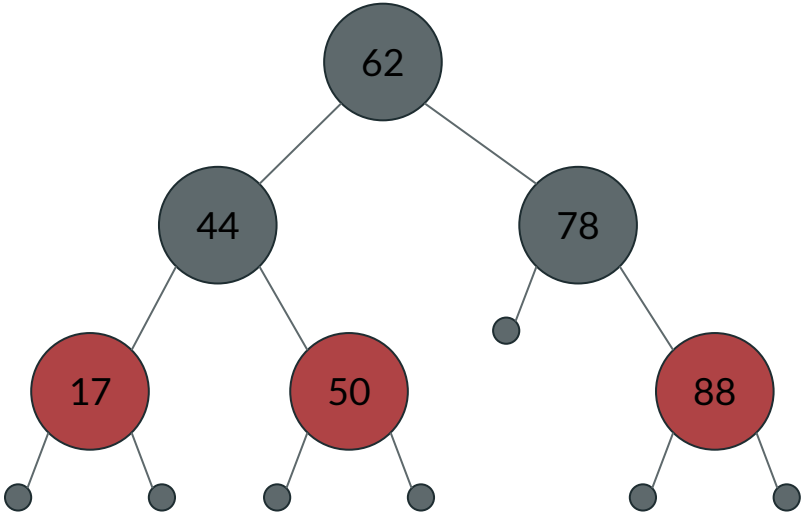
- ARN: árbol binario de búsqueda con nodos coloreados de rojo o negro

- La forma de colorear los nodos asegura que ningún camino (raíz-hoja) es más del doble de largo que otro

- Los borrados en ARN realizan menos operaciones de restructuración que los AVL

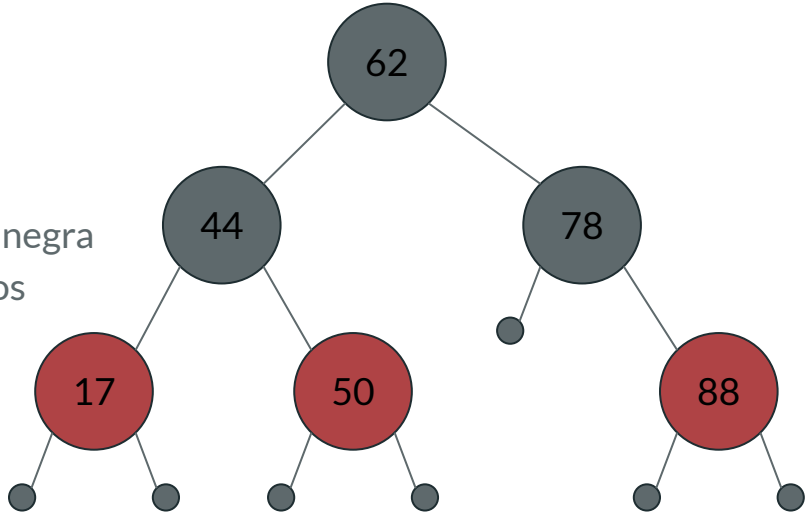
Árboles Rojo-Negro

- Cada nodo contiene:
  - color
  - valor
  - hijo izquierdo
  - hijo derecho
  - padre



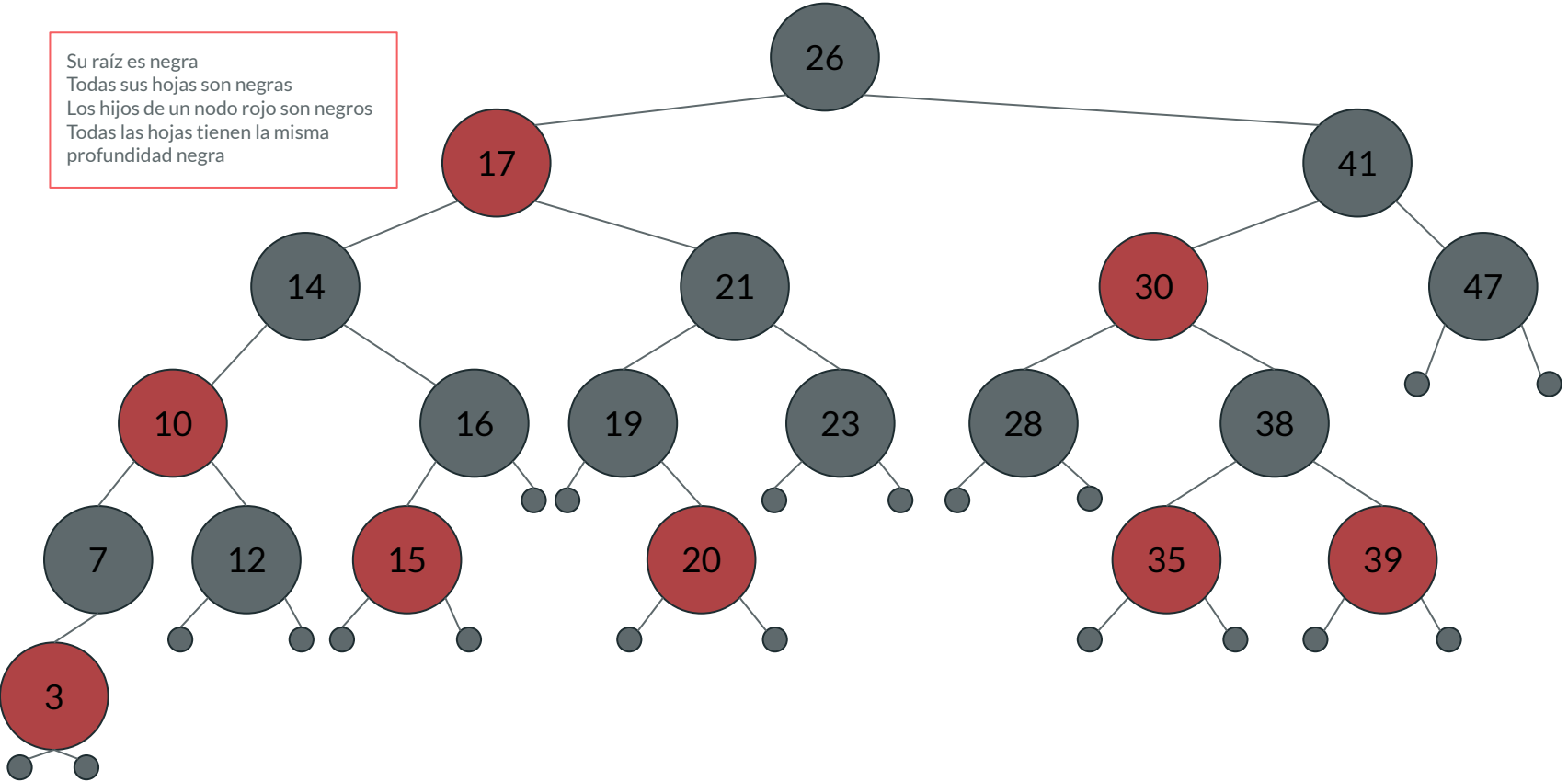
Árboles Rojo-Negro

- Propiedades de los ARN
  - Su raíz es negra
  - Todas sus hojas (nodos nulos) son negros
  - Los hijos de un nodo rojo son negros
  - Todas las hojas tienen la misma profundidad negra
    - Profundidad negra: antepasados negros



Árboles Rojo-Negro

Su raíz es negra
Todas sus hojas son negras
Los hijos de un nodo rojo son negros
Todas las hojas tienen la misma profundidad negra



Árboles Rojo-Negro

- Altura negra de un nodo x, bh(x): número de nodos negros de los caminos entre el nodo x y las hojas, sin incluir a x.

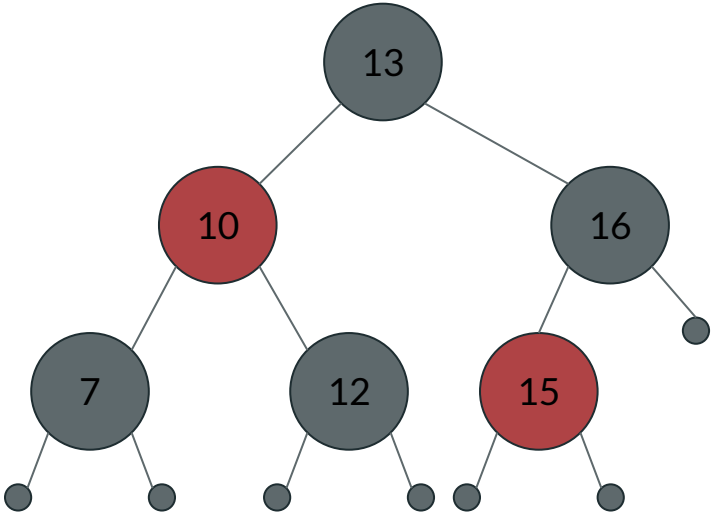
- Altura negra de un árbol rojo-negro: altura negra de su nodo raíz.

- Un árbol rojo-negro que almacena n elementos tiene una altura h que verifica $\log_2(n+1) \leq h \leq 2\log_2(n+1)$

- La búsqueda de una elemento en un árbol rojo negro es $O(\log_2 n)$

Árboles Rojo-Negro

- Inserción en un ARN
  - Se inserta como en un ABB
  - El nodo insertado se colorea de rojo
  - Si es la raíz se deja negro
  - Ejemplo: insertar el valor 3

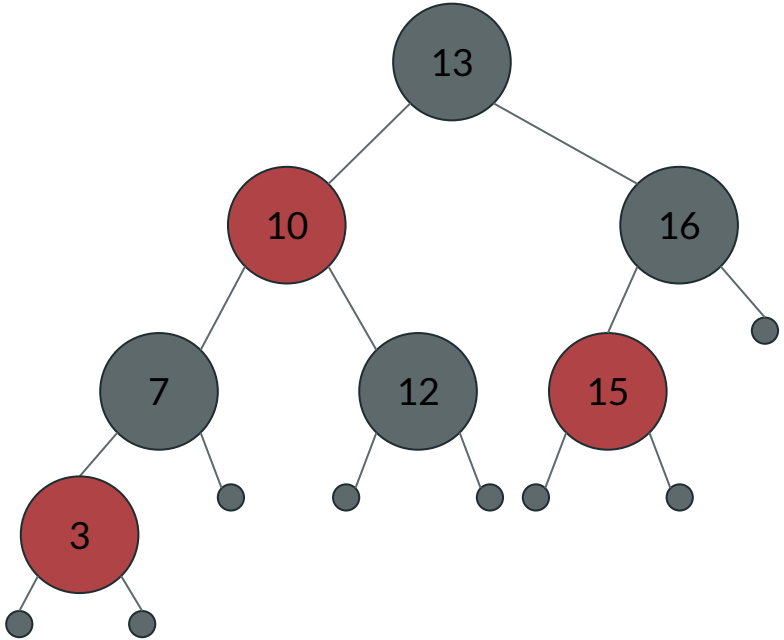


Árboles Rojo-Negro

- Inserción en un ARN
  - Se inserta como en un ABB
  - El nodo insertado se colorea de rojo
  - Si es la raíz se deja negro

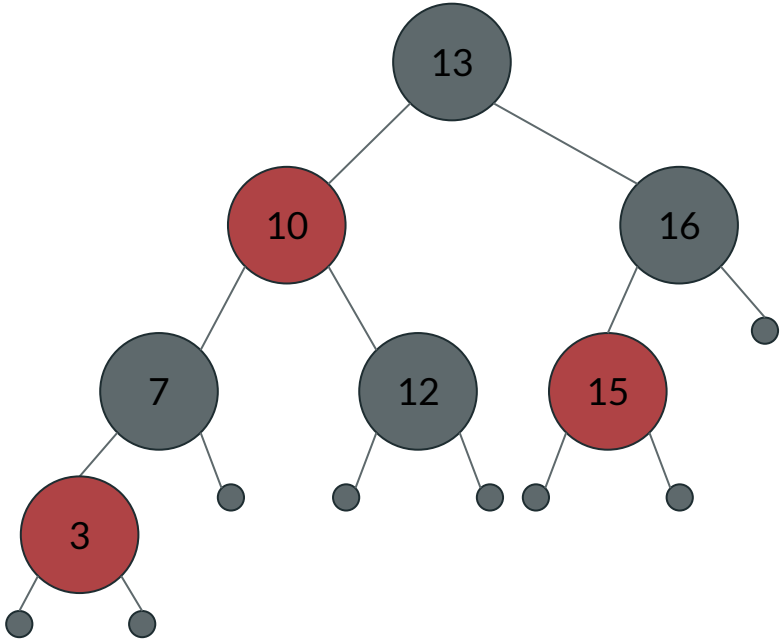
  - Ejemplo: insertar el valor 3

Si su padre es negro, se stisfacen todas las propiedades



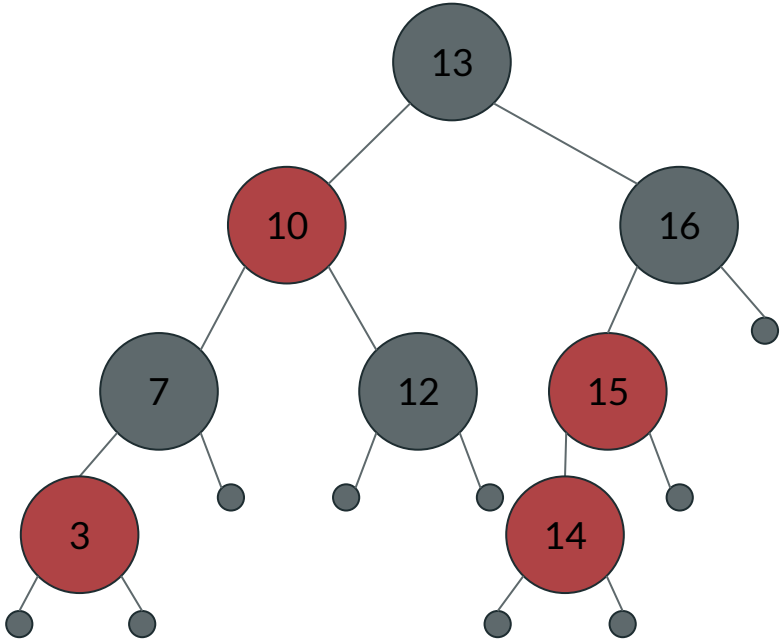
Árboles Rojo-Negro

- Inserción en un ARN
  - Se inserta como en un ABB
  - El nodo insertado se colorea de rojo
  - Si es la raíz se deja negro
  - Ejemplo: insertar el valor 14



Árboles Rojo-Negro

- Inserción en un ARN
  - Se inserta como en un ABB
  - El nodo insertado se colorea de rojo
  - Si es la raíz se deja negro
  - Ejemplo: insertar el valor 14

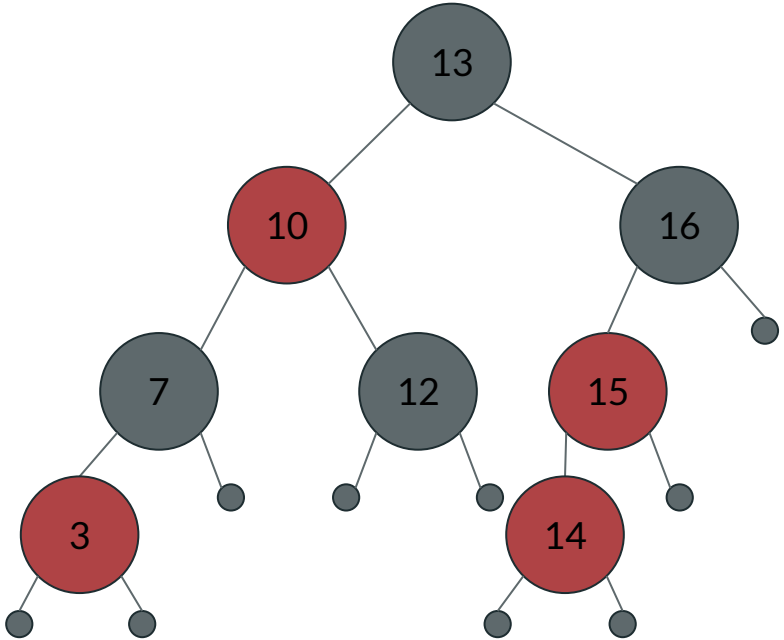


Árboles Rojo-Negro

- Inserción en un ARN
  - Se inserta como en un ABB
  - El nodo insertado se colorea de rojo
  - Si es la raíz se deja negro

  - Ejemplo: insertar el valor 14

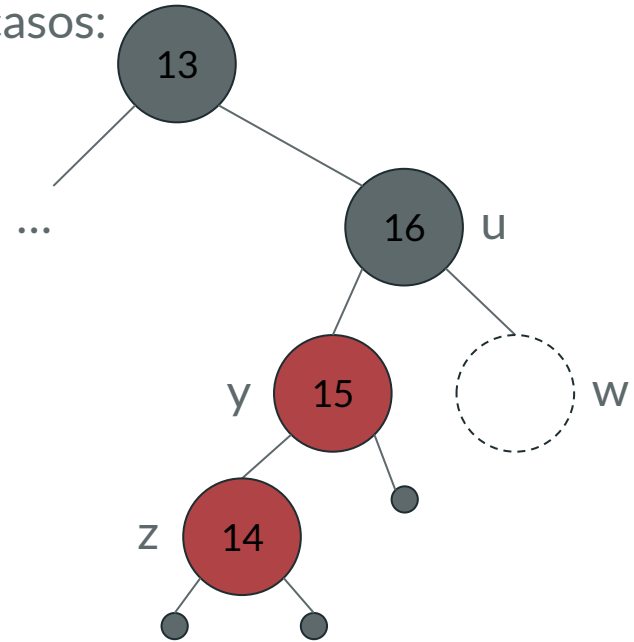
La propiedad 3 no se cumple, se produce un doble rojo



Árboles Rojo-Negro

- Para remediar el doble rojo se pueden cumplir 2 casos:
  - Caso 1: el tío del nodo insertado es negro
  - Caso 2: el tío del nodo insertado es rojo

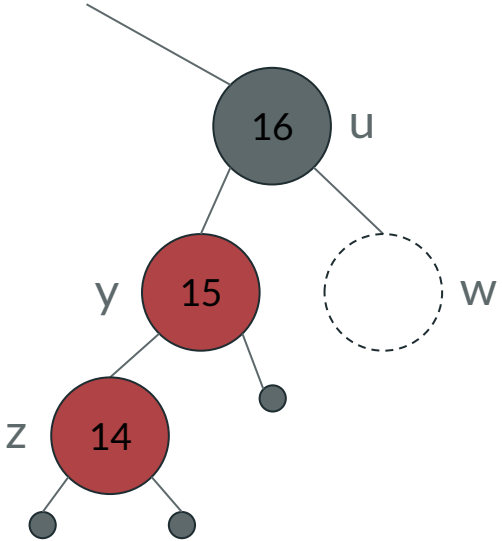
- Notación:
  - z: nodo insertado
  - v: padre de z
  - w: tío de z
  - u: abuelo de z



Árboles Rojo-Negro

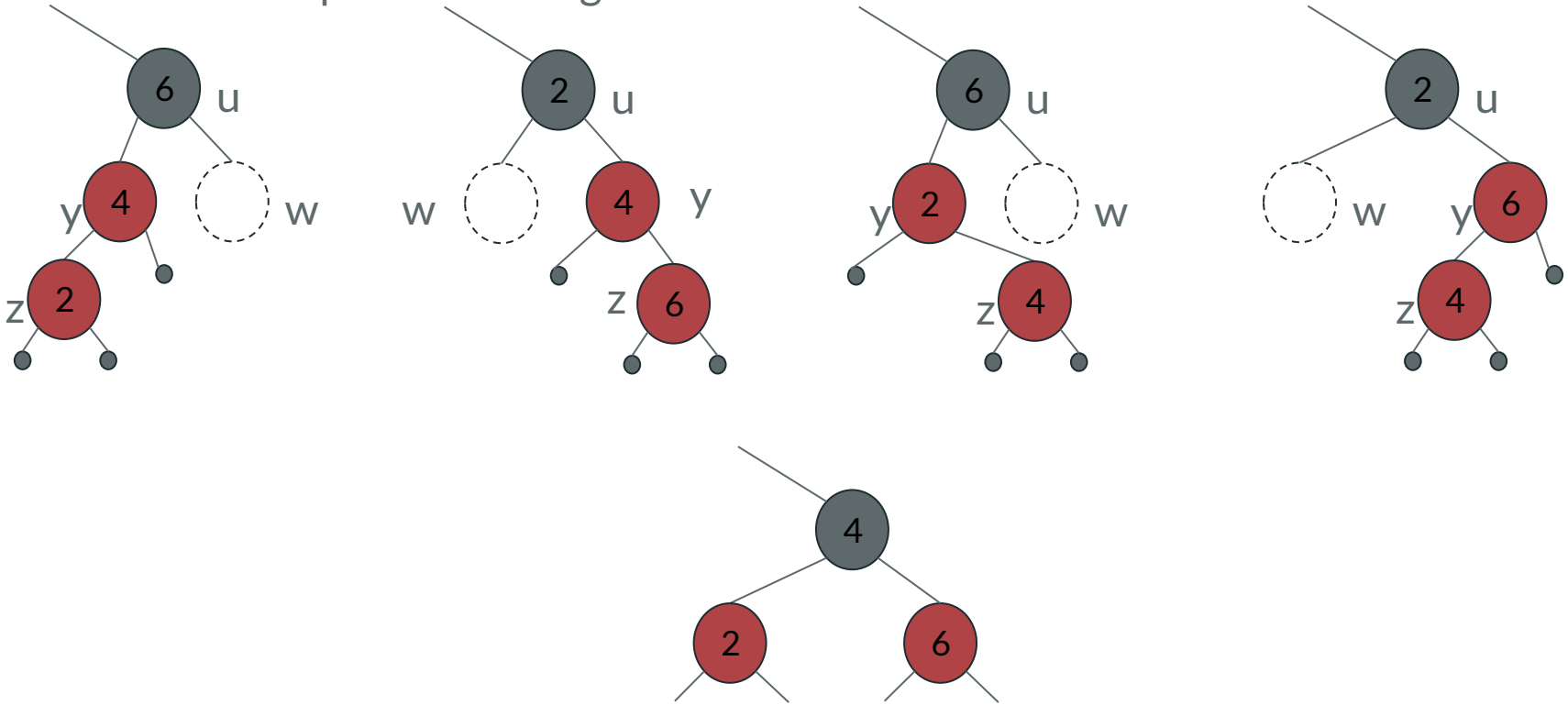
- Caso 1: el tío del nodo insertado es negro
  - Se lleva uno de los nodos del doble rojo a la posición del abuelo. Repartiendo los rojos se elimina el problema del doble rojo

- Para ello:
  - Aplicar reestructuración trinodo
    - Dado el nodo insertado z, su padre v y su abuelo u se reestructura
  - Colorear el nodo medio como negro y sus hijos de rojo



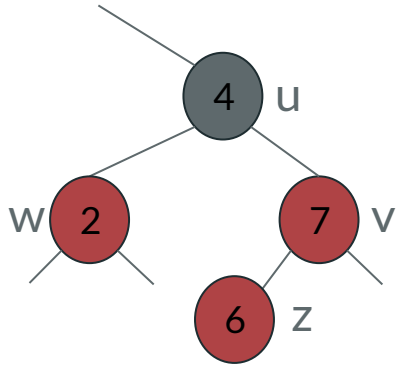
Árboles Rojo-Negro

- Solo cuatro posibles configuraciones del caso 1



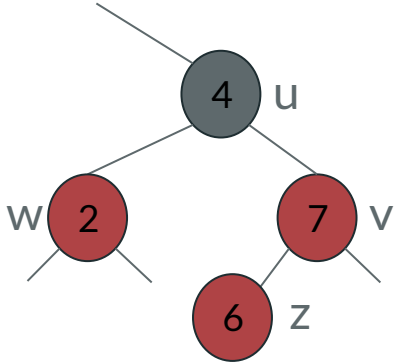
Árboles Rojo-Negro

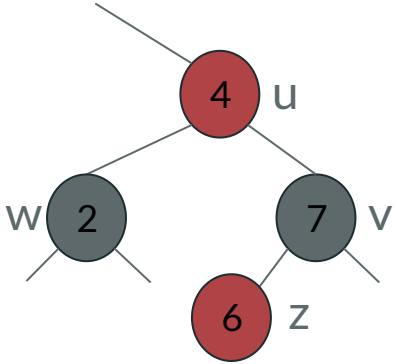
- Caso 2: el tío del nodo insertado es rojo
  - Un recoloreado a negro en ambas rama y un recoloreado a rojo en un ascendiente para no cambiar la altura negra.



Árboles Rojo-Negro

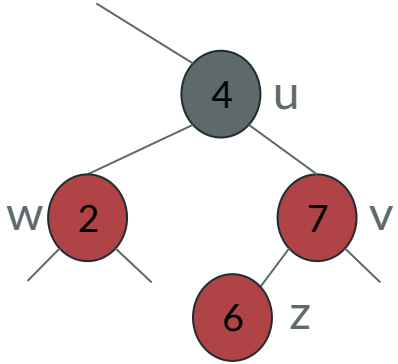
- Caso 2: el tío del nodo insertado es rojo
  - Un recoloreado a negro en ambas rama y un recoloreado a rojo en un ascendiente para no cambiar la altura negra.
- Para ello:
  - Aplicar un recoloreado
  - El padre v y el tío w se colorean a negro
  - El abuelo u se colorea de rojo (salvo si es raíz).

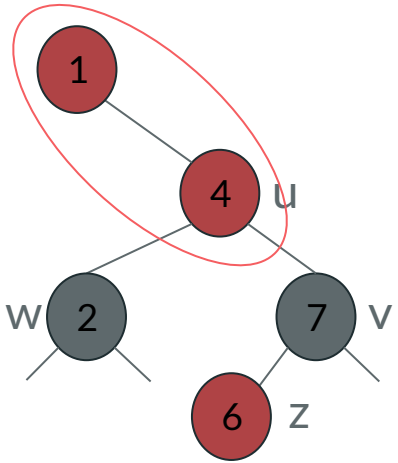




Árboles Rojo-Negro

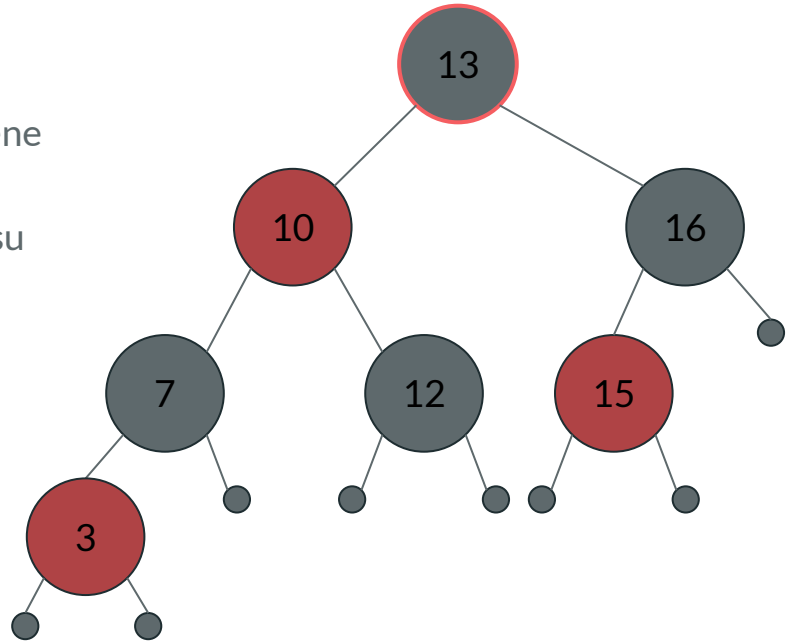
- Caso 2: el tío del nodo insertado es rojo
  - Tras el recoloreado, puede aparecer el problema del doble rojo en el abuelo “u”. Si aparece, aplicar el caso que corresponda (caso 1 o caso 2).
  - El recoloreado, elimina el problema del doble rojo o lo propaga





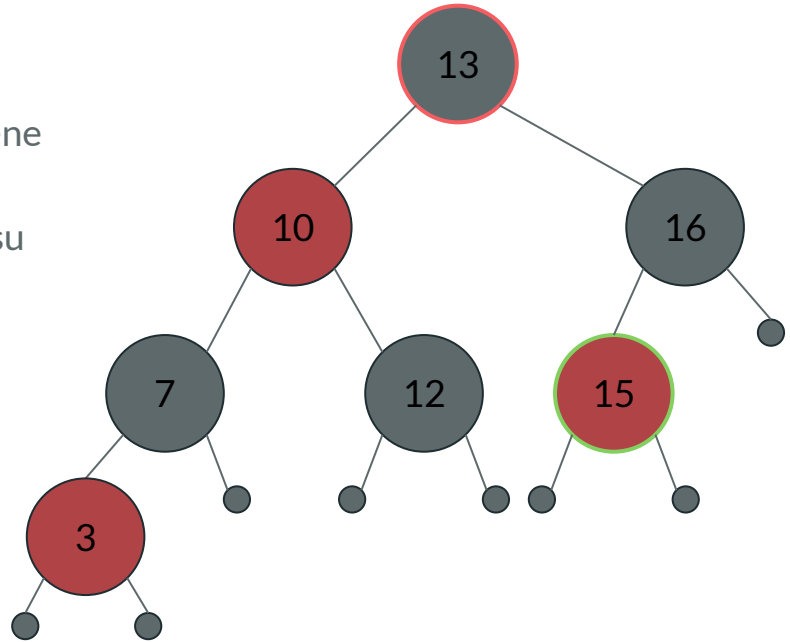
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 13



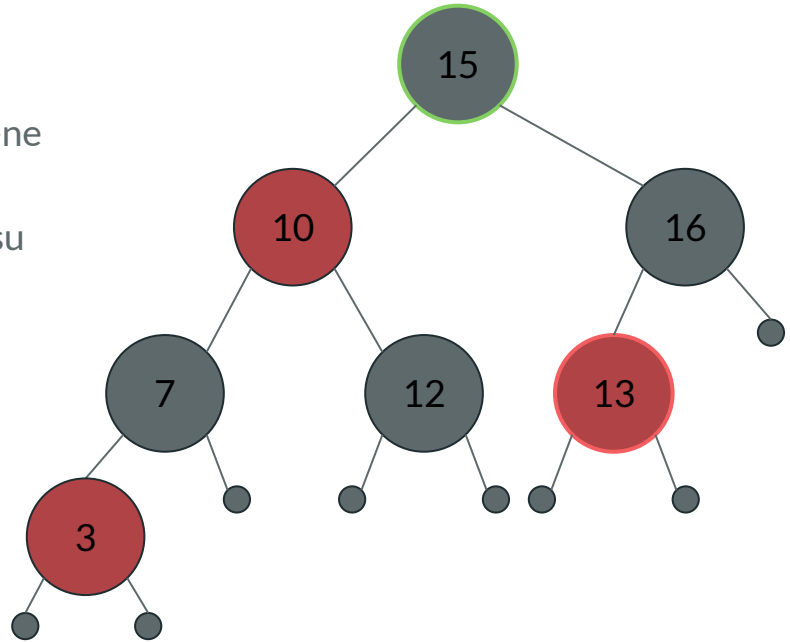
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 13



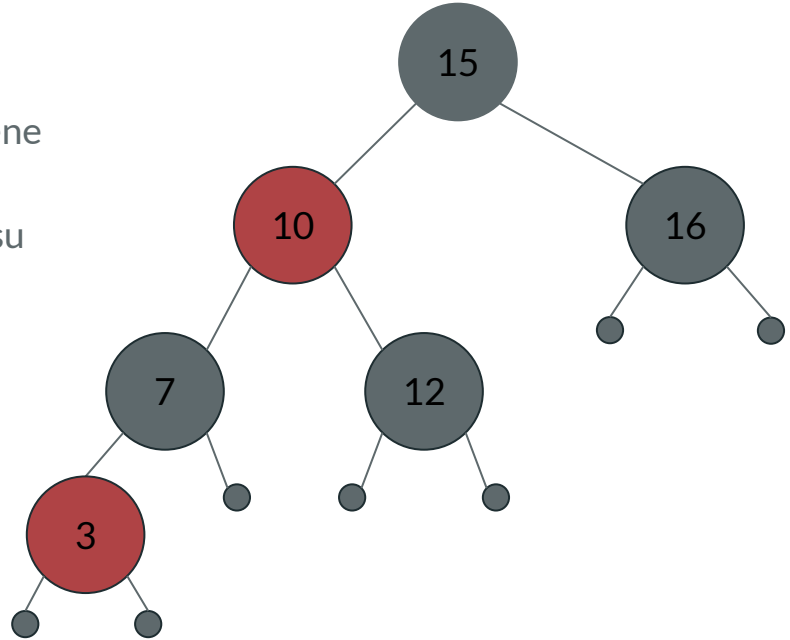
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 13



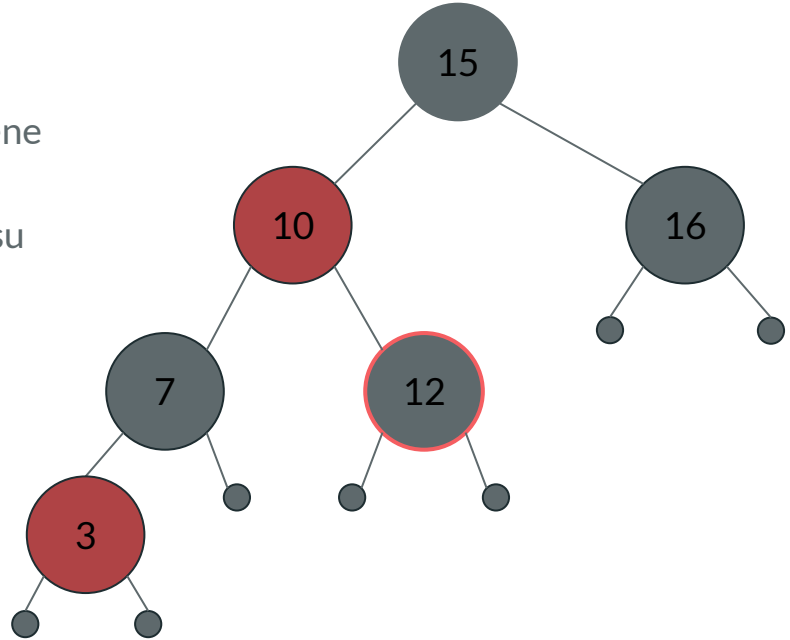
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 13



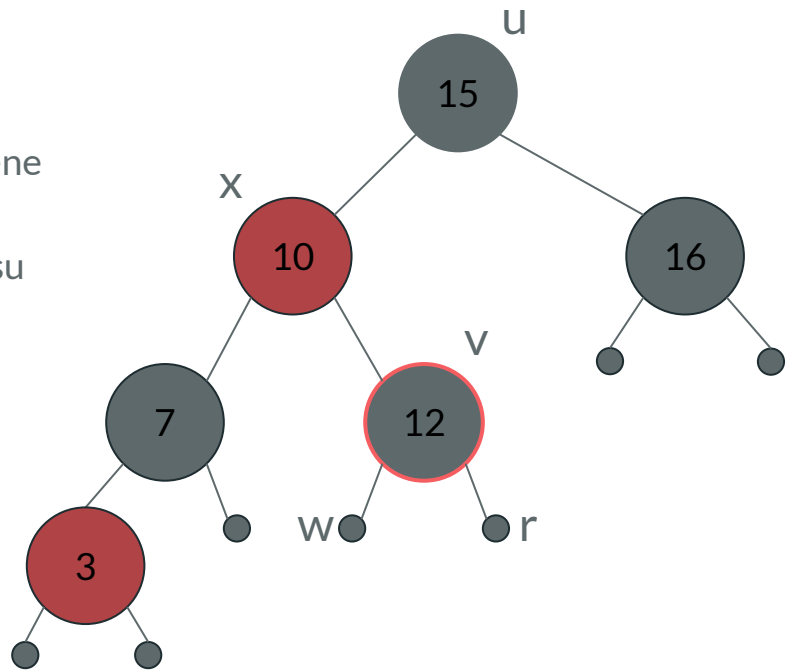
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 12



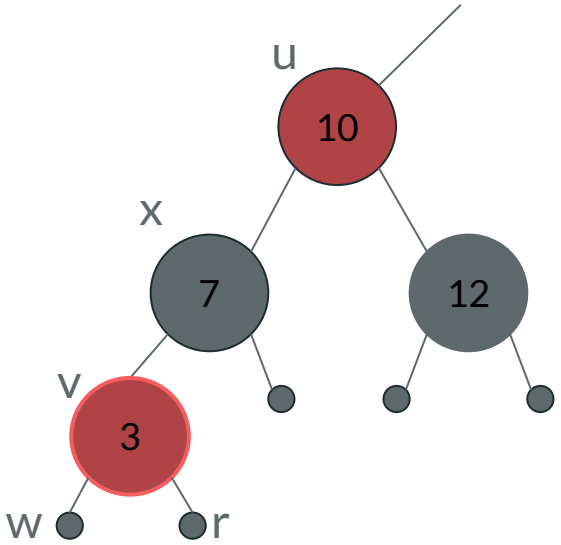
Árboles Rojo-Negro

- Borrado en un ARN
  - Se borra como en un ABB
  - Aplicable solo si el elemento a borrar tiene un hijo nulo
  - Si no tiene hijo nulo se intercambia por su sucesor y se borra
  - Ejemplo: borrar el valor 12



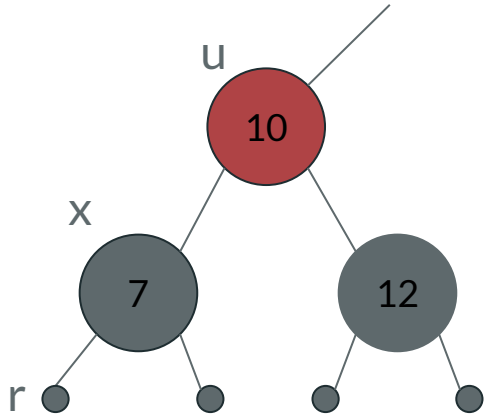
Árboles Rojo-Negro

- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.



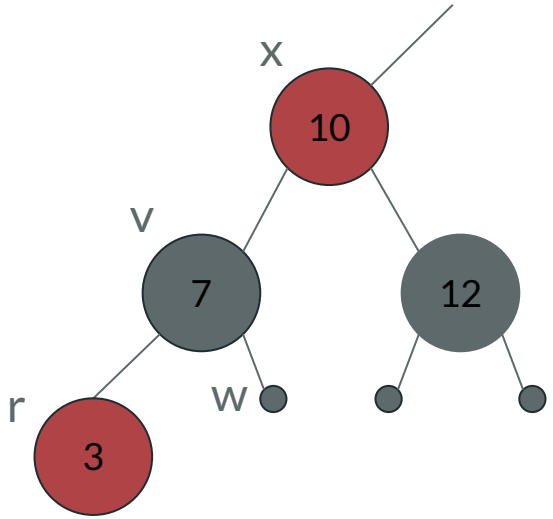
Árboles Rojo-Negro

- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.



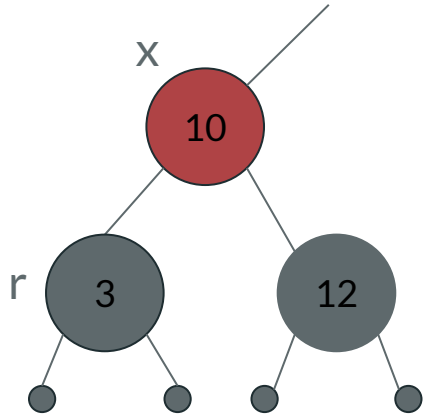
Árboles Rojo-Negro

- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.



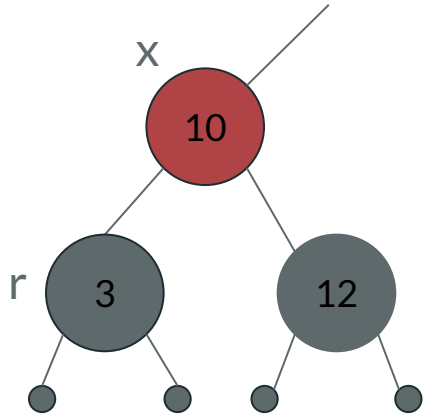
Árboles Rojo-Negro

- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.



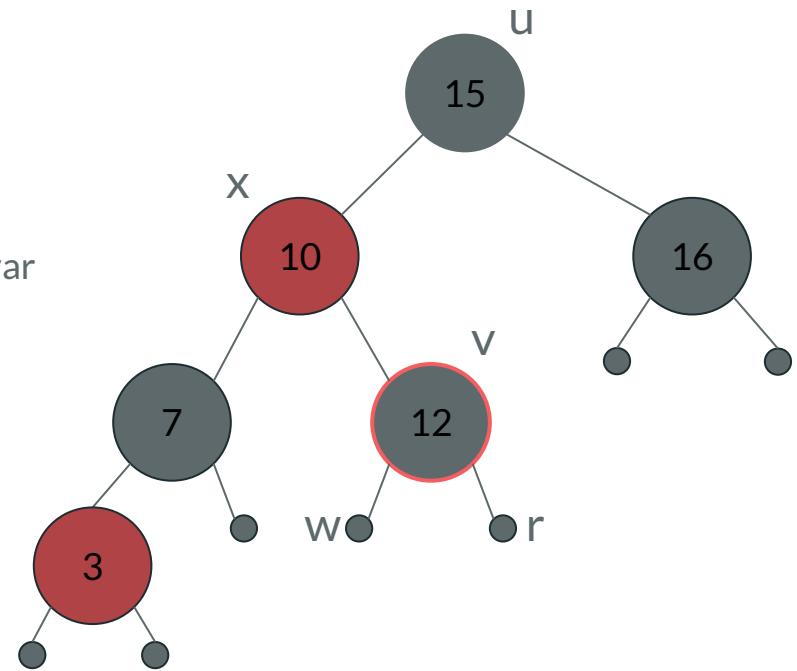
Árboles Rojo-Negro

- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.
  - Si v y r eran ambos negros, para preservar la profundidad negra, hay que asignar a “r” un doble negro.



Árboles Rojo-Negro

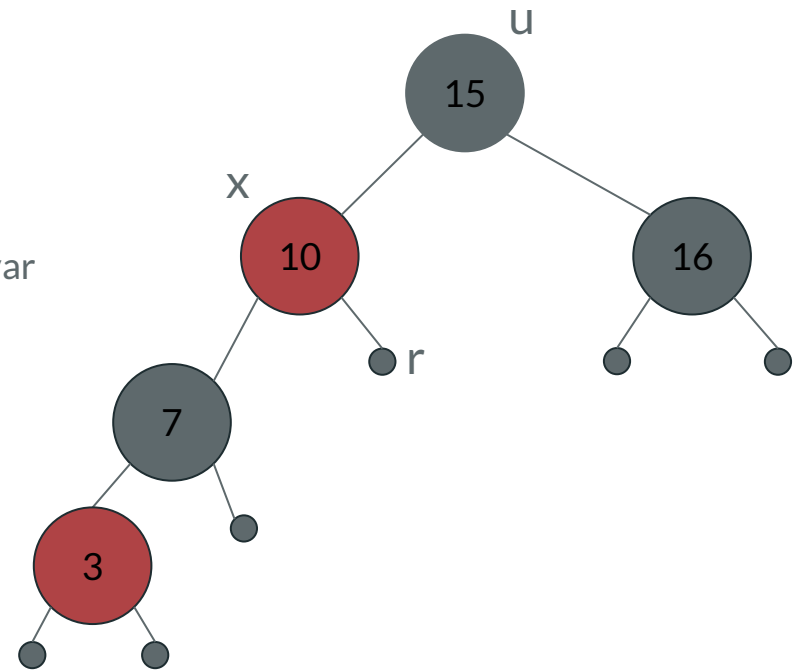
- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.
  - Si v y r eran ambos negros, para preservar la profundidad negra, hay que asignar a “r” un doble negro.
  - Ejemplo: borrar el valor 12



Árboles Rojo-Negro

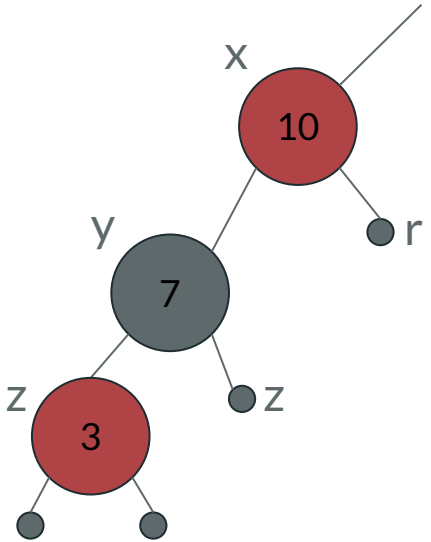
- Borrado en un ARN
  - Eliminar nodo v y su hijo nulo w.
  - Hacer que r sea hijo de x.
  - Si v o r eran rojos, r se colorea de negro.
  - Si v y r eran ambos negros, para preservar la profundidad negra, hay que asignar a “r” un doble negro.

  - Ejemplo: borrar el valor 12



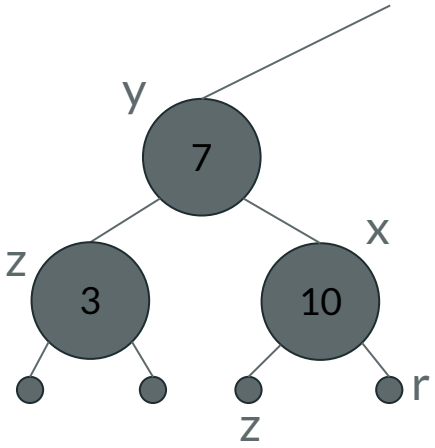
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 1: el hermano de r (y) es negro y uno de sus sobrinos(z) es rojo -> reestructuración
    - Caso 2: el hermano de r (y) es negro y sus sobrinos negros(z) - recoloración
    - Caso 3: el hermano de r(y) es rojo -> ajuste



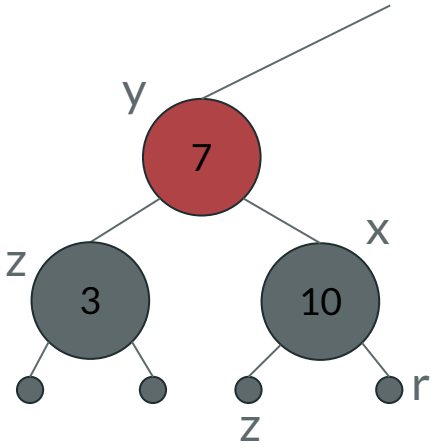
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 1: el hermano de r (y) es negro y uno de sus sobrinos(z) es rojo -> reestructuración
  - Para ello:
    - Reestructuración
    - Colorear menor y mayor de negro



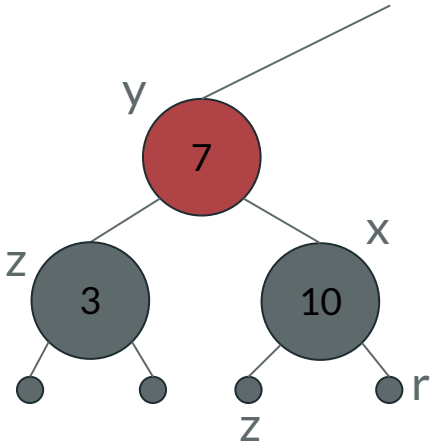
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 1: el hermano de r (y) es negro y uno de sus sobrinos(z) es rojo -> reestructuración
  - Para ello:
    - Reestructuración
    - Colorear menor y mayor de negro
    - El nodo medio queda con el color anterior de X



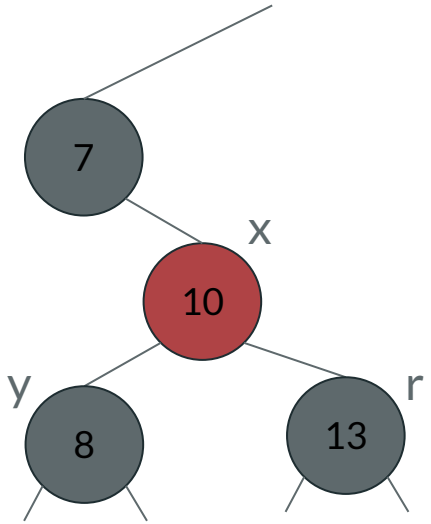
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 1: el hermano de r (y) es negro y uno de sus sobrinos(z) es rojo -> reestructuración
  - Para ello:
    - Reestructuración
    - Colorear menor y mayor de negro
    - El nodo medio queda con el color anterior de X
    - Colorear r de negro



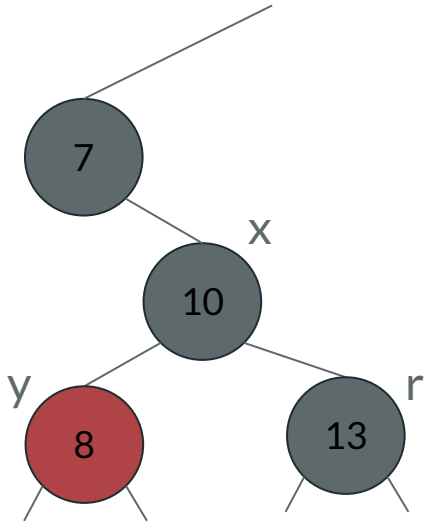
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 2: el hermano de r (y) es negro y sus sobrinos negros(z) - recoloración
  - Para ello:
    - Recoloreado de r como negro e y como rojo
    - Si x es rojo, se pasa a negro



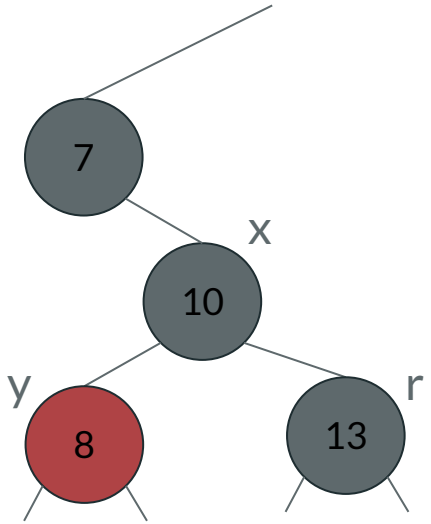
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 2: el hermano de r (y) es negro y sus sobrinos negros(z) - recoloración
  - Para ello:
    - Recoloreado de r como negro e y como rojo
    - Si x es rojo, se pasa a negro



Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 2: el hermano de r (y) es negro y sus sobrinos negros(z) - recoloración



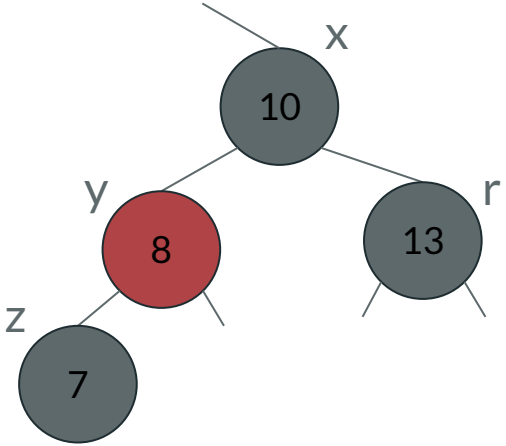
  - Para ello:
    - Recoloreado de r como negro e y como rojo
    - Si x es rojo, se pasa a negro

Si x no era rojo, x vuelve a ser doble negro. Ver caso (1, 2 o 3)

Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 3: el hermano de r(y) es rojo -> ajuste

z se elige en función de la posición de y (rotación simple).

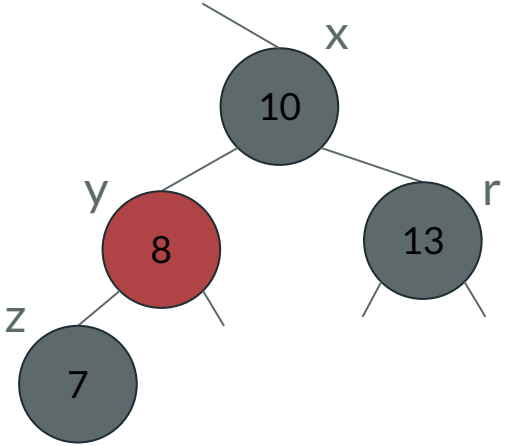


Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 3: el hermano de r(y) es rojo -> ajuste

z se elige en función de la posición de y (rotación simple).

  - Para ello:
    - Reestructuración

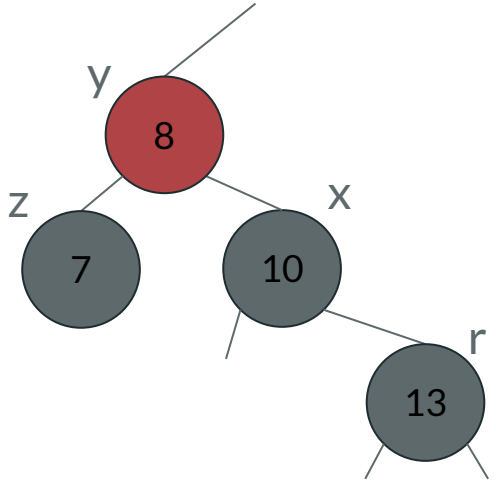


Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 3: el hermano de r(y) es rojo -> ajuste

z se elige en función de la posición de y (rotación simple).

  - Para ello:
    - Reestructuración

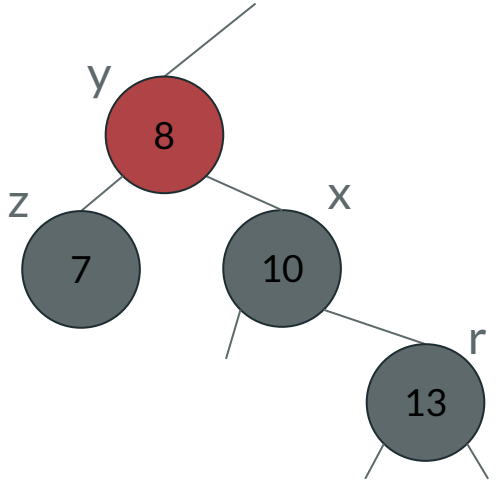


Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 3: el hermano de r(y) es rojo -> ajuste

z se elige en función de la posición de y (rotación simple).

  - Para ello:
    - Reestructuración
    - Colorear y de negro y x de rojo



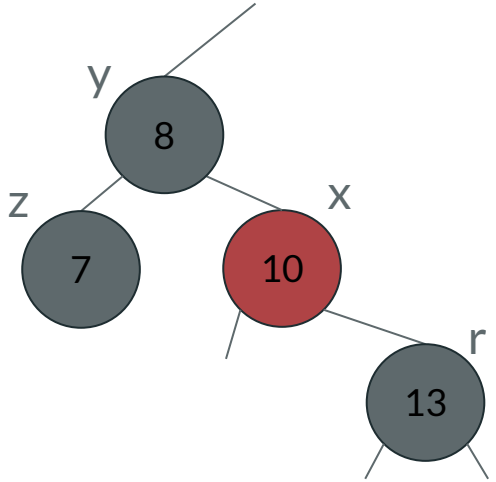
Árboles Rojo-Negro

- Borrado en un ARN
  - 3 posibles situaciones:
    - Caso 3: el hermano de r(y) es rojo -> ajuste

z se elige en función de la posición de y (rotación simple).

  - Para ello:
    - Reestructuración
    - Colorear y de negro y x de rojo

El doble negro no desaparece. Caso 1 o 2.



Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

Moodle de la asignatura

Guía docente