




Descripción de la asignatura

- Tema 1: Introducción  Bloque 1
- Tema 2: Árboles generales
- Tema 3: Mapas y Diccionarios
- **Tema 4: Mapas y Diccionarios ordenados**  Bloque 2
- Tema 5: Grafos
- Tema 6: EEDD en Memoria Secundaria  Bloque 3

V 1.1

Tema II.4

Mapas y Diccionarios ordenados

jose.velez@urjc.es
angel.sanchez@urjc.es
mariateresa.gonzalezdelena@urjc.es

abraham.duarte@urjc.es
raul.cabido@urjc.es

Resumen

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

Objetivos

- Descripción de las estructuras de datos mapa y diccionario ordenado
- Aproximaciones basadas en árboles binarios de búsqueda (ABBs)
- Diseño e implementación de los TADs ABBs y diccionario, manteniendo la encapsulación y modularización

Árbol Binario de Búsqueda

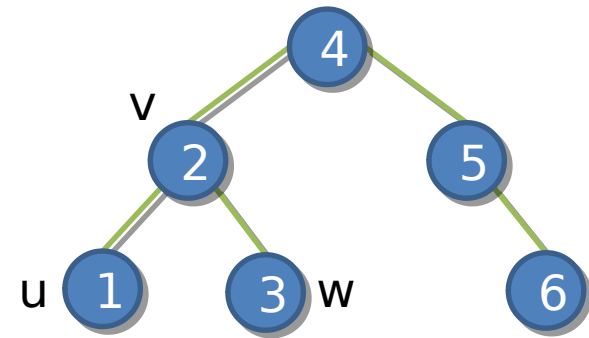
Árbol binario de búsqueda

Un árbol Binario de Búsqueda (ABB) es un **árbol binario** en el que todos los nodos satisfacen las siguiente propiedad:

- 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

$$value(u) < value(v) \leq value(w)$$

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

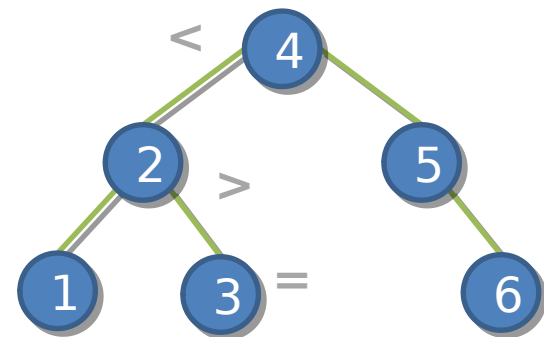


Búsqueda en un ABB

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
- Si la clave no está devuelve *null*

```
algorithm 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 (node)
  else // value > node.value
    return TreeSearch(value, node.right)
```



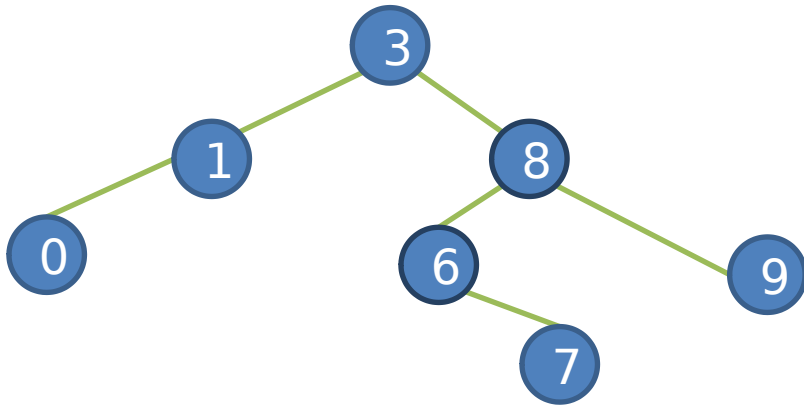
TreeSearch(3, root)

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

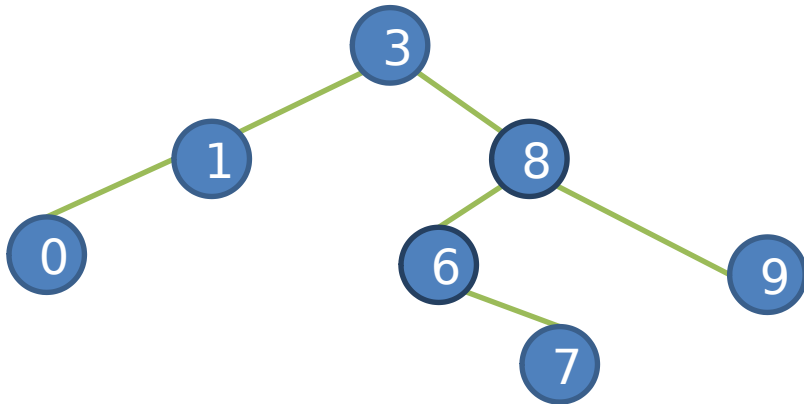
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```



Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)




auxParent

Algorithm 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

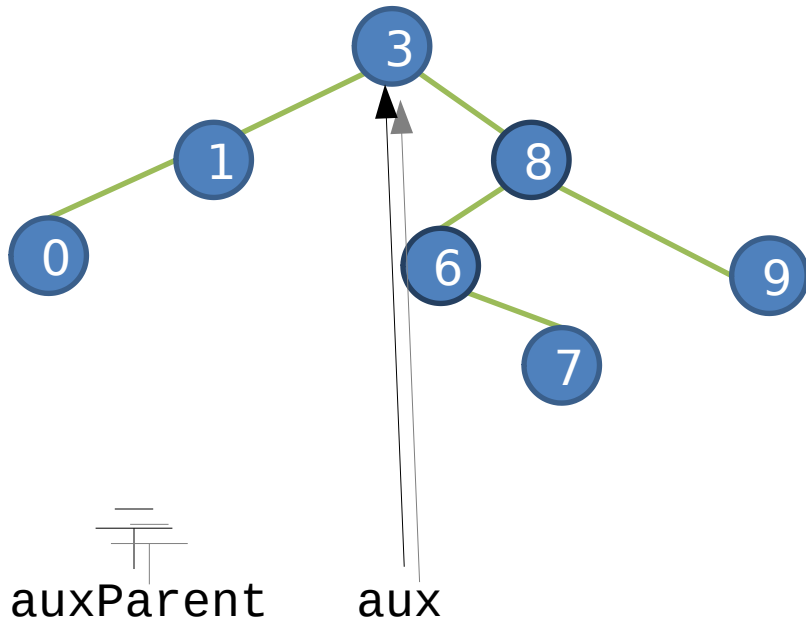
 newNode.parent = auxParent

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

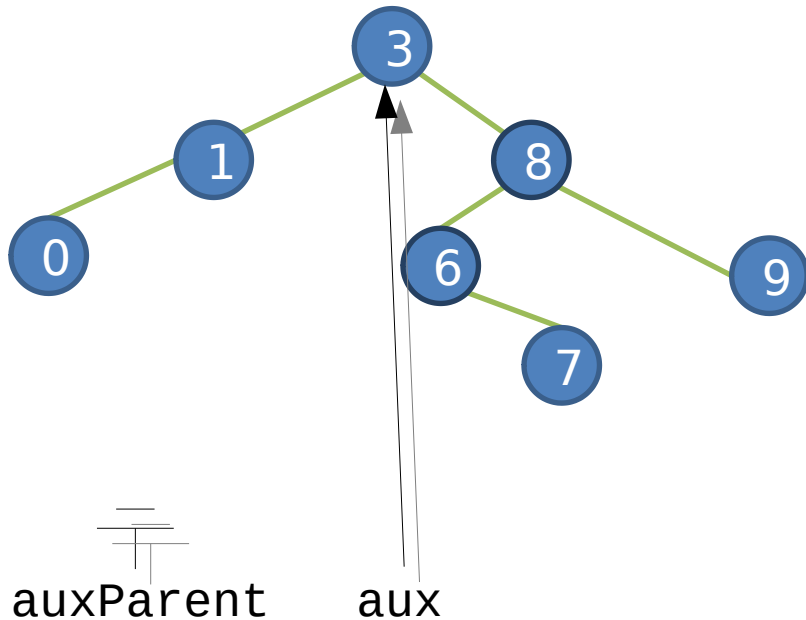
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

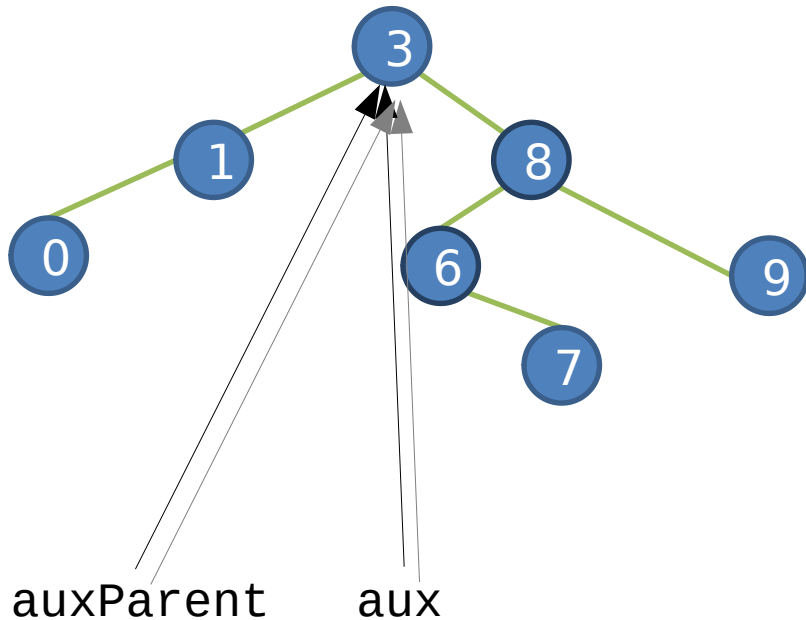
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

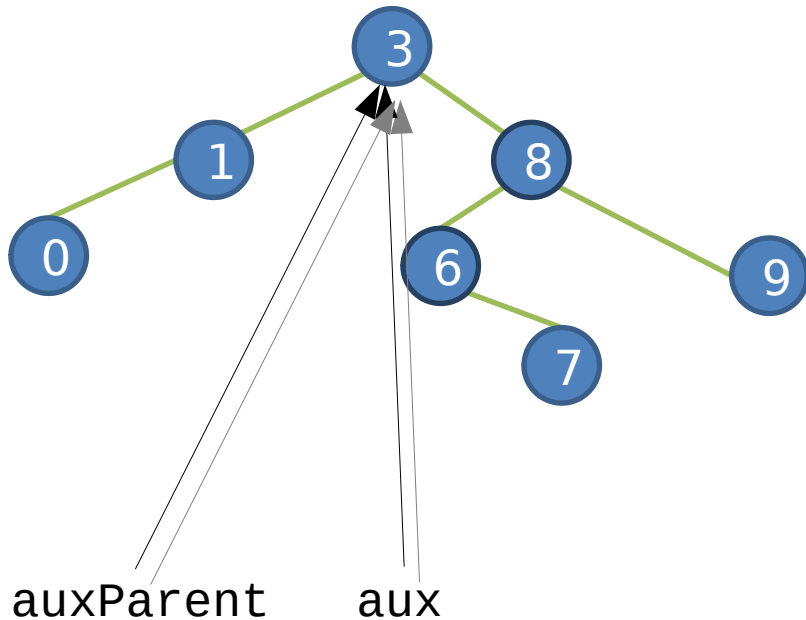
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

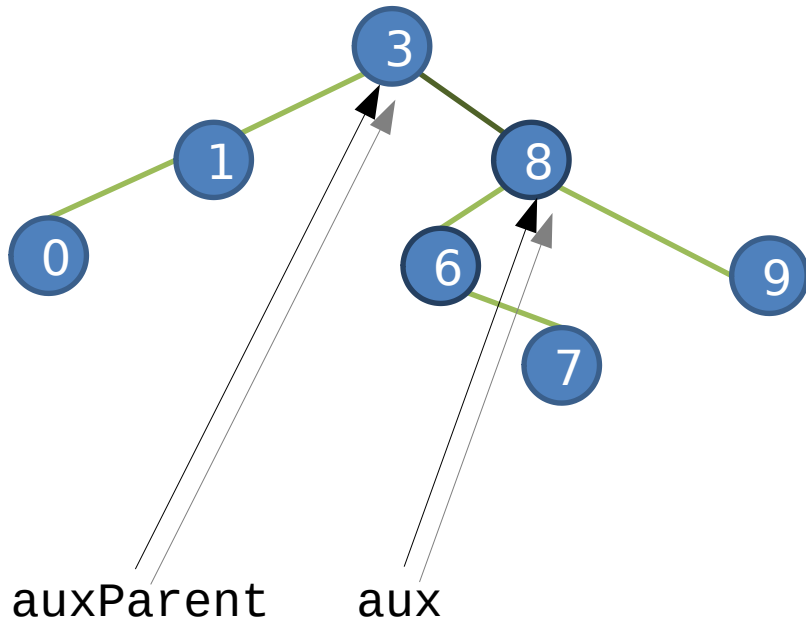
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

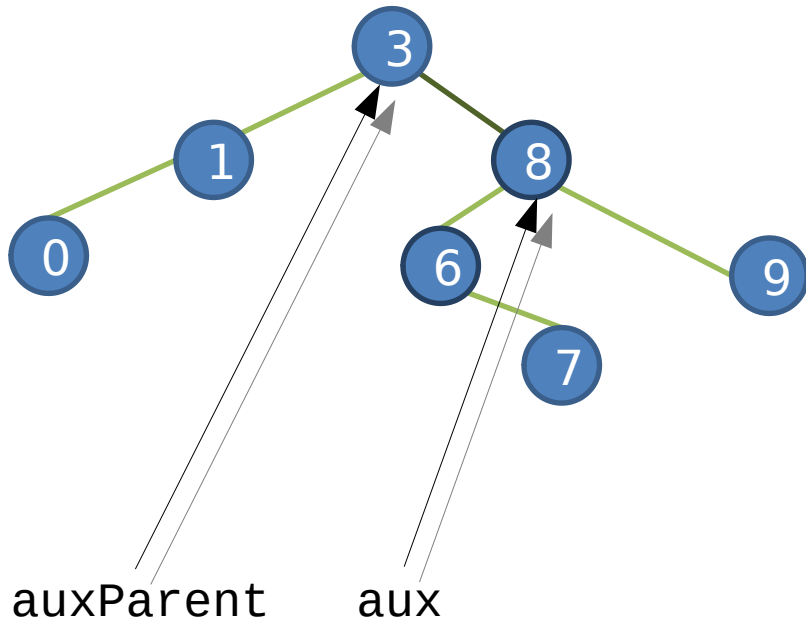
```
Node newNode(value)
if auxParent == null
    root = newNode
else if value < auxParent.value
    auxParent.left = newNode
else
    auxParent.right = newNode
newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

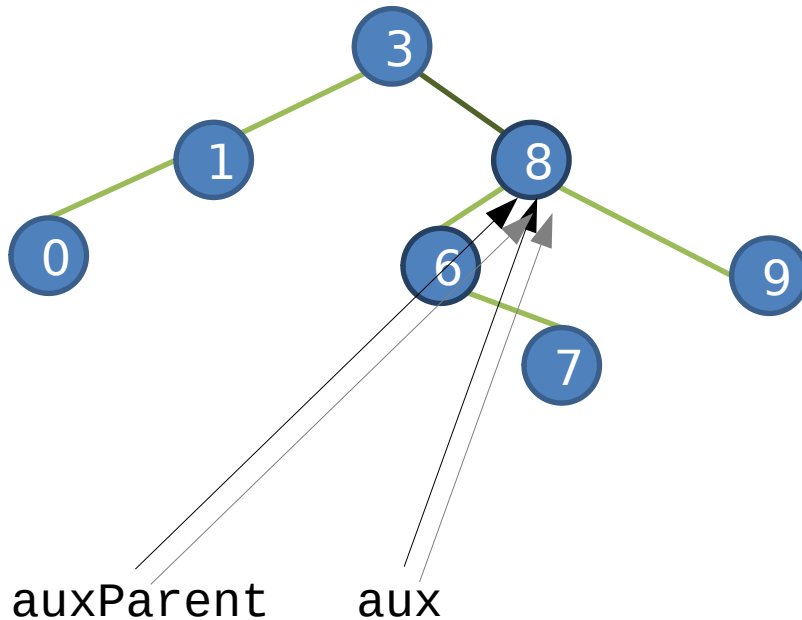
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

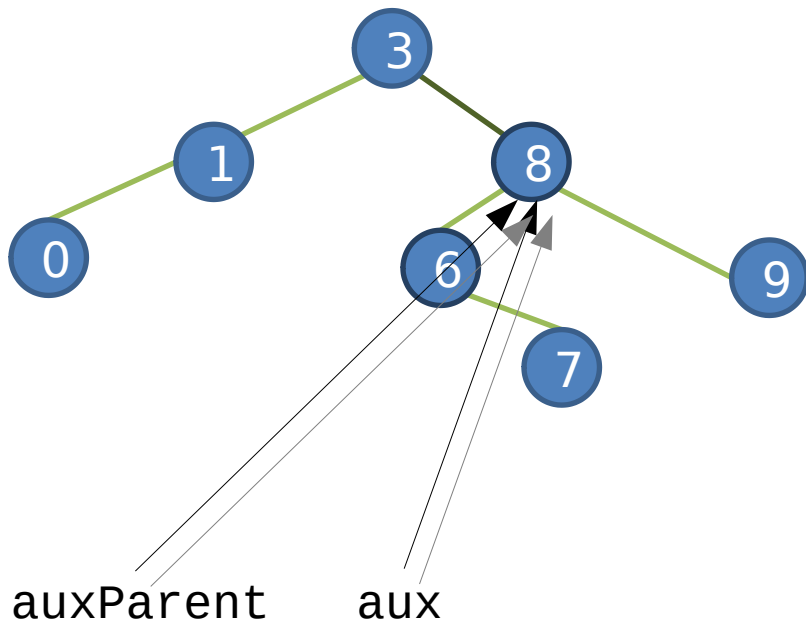
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```


Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

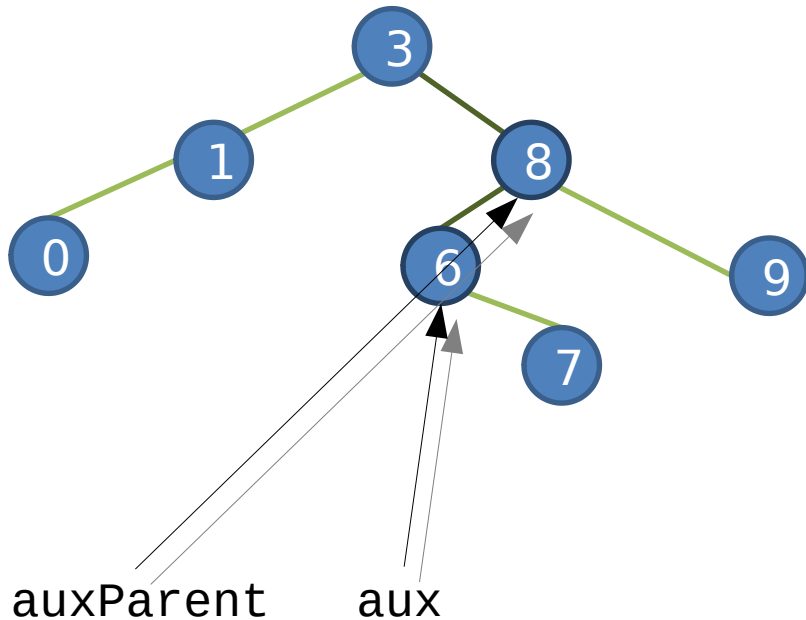
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

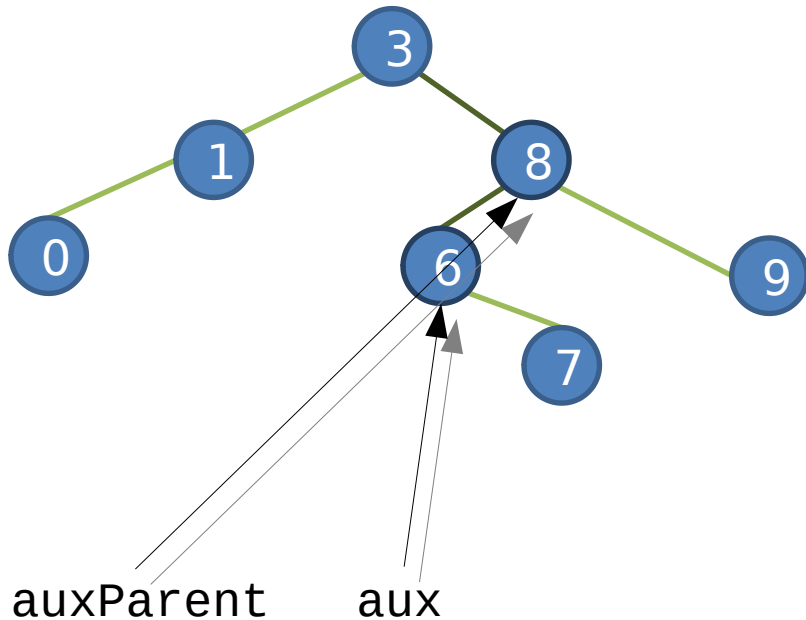
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

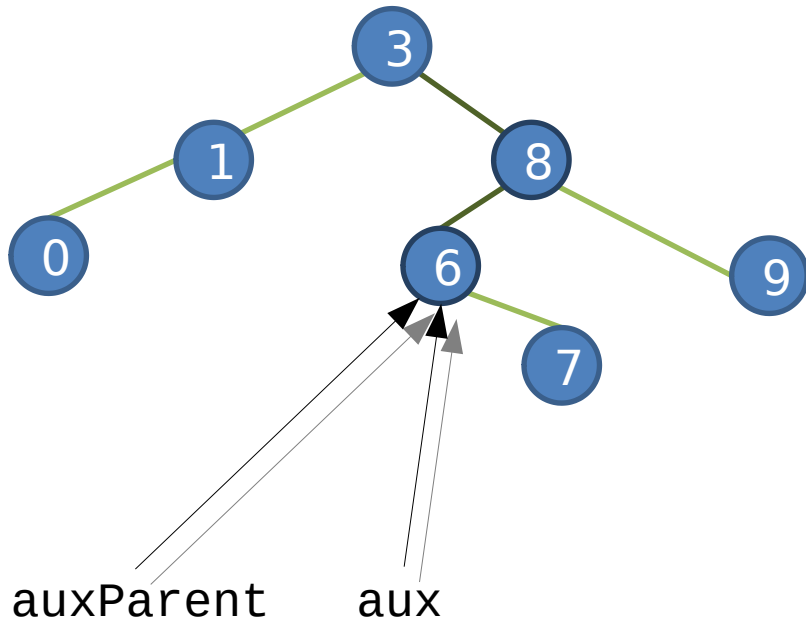
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

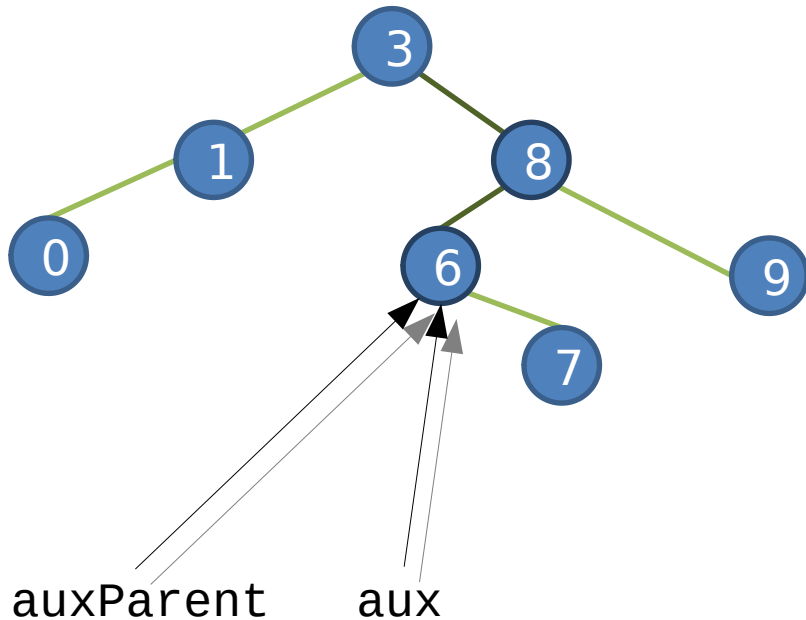
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

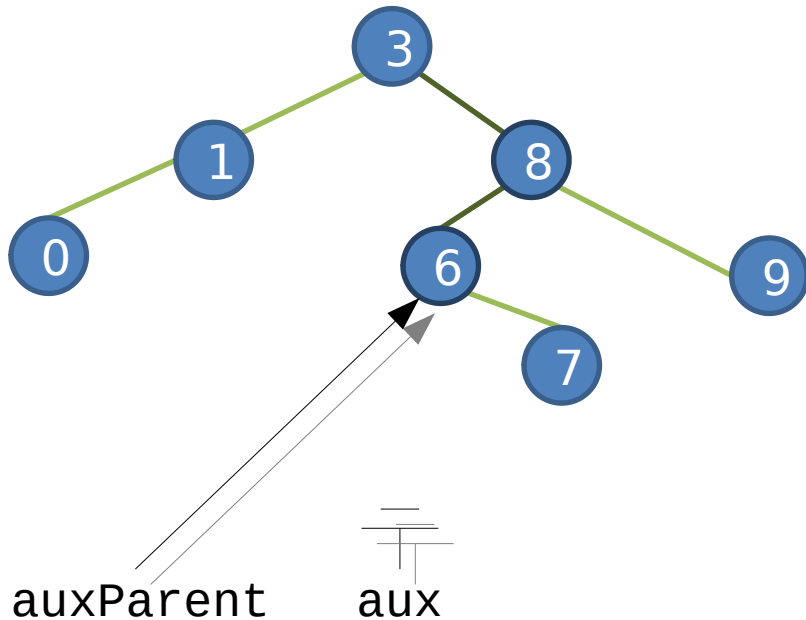
Node newNode(value)
if auxParent == null
    root = newNode
else if value < auxParent.value
    auxParent.left = newNode
else
    auxParent.right = newNode
newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

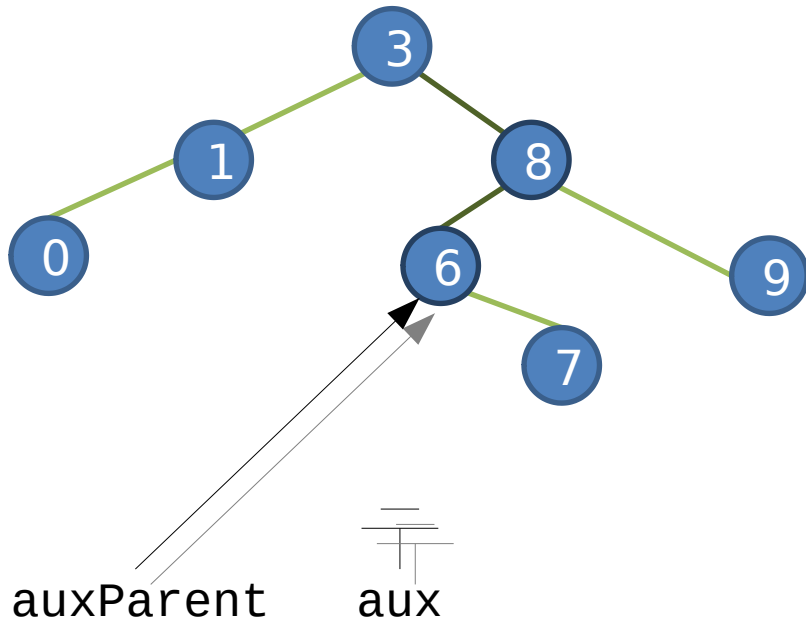
Node newNode(value)
if auxParent == null
    root = newNode
else if value < auxParent.value
    auxParent.left = newNode
else
    auxParent.right = newNode
newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



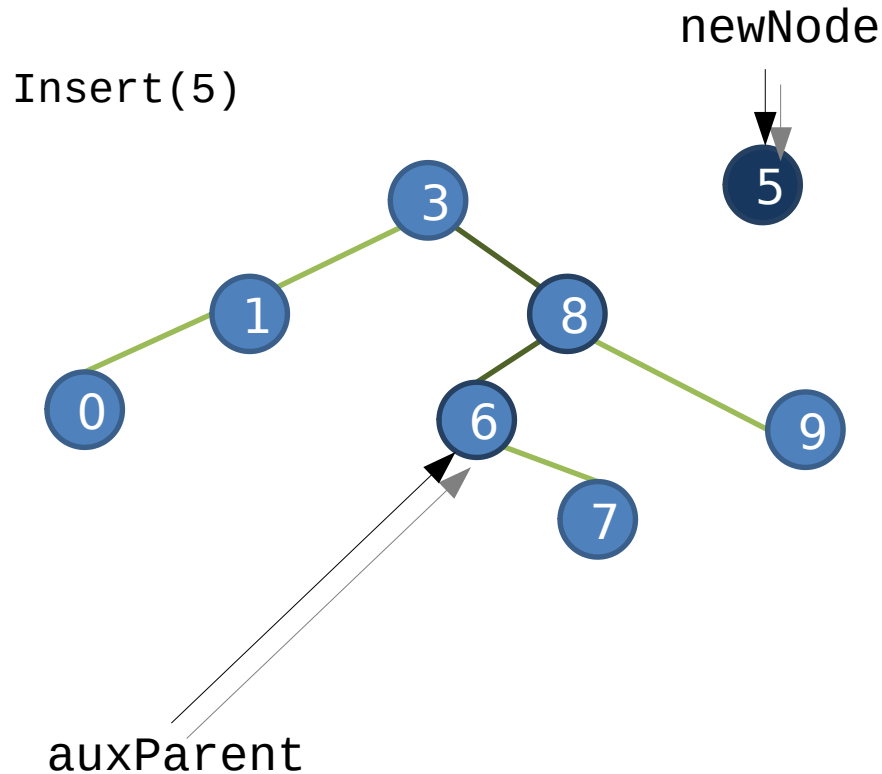
```
Algorithm Insert(value)
  Node auxParent = null
  Node aux = root
  while aux != null
    auxParent = aux
    if value < aux.value
      aux = aux.left
    else
      aux = aux.right

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

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura



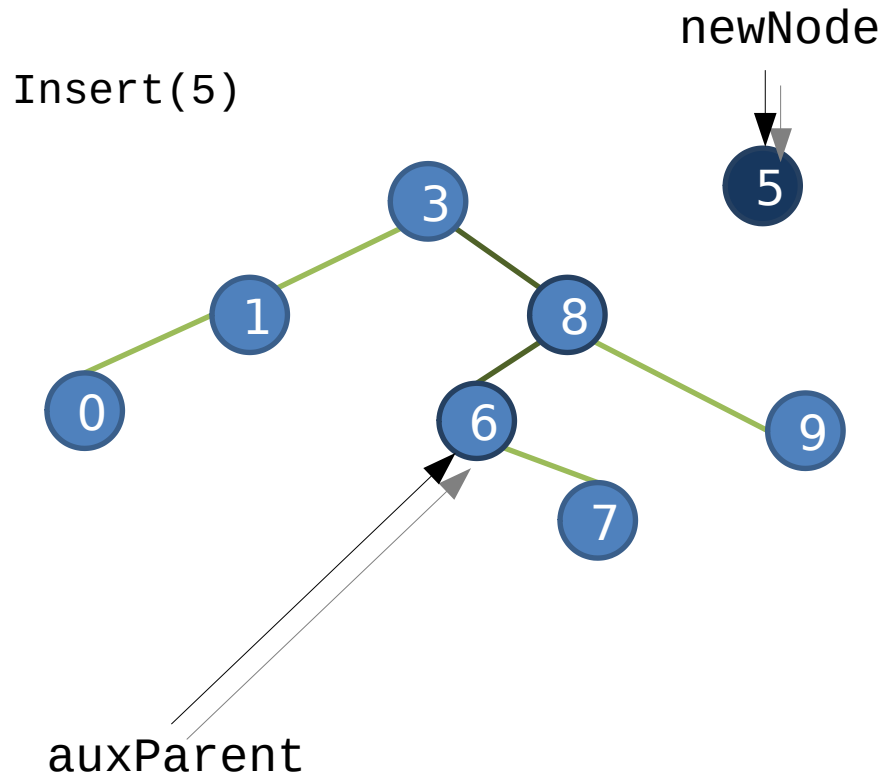
```
Algorithm Insert(value)
Node auxParent = null
Node aux = root
while aux != null
    auxParent = aux
    if value < aux.value
        aux = aux.left
    else
        aux = aux.right
```

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


Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura



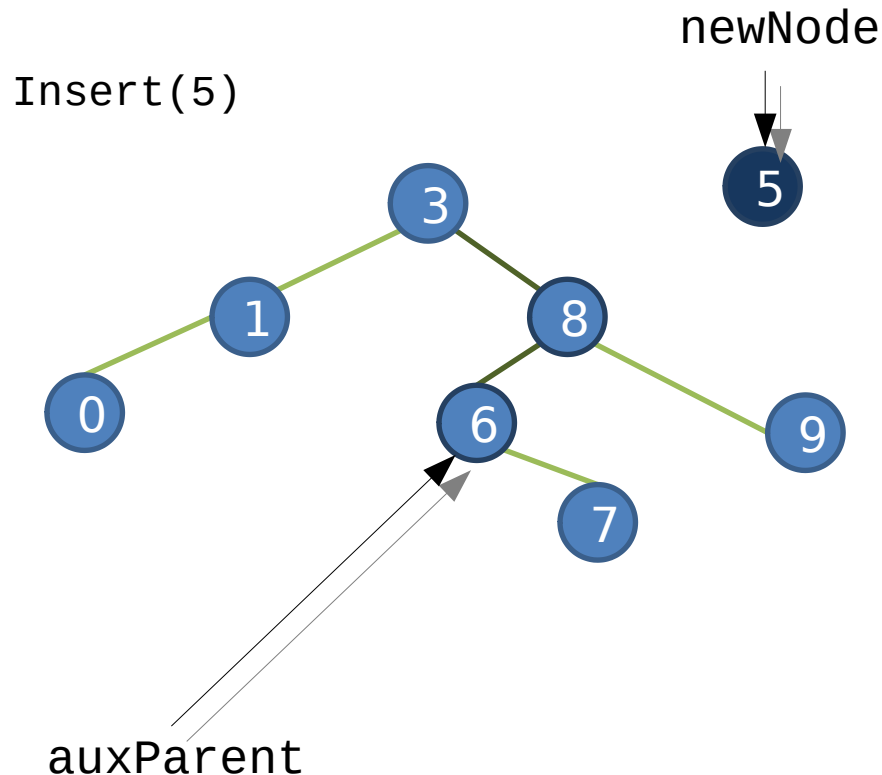
```
Algorithm Insert(value)
Node auxParent = null
Node aux = root
while aux != null
    auxParent = aux
    if value < aux.value
        aux = aux.left
    else
        aux = aux.right

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

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura



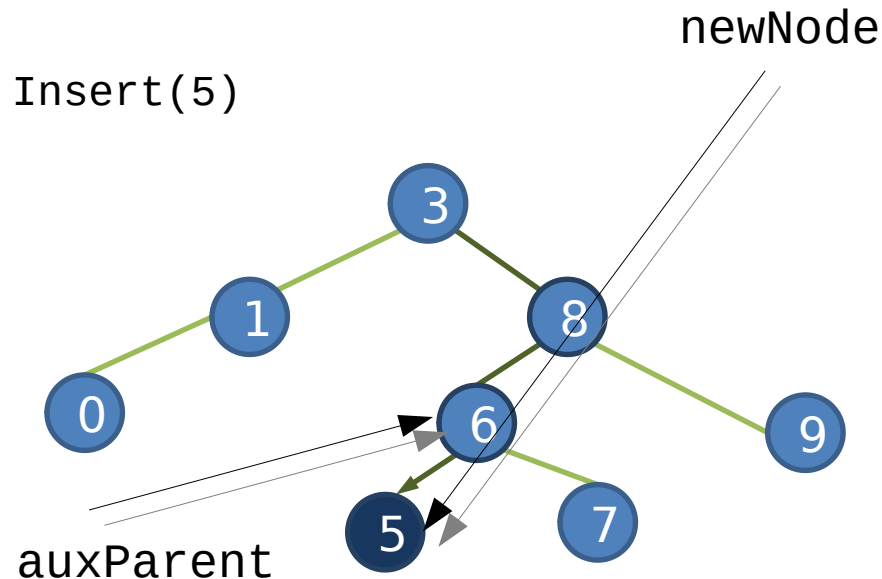
```
Algorithm Insert(value)
Node auxParent = null
Node aux = root
while aux != null
    auxParent = aux
    if value < aux.value
        aux = aux.left
    else
        aux = aux.right

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

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura



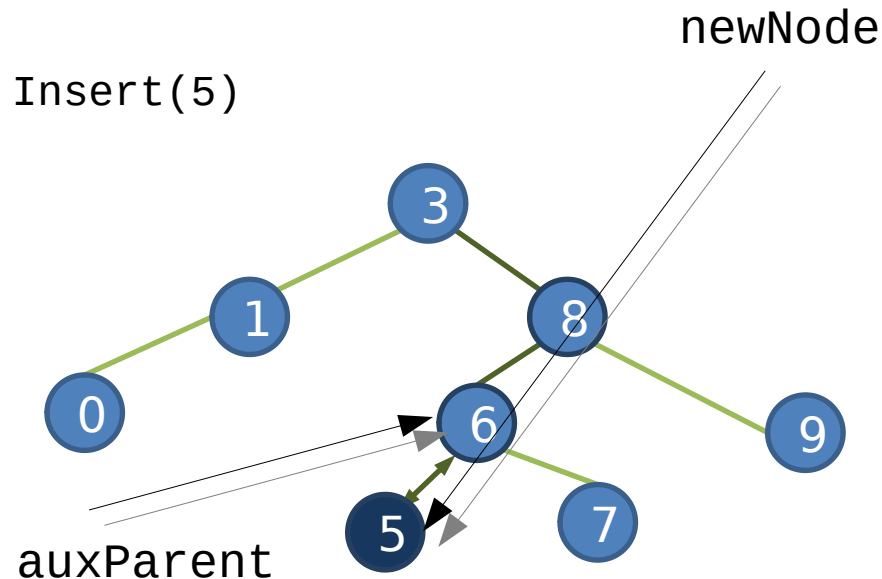
```
Algorithm Insert(value)
Node auxParent = null
Node aux = root
while aux != null
    auxParent = aux
    if value < aux.value
        aux = aux.left
    else
        aux = aux.right

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

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura



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

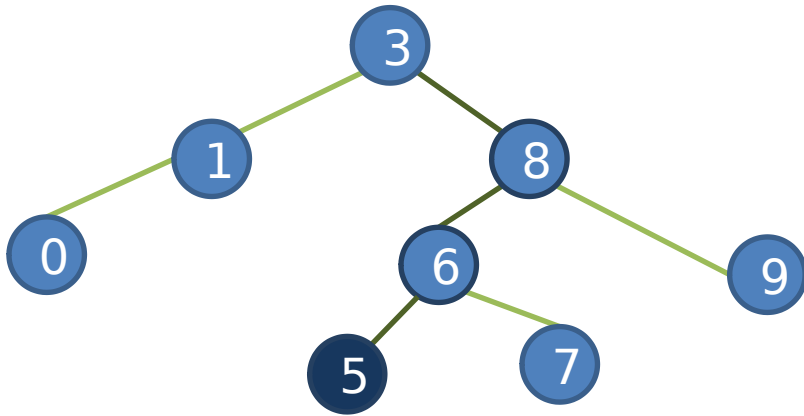
Node newNode(value)
if auxParent == null
    root = newNode
else if value < auxParent.value
    auxParent.left = newNode
else
    auxParent.right = newNode
newNode.parent = auxParent
```

Inserción en un ABB

El algoritmo para insertar la clave value en un ABB se puede expresar mediante un algoritmo iterativo

- La **complejidad** del algoritmo es $O(h)$, donde h es la altura

Insert(5)



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

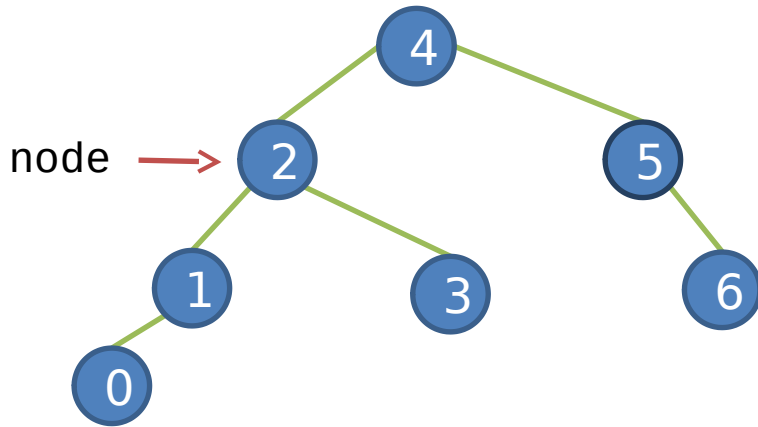
  Node newNode(value)
  if auxParent == null
    root = newNode
  else if value < auxParent.value
    auxParent.left = newNode
  else
    auxParent.right = newNode
  newNode.parent = auxParent
```

Mínimo en un ABB

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 las claves.

Mínimo(node)



```
Algorithm Minimum(node)
  while not node.left == null
    node = node.left
  return node
```

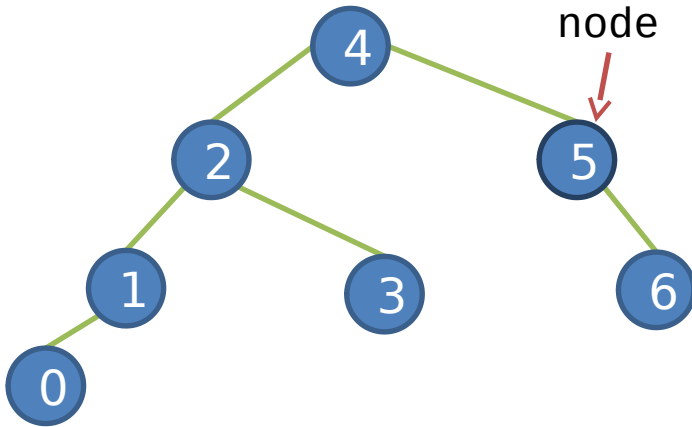
En este caso el mínimo corresponde al nodo con valor 0

Mínimo en un ABB

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 las claves.

Mínimo(node)



```
Algorithm Minimum(node)
  while not node.left == null
    node = node.left
  return node
```

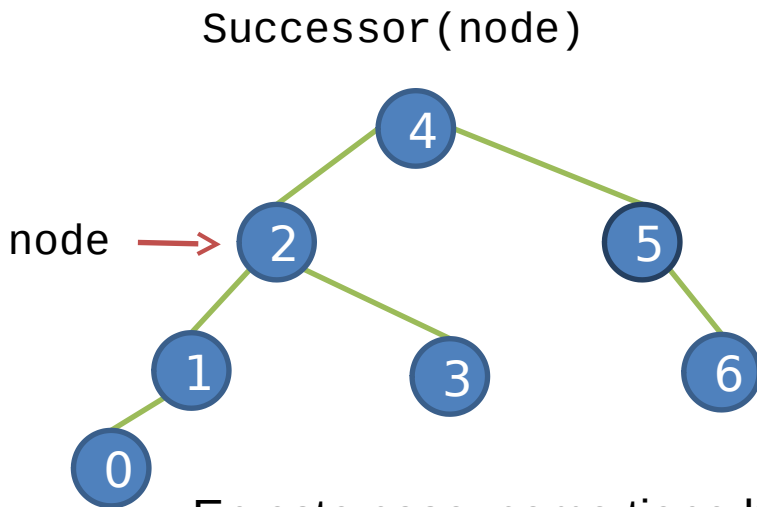
En este caso, como el nodo no tiene hijo izquierdo, el mínimo corresponde con el mismo

Sucesor en un ABB

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

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

- Obsérvese que, gracias a la estructura del árbol binario de búsqueda, no es necesario comparar las claves.
- La complejidad del algoritmo de cálculo del sucesor es $O(h)$, donde h es la altura.



En este caso, como tiene hijo derecho, el sucesor es el mínimo del subárbol derecho

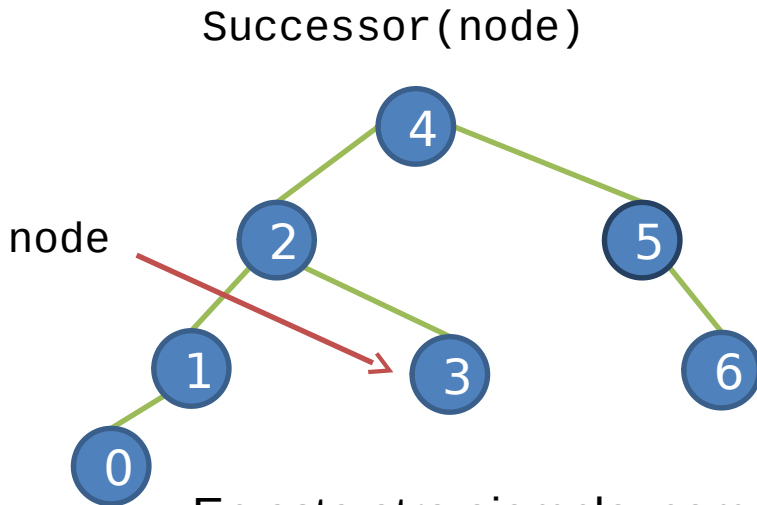
```
Algorithm Successor(node)
  if not node.right==null
    return Minimum(node.right)
  Node aux = node.parent
  while aux!=null and node==aux.right
    node = aux
    aux = aux.parent
  return aux
```


Sucesor en un ABB

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

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

- Obsérvese que, gracias a la estructura del árbol binario de búsqueda, no es necesario comparar las claves.
- La complejidad del algoritmo de cálculo del sucesor es $O(h)$, donde h es la altura.



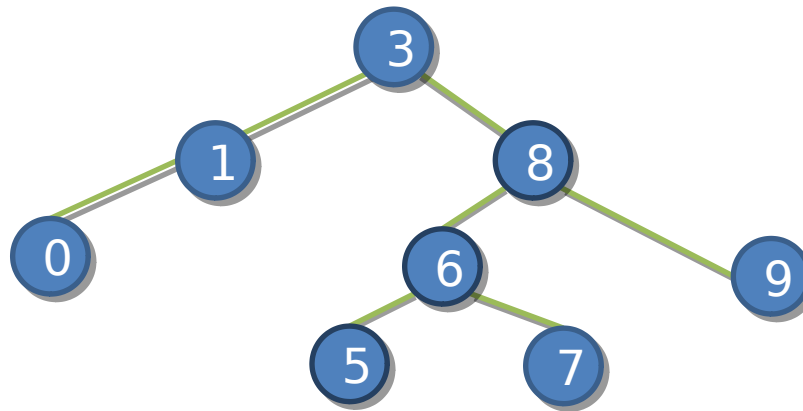
```
Algorithm Successor(node)
  if not node.right==null
    return Minimum(node.right)
  Node aux = node.parent
  while aux!=null and node==aux.right
    node = aux
    aux = aux.parent
  return aux
```

En este otro ejemplo, como el nodo no tiene hijo derecho, su sucesor es el primer ancestro mayor que él, el nodo 4.

Borrado en un ABB

El borrado tiene 3 casos:

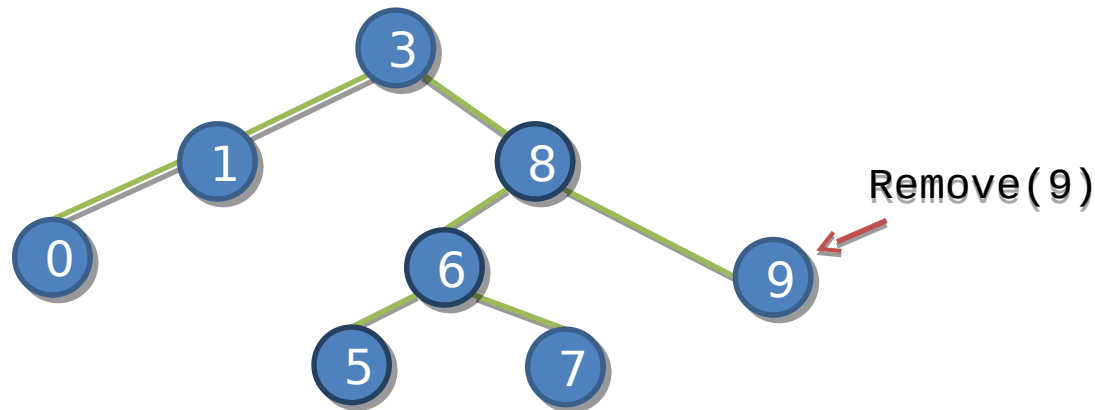
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

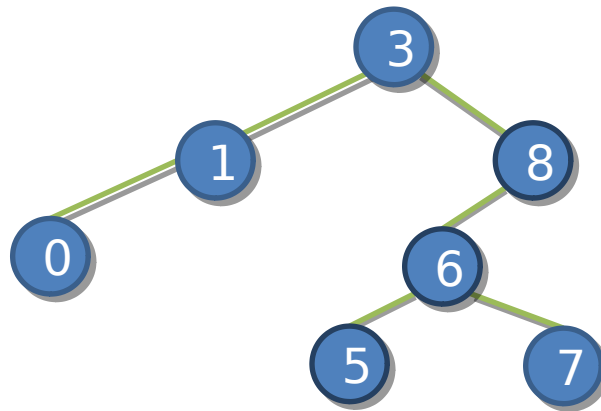
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

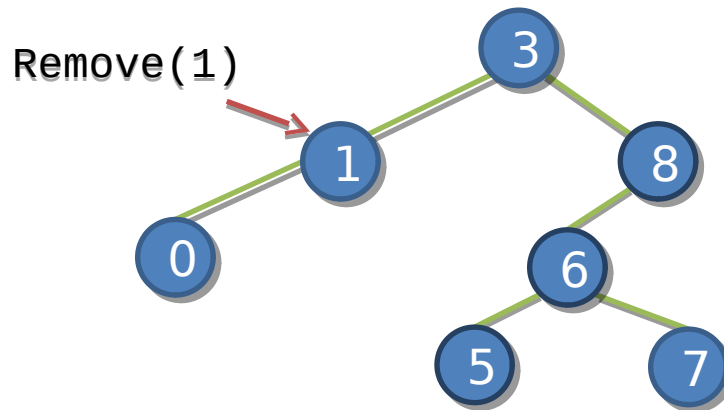
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

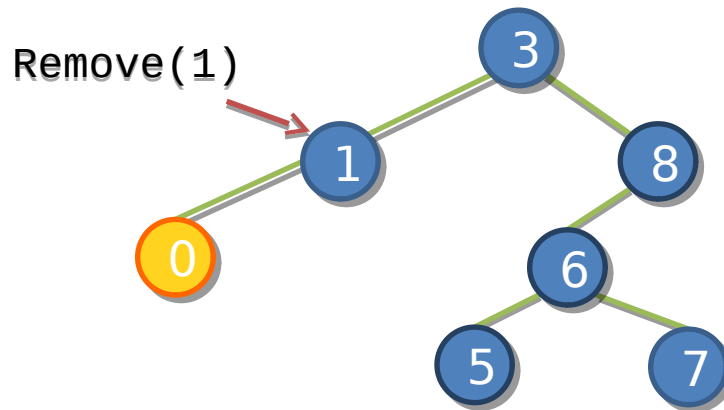
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

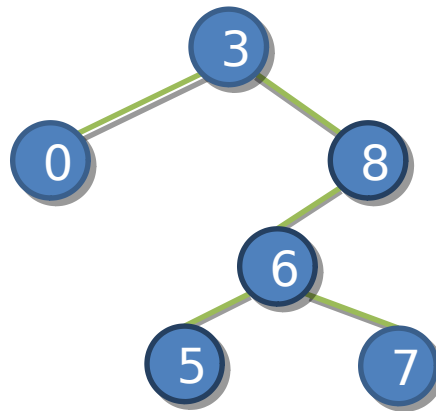
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su **hijo**
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

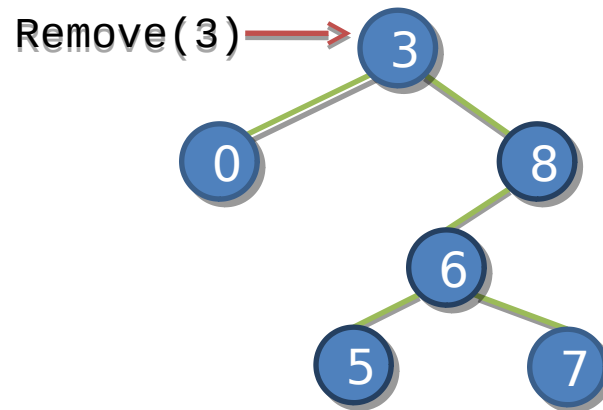
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

El borrado tiene 3 casos:

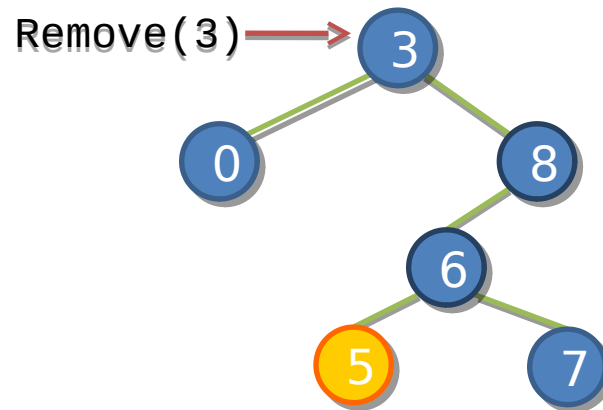
- El nodo a borrar es una hoja
 - Se elimina el nodo directamente
- El nodo a borrar solo tiene un hijo
 - Se reemplaza el nodo por su hijo
- El nodo a borrar tiene dos hijos
 - Se pone al nodo el valor del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

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 del **sucesor** (el nodo más a la izquierda del subárbol derecho) y se elimina este último



Borrado en un ABB

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 del sucesor (el nodo más a la izquierda del subárbol derecho) y se elimina este último

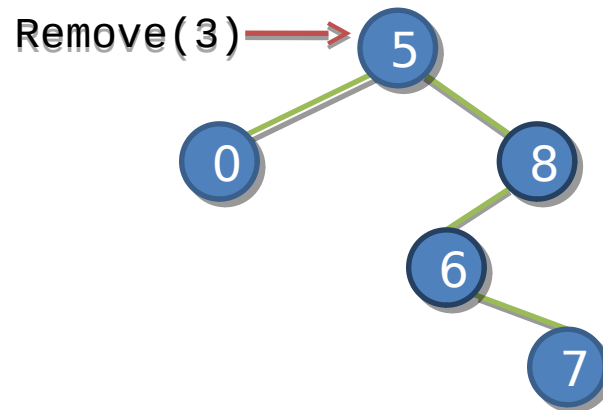
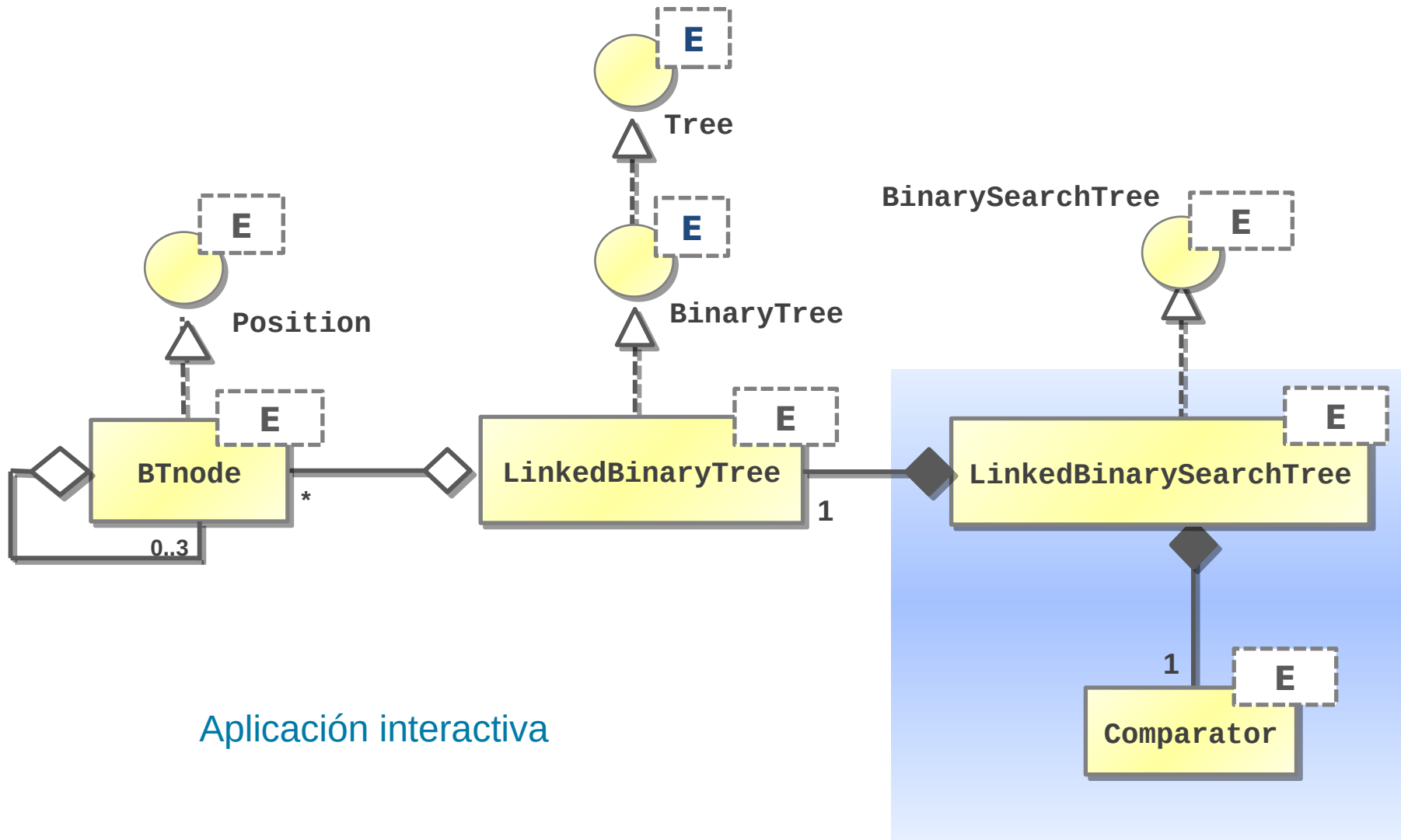


ABB. Implementación Java



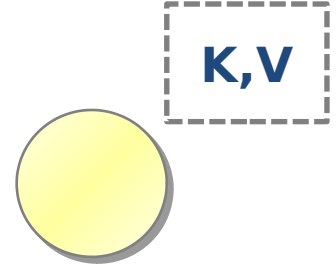
TAD Diccionario ordenado

TAD Diccionario ordenado

- Un **diccionario ordenado** almacena información que puede ser localizada eficientemente mediante la **comparación** de claves
 - Almacena colección de pares clave-valor, denominadas **entradas**
- En el Mapa no están permitidas múltiples entradas con la misma clave pero en el Diccionario sí.
- Entradas ordenadas en base a un criterio de ordenación entre sus claves
- Operaciones principales: Insertar, Buscar y Borrar

Diccionario ordenado (operaciones)

```
public interface OrderedDictionary<K, V> {  
  
    public int size();  
  
    public boolean isEmpty();  
  
    public Entry<K, V> find(K key) throws InvalidKeyException;  
  
    public Iterable<Entry<K, V>> findAll(K key)  
        throws InvalidKeyException;  
  
    public Entry<K, V> insert(K key, V value)  
        throws InvalidKeyException;  
  
    public Entry<K, V> remove(Entry<K, V> e)  
        throws InvalidEntryException;  
  
    public Iterable<Entry<K, V>> entries();  
}
```



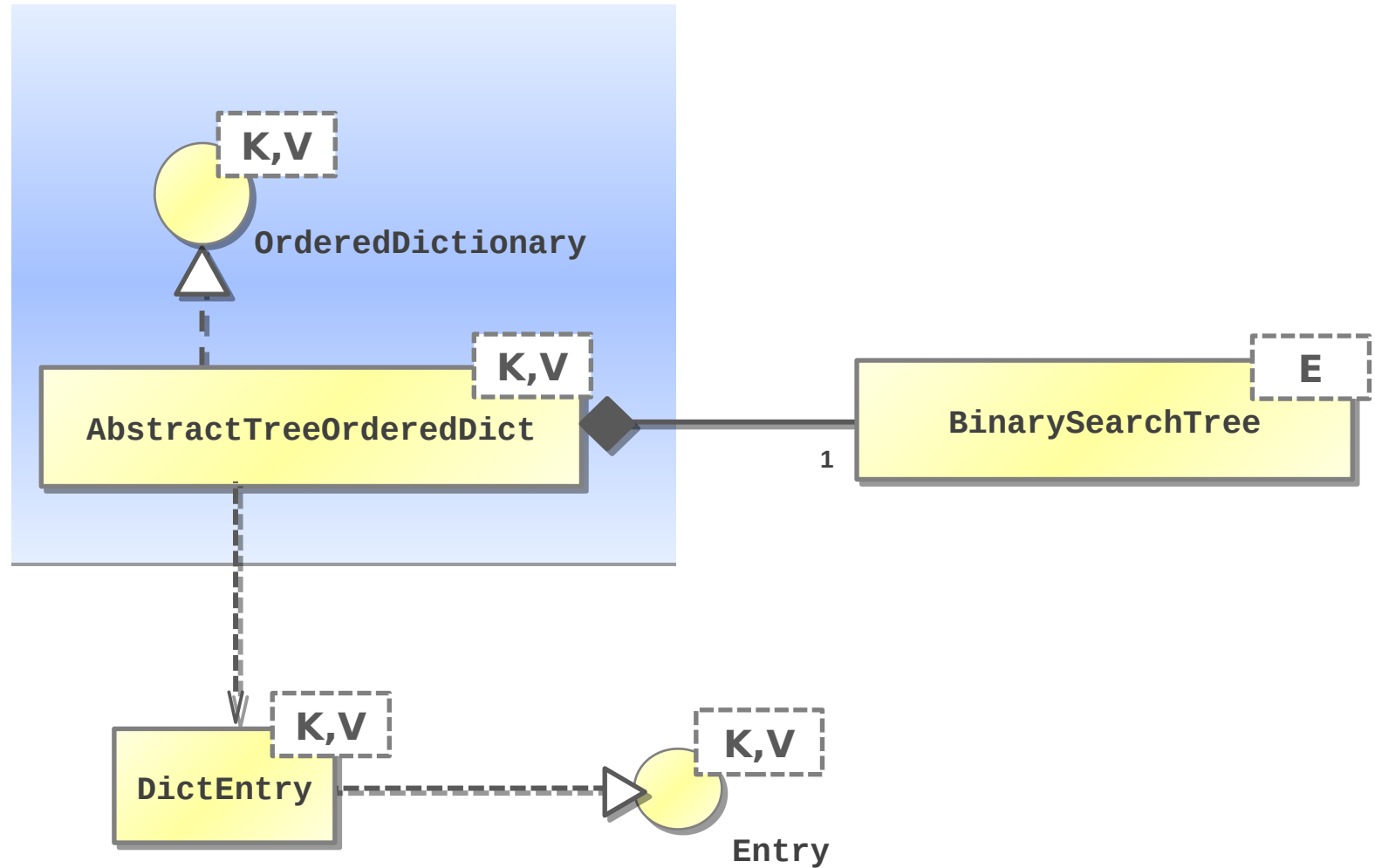
OrderedDictionary

Diccionario ordenado (operaciones)

De manera adicional, suelen considerarse también las siguientes operaciones:

- `first()`: primera entrada en el diccionario/mapa ordenado
- `last()`: última entrada en el diccionario/mapa ordenado
- `successors()`: iterador sobre una colección de entradas con claves mayores o iguales que `k` (orden ascendente)
- `predecessors()`: iterador sobre una colección de entradas con claves menores que `k` (orden descendente)
- `range()`: iterador sobre una colección de entradas con claves mayores o iguales que `k1` y menores y iguales que `k2` (orden ascendente)

Diccionario ordenado (implementación java)

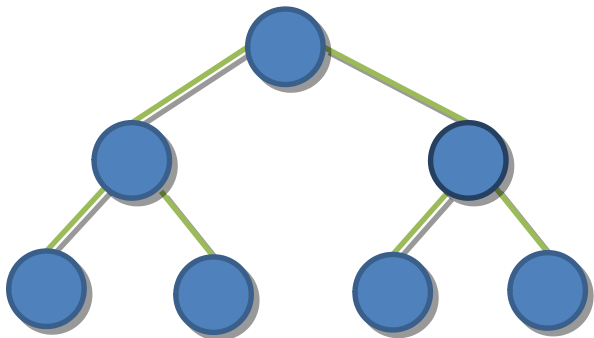


Equilibrado de ABB

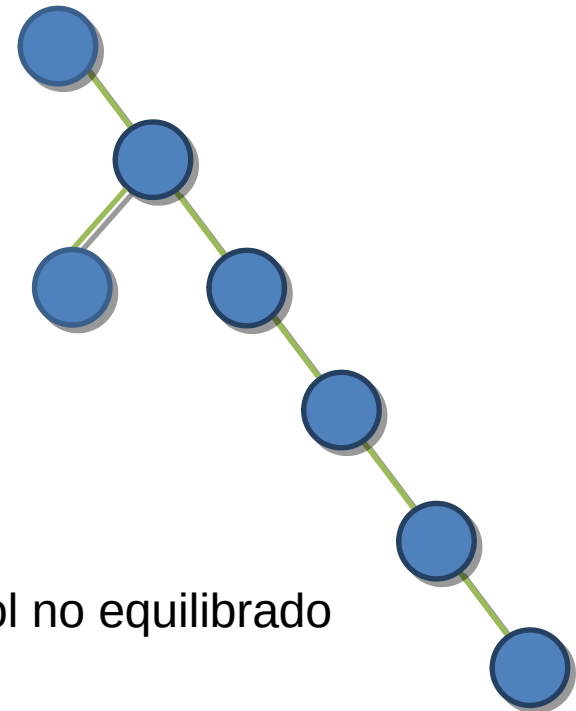
Aunque la complejidad no cambia, la velocidad de las operaciones sobre un ABB depende enormemente del equilibrio del árbol.

Los siguientes ejemplos muestran dos ABB, uno equilibrado y otro muy poco equilibrado.

En este caso, la búsqueda del máximo en el árbol equilibrado solo precisa de 2 comparaciones, mientras que en el no equilibrado precisa de 5.



Árbol equilibrado



Árbol no equilibrado

Equilibrado de ABB

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

En este tema estudiaremos dos tipos de árboles equilibrados:

- Árboles AVL (de Adelson-Velskii y Landis)
- Árboles Rojo y Negro

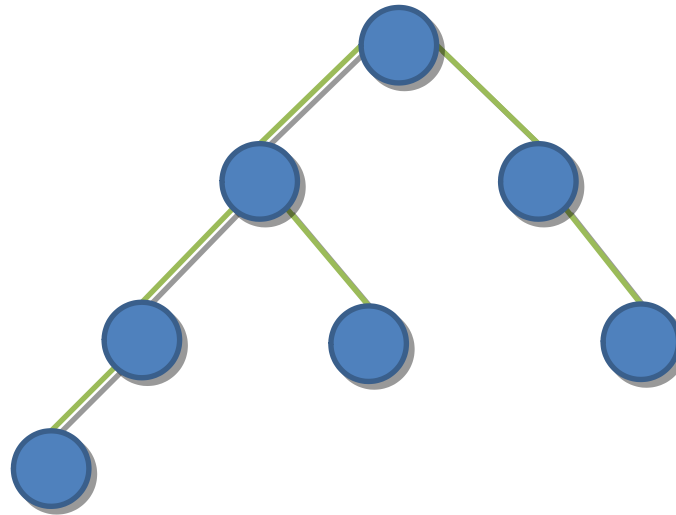
Árboles AVL

Árboles AVL

Idea intuitiva del árbol AVL



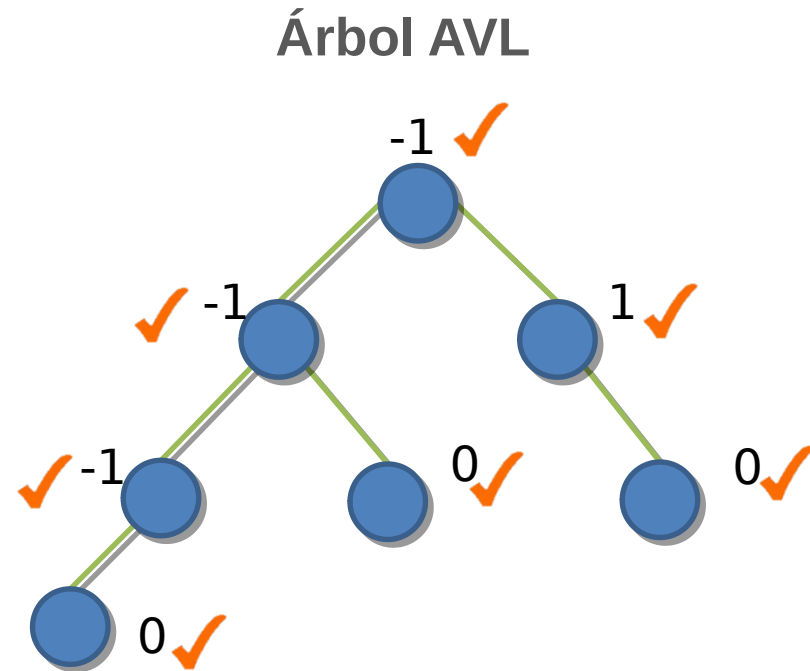
Georgy Adelson-Velsky y Evgenii Landis tuvieron la idea de equilibrar un árbol de manera que el sub-árbol de la derecha y el sub-árbol de la izquierda de cualquier nodo tuviese aproximadamente la misma altura.



Árboles AVL

Un árbol AVL es un árbol binario de búsqueda con una **condición de equilibrio**:

- Las alturas de los 2 sub-árboles para cada nodo no difieren en más de una unidad.
- Factor de equilibrio o balance de un nodo se define como altura del subárbol derecho menos altura del subárbol izquierdo para ese nodo
- Cada nodo del árbol AVL puede tener un balance de -1 , 0 , 1
- Para realizar el cálculo, en cada nodo hay que mantener información de su altura.

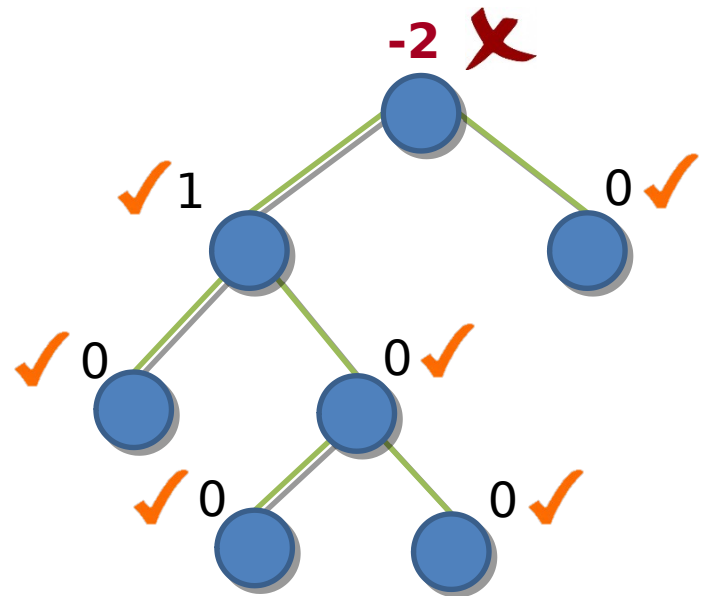


Árboles AVL

Un árbol AVL es un árbol binario de búsqueda con una **condición de equilibrio**:

- Las alturas de los 2 sub-árboles para cada nodo no difieren en más de una unidad.
- Factor de equilibrio o balance de un nodo se define como altura del subárbol derecho menos altura del subárbol izquierdo para ese nodo
- Cada nodo del árbol AVL puede tener un balance de -1 , 0 , 1
- Para realizar el cálculo, en cada nodo hay que mantener información de su altura.

Árbol no AVL

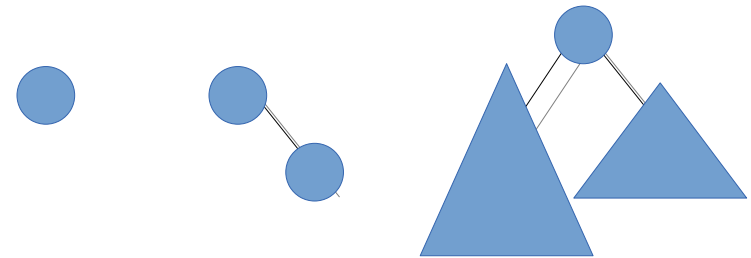


Árboles AVL

La altura de un ABB-AVL con n nodos es $O(\log(n))$

Demostración:

- Si $n(h)$ es el número mínimo de nodos internos de un ABB-AVL con altura h se cumple que:
 - $n(0) = 1$
 - $n(1) = 2$
 - $n(h) = n(h-1) + n(h-2) + 1$
- Obsérvese que al ser mínimo, la altura de los sub-árboles de un nodo X de altura h siempre diferencian en 1 (menos en el caso base). Por eso sumamos $n(h-1) + n(h-2)$ y le sumamos 1 por el nodo X .
- Y esta fórmula se corresponde a la serie de Fibonacci cuyo crecimiento es exponencial.

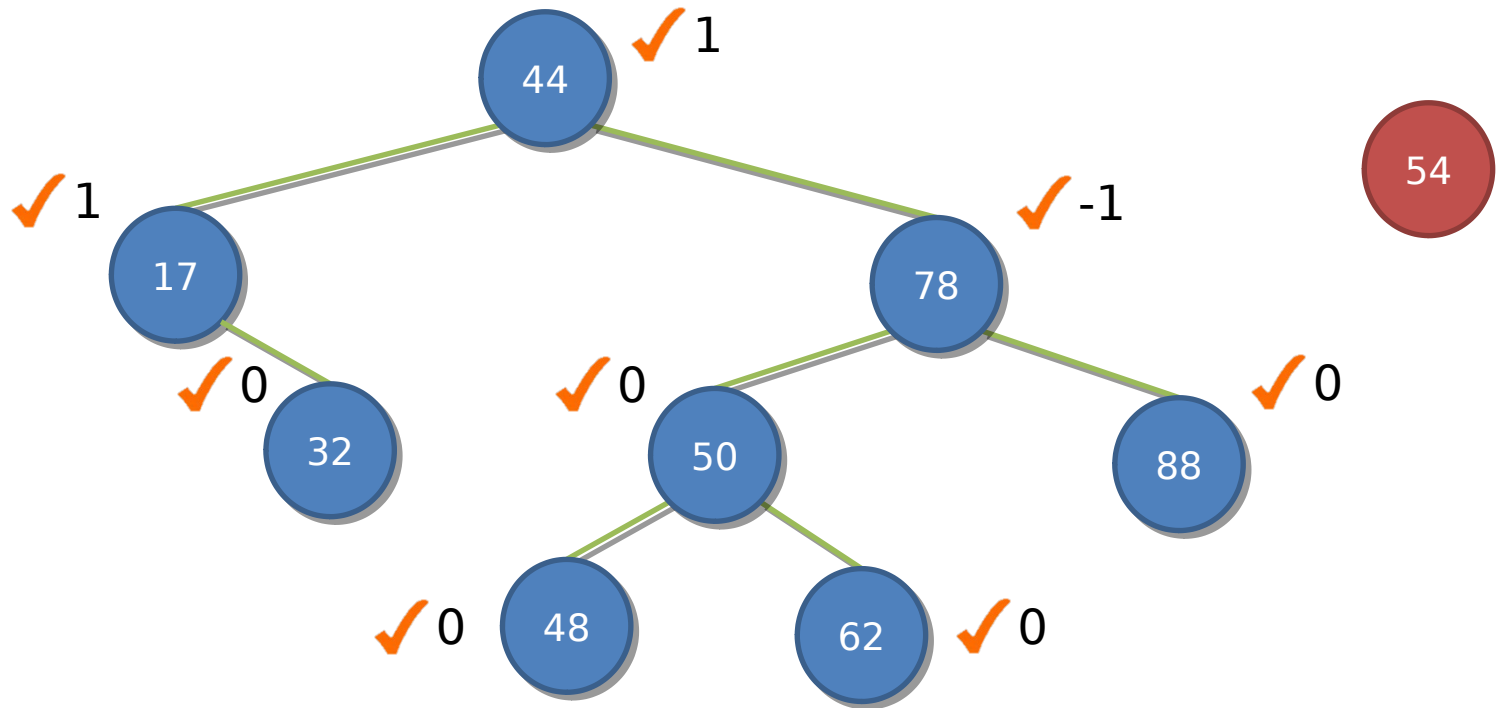


Inserción:

- Se realiza igual que en un AB
- Puede alterar la condición de equilibrio del árbol. Para restaurar el equilibrio hay que aplicar rotaciones. Hay dos tipos de rotaciones:
 - Rotación simple: izq-izq ó dcha-dcha
 - Rotación doble: izq-dcha ó dcha-izq

Árboles AVL - Inserción

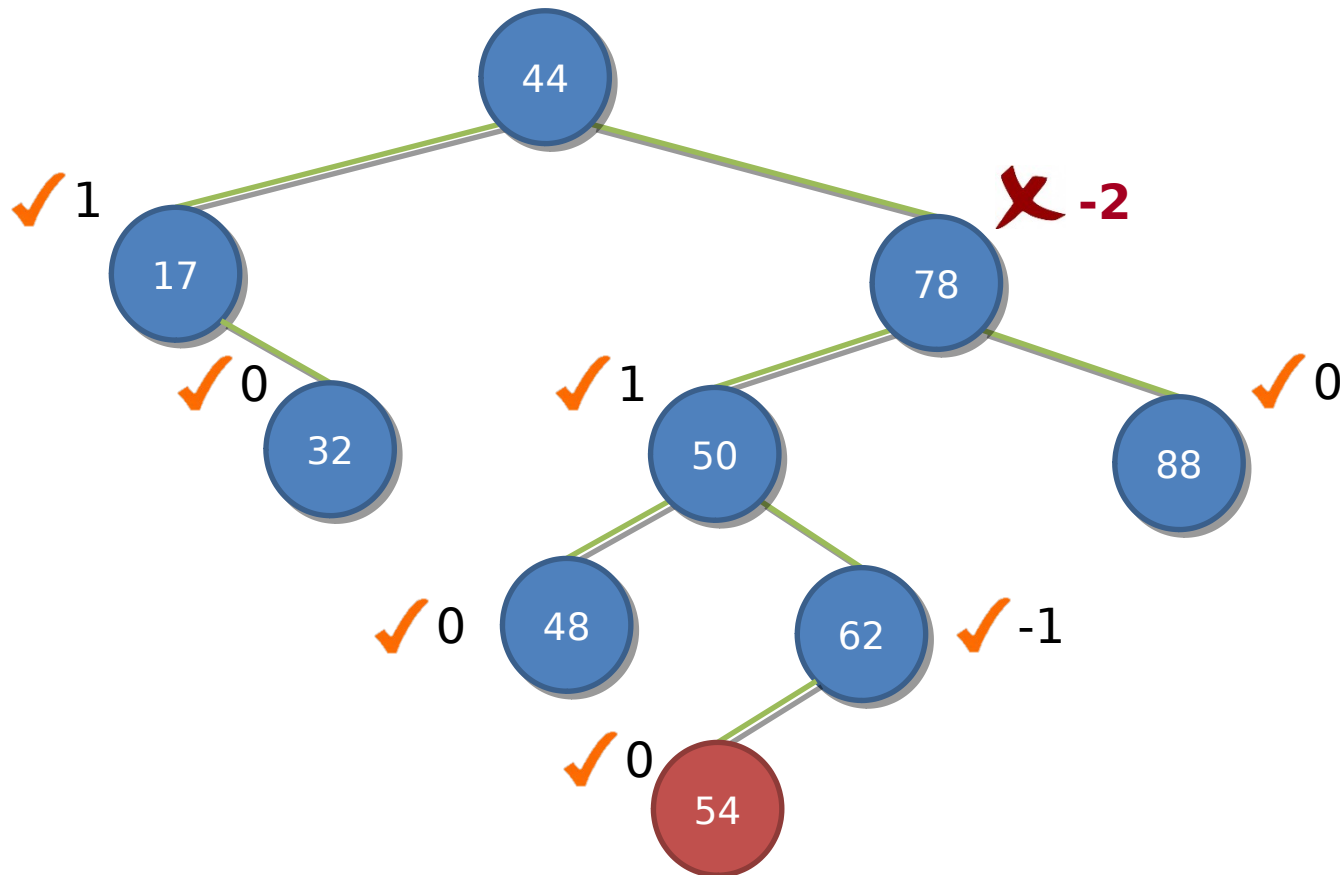
Ejemplo de inserción de nodo con valor 54.



Árboles AVL - Inserción

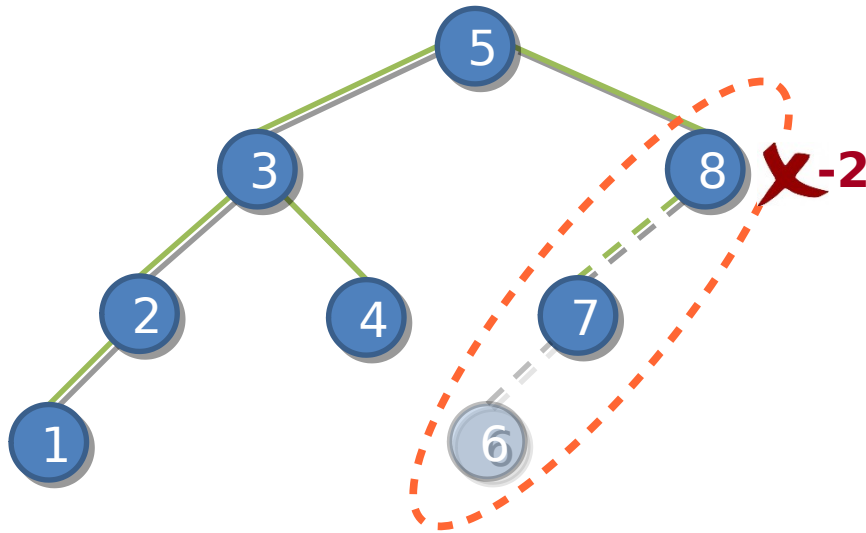
Ejemplo de inserción de nodo con valor 54

- Produce un desequilibrio en el nodo con clave 78

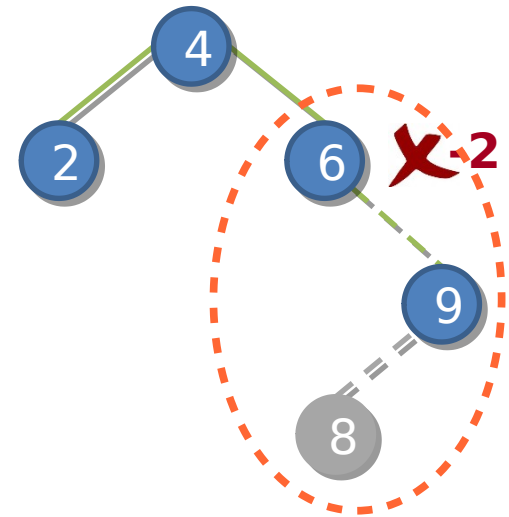


Árboles AVL - Re-estructuración trinodo

Hay 4 casos en los que hay problemas tras la inserción.



Problema izq-izq

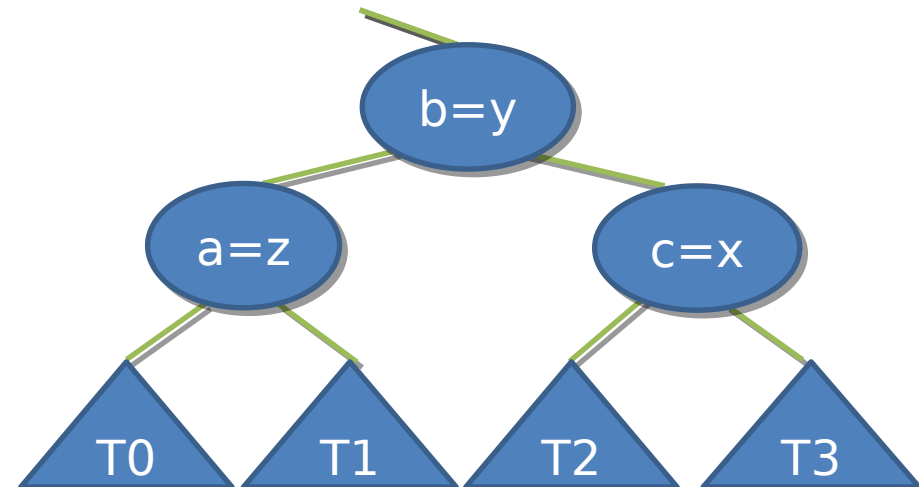
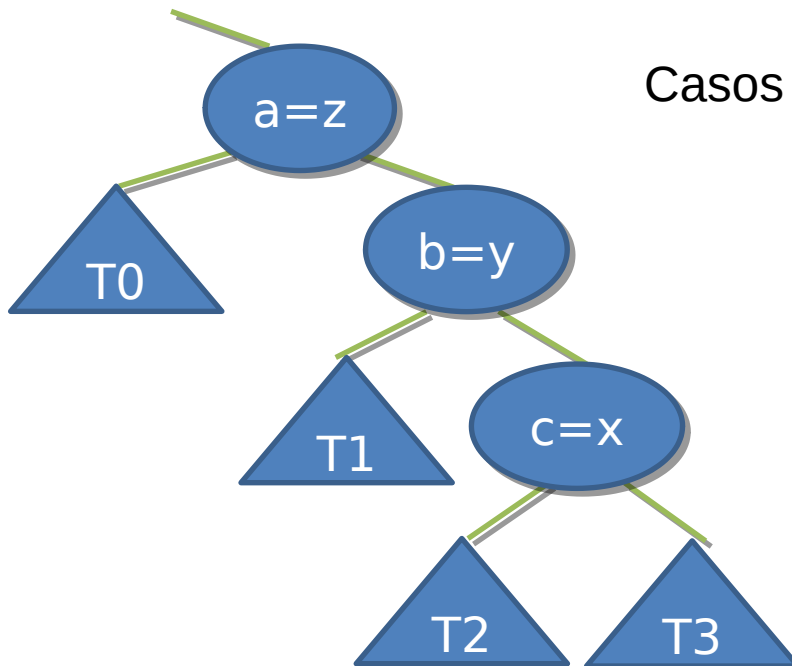


Problema der-izq

- Siempre hay 3 nodos involucrados: en el que se detecta el desequilibrio y los 2 primeros descendientes en la rama que tiene más profundidad.
- Los otros 2 problemas (der-der e izq-der) son simétricos.

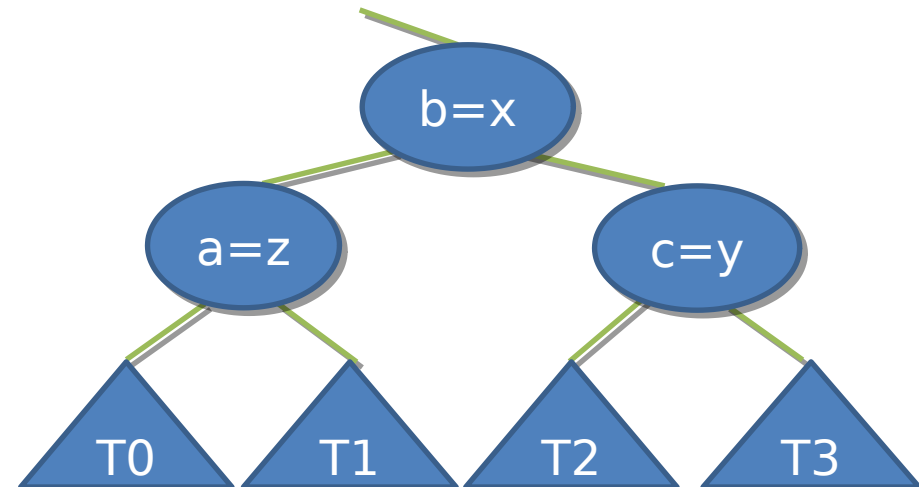
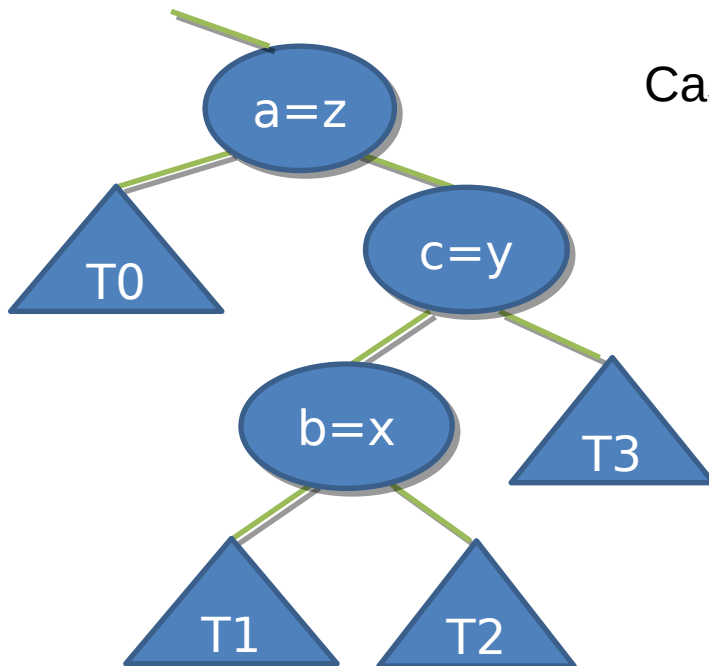
Árboles AVL - Re-estructuración trinodo

Sean las claves a, b, c ordenadas de los nodos involucrados x, y, z ($a < b < c$). La re-estructuración consiste en tomar el nodo con valor intermedio (b) y poner los nodos a y c como hijos de este. Los sub-árboles deben distribuirse adecuadamente.



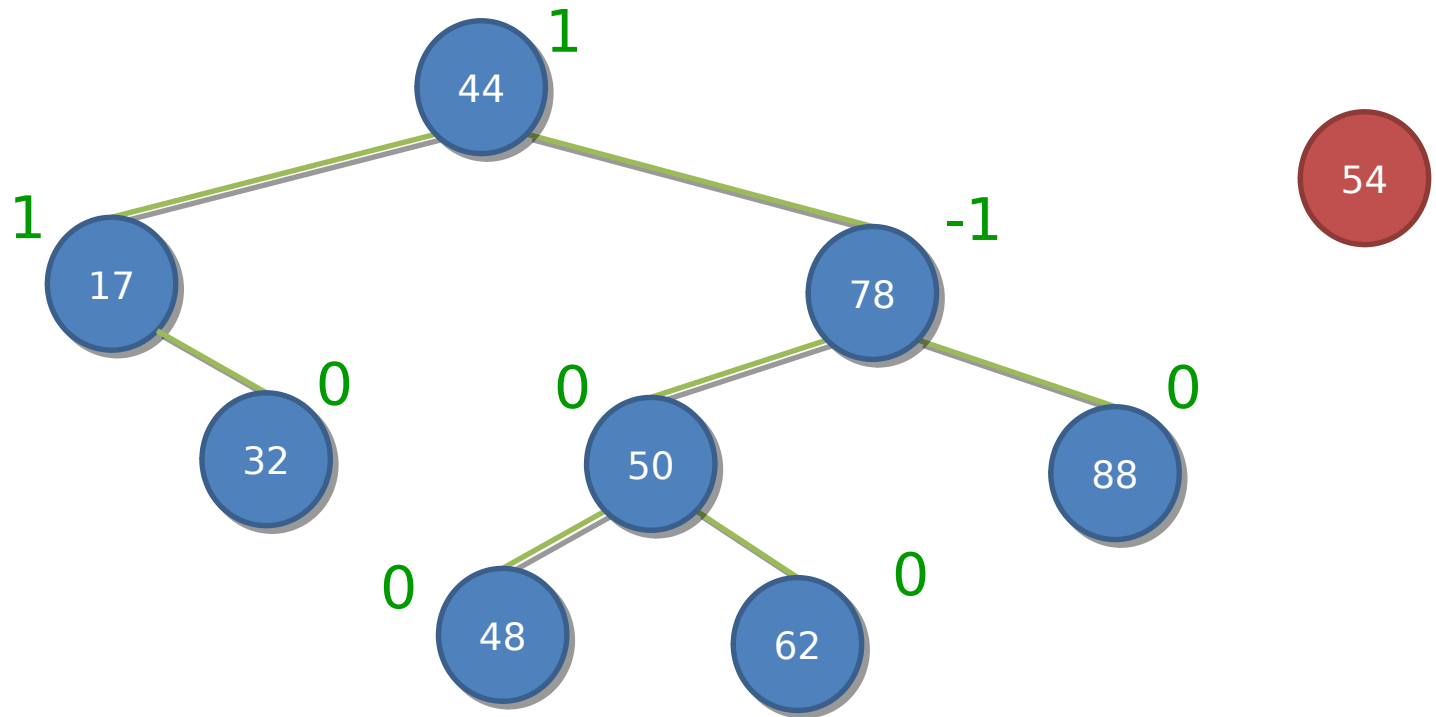
Árboles AVL - Re-estructuración trinodo

Sean las claves a, b, c ordenadas de los nodos involucrados x, y, z ($a < b < c$). La re-estructuración consiste en tomar el nodo con valor intermedio (b) y poner los nodos a y c como hijos de este. Los sub-árboles deben distribuirse adecuadamente.



Árboles AVL - Re-estructuración trinodo

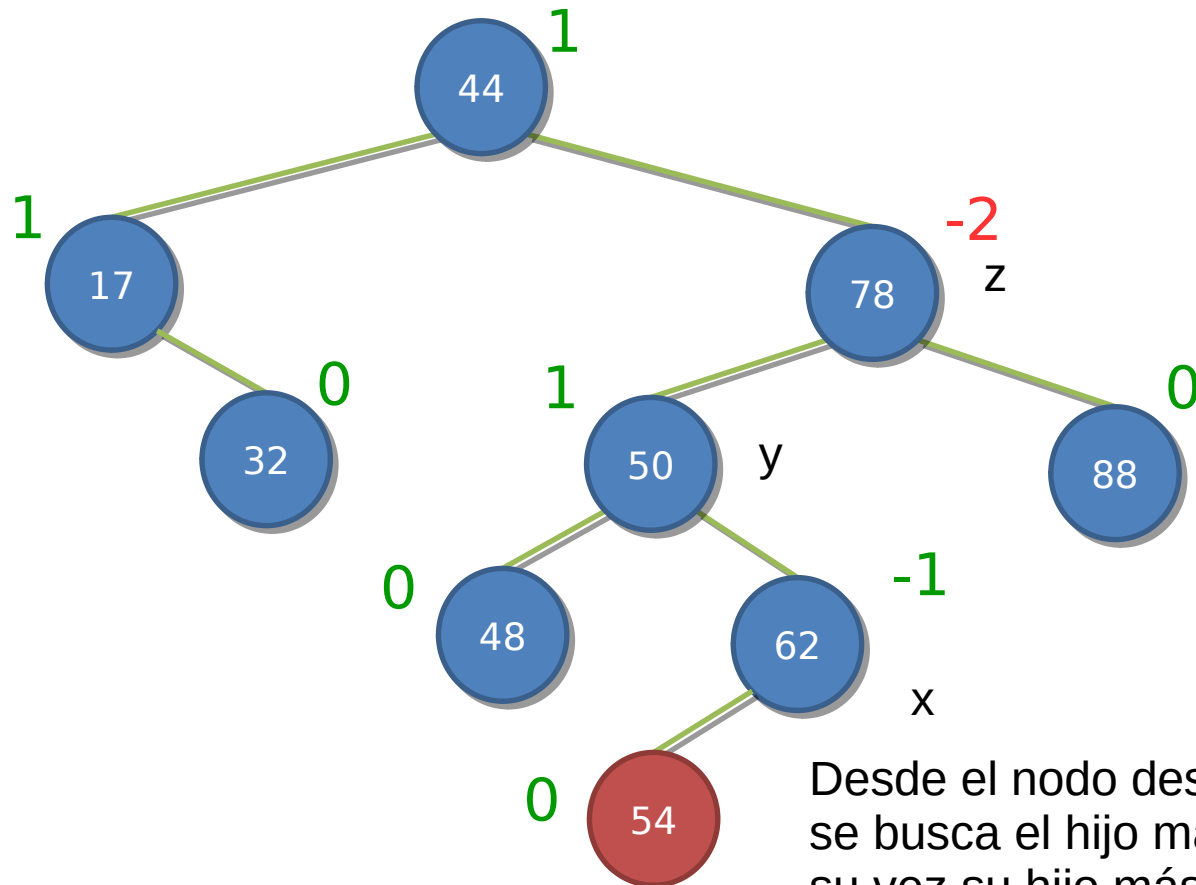
Ejemplo de inserción de nodo con valor 54



Árboles AVL - Re-estructuración trinodo

Ejemplo de inserción de nodo con valor 54

- Produce un desequilibrio en el nodo con clave 78

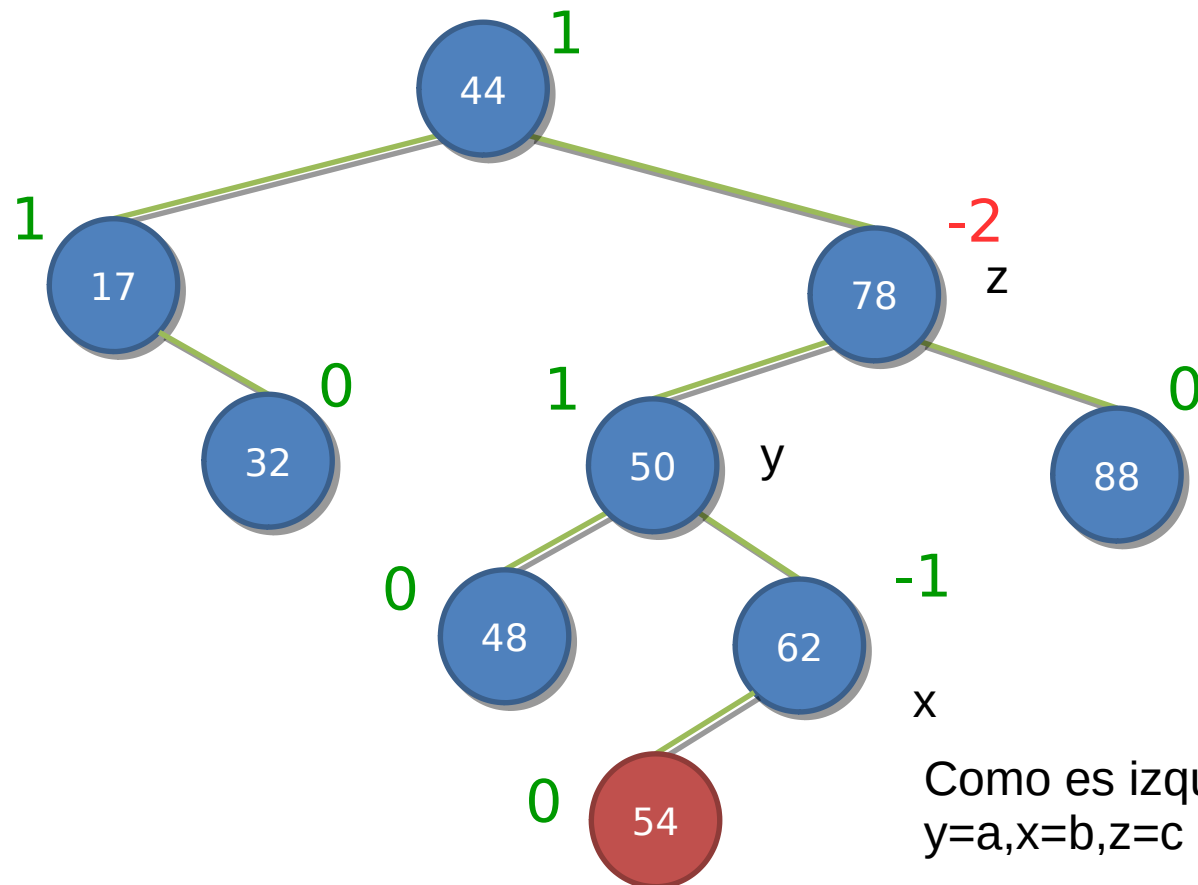


Desde el nodo desequilibrado “z” se busca el hijo más alto “y” y a su vez su hijo más alto “x”.

Árboles AVL - Re-estructuración trinodo

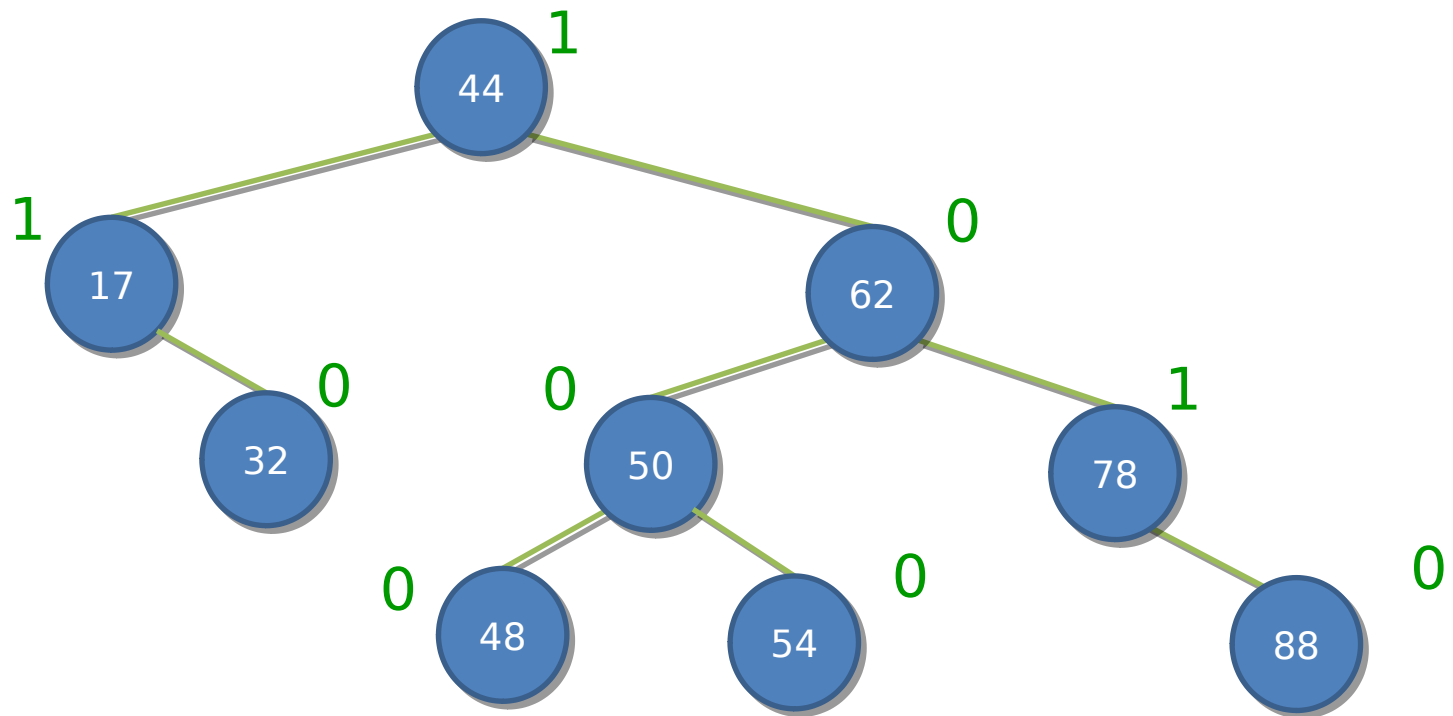
Ejemplo de inserción de nodo con valor 54

- Produce un desequilibrio en el nodo con clave 78



Árboles AVL - Re-estructuración trinodo

Ejemplo de inserción de nodo con valor 54



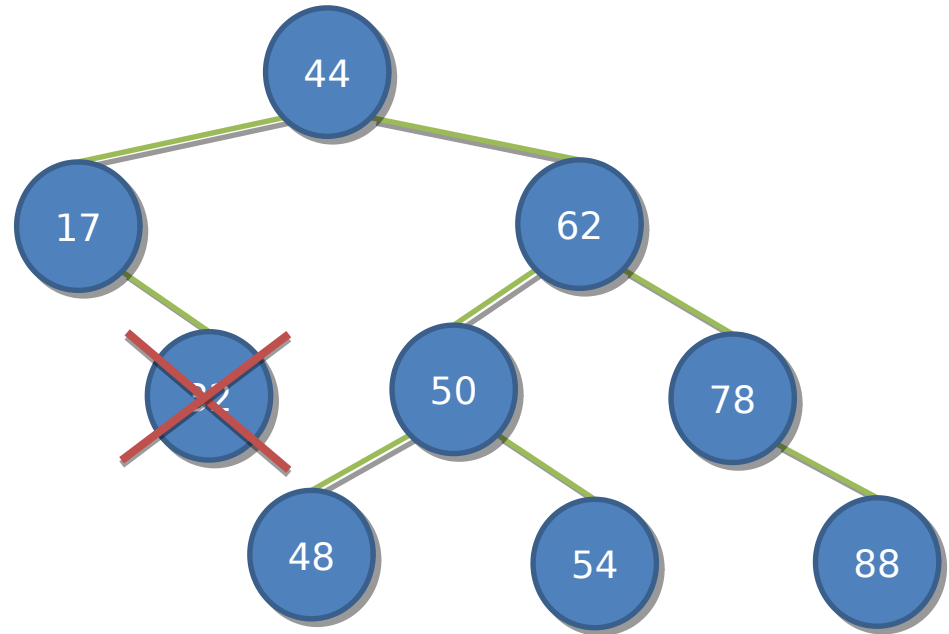
Tras realizar la rotación el árbol queda equilibrado. Si al ir ascendiendo se detectase otro desequilibrio se volvería a re-estructurar.

Árboles AVL - Borrado

- Borrado equivalente a un ABB
 - Caso 1 (simple): nodo a borrar = hoja
 - Caso 2: **nodo a borrar != hoja**
 - **Sustituir** por el valor sucesor (**mínimo de los hijos mayores**) o predecesor(máximo de los hijos menores)

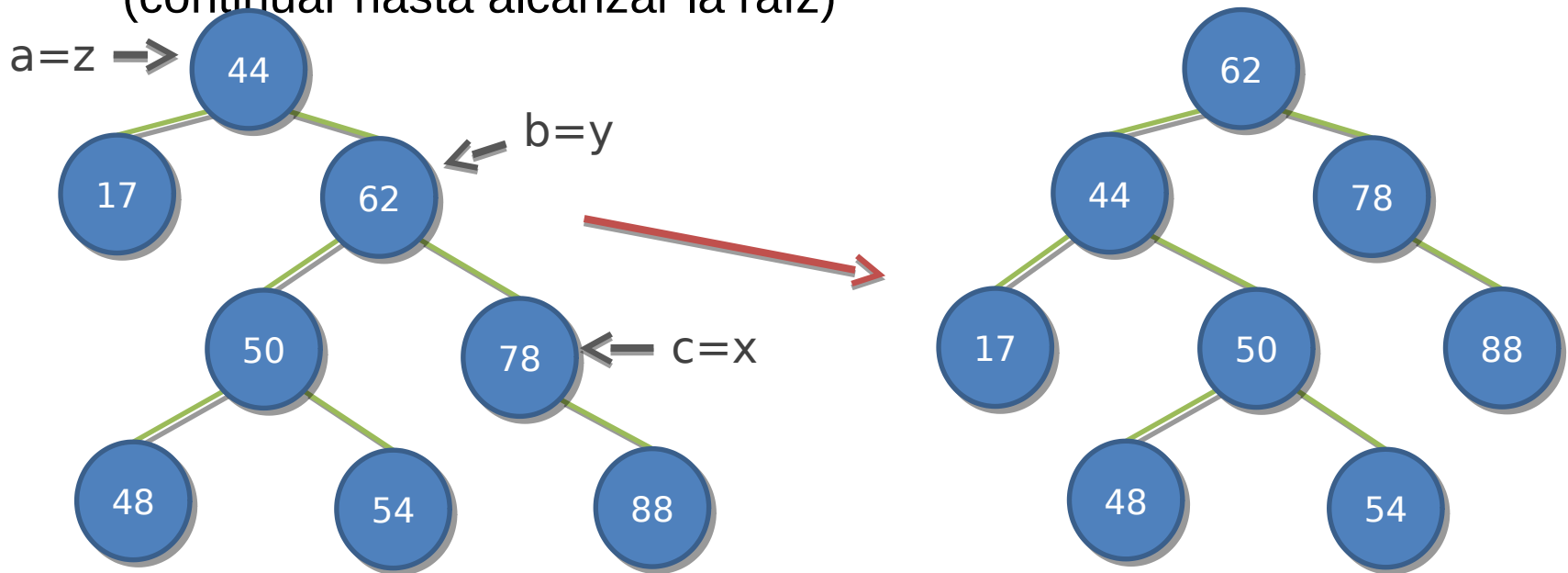
Ejemplo: borrar nodo con clave 32

Después de borrar el 32, el 44 está desequilibrado

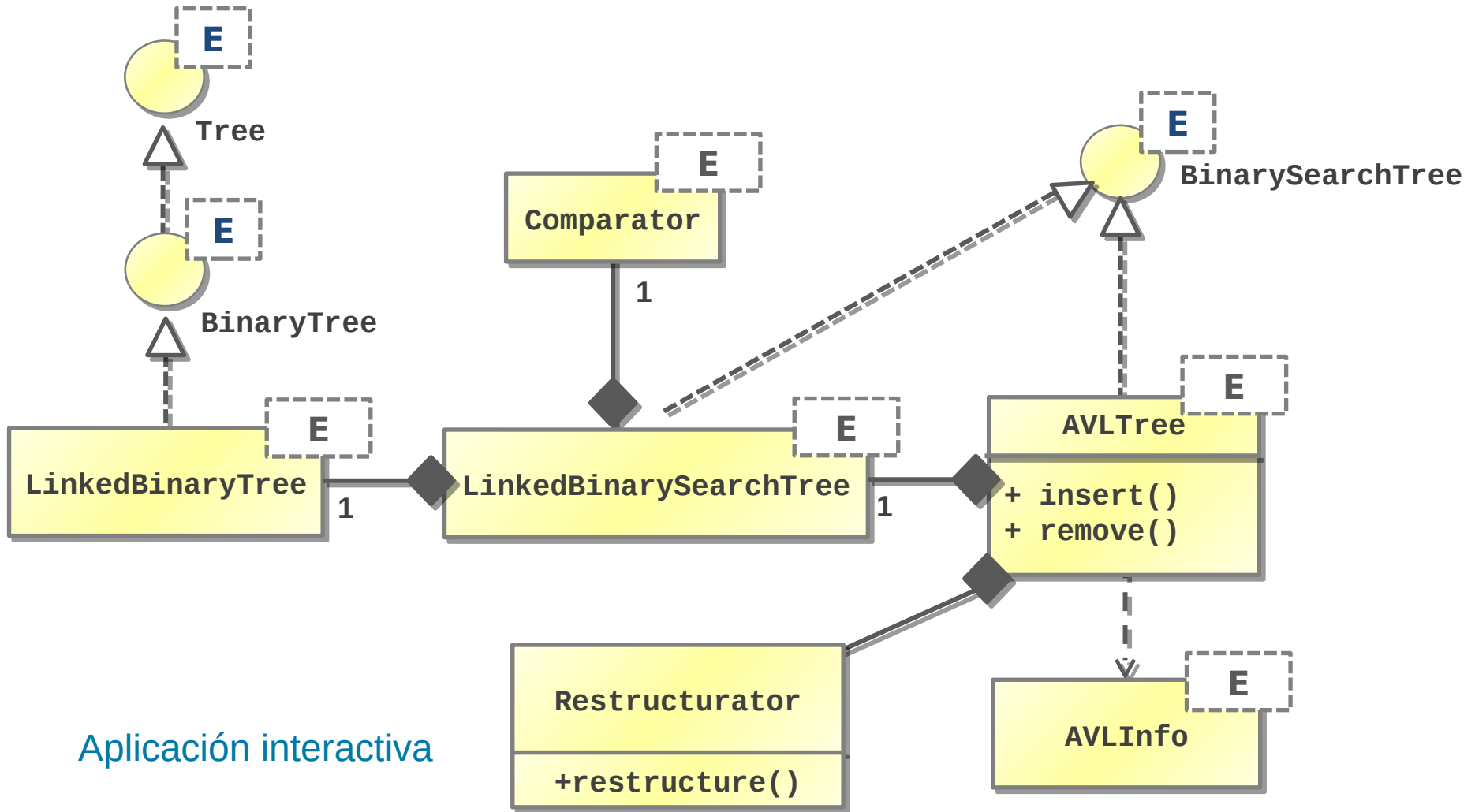


Árboles AVL - Borrado

- **Reequilibrado** tras el borrado equivalente a un ABB
 - Recorrer el árbol hacia la raíz buscando el primer nodo no equilibrado
 - Reestructuración trinodo, partiendo del nodo detectado y buscando la rama de mayor profundidad
 - Podría provocar el desequilibrio en un nodo con mayor altura (continuar hasta alcanzar la raíz)



AVL - Implementación Java



Árboles Rojo-Negro

Árboles rojo-negro (ARN)

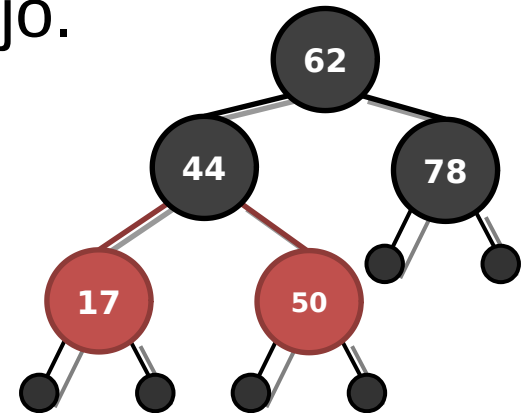
Idea intuitiva de árbol Rojo-Negro



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

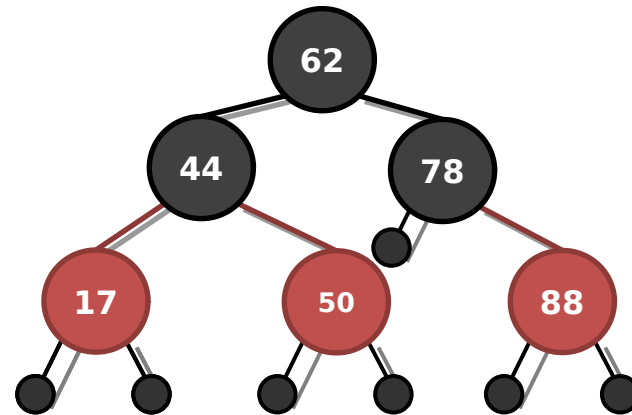
Ya que esto no siempre es posible, se plantea crear un árbol donde eso se cumpla excepto por unos pocos nodos que pintaremos de rojo.

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



Árboles rojo-negro (ARN)

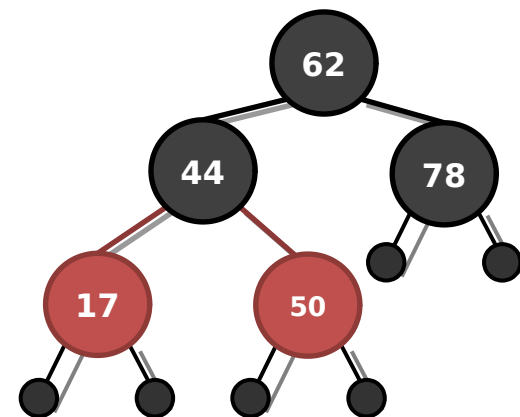
- **ARN**: árbol binario de búsqueda con nodos coloreados de rojo o negro
- La forma de colorear los nodos asegura que ningún camino (raíz-hoja) es más del doble de largo que otro
- **Borrados** en ARN
 - menos operaciones de reestructuración que los AVL
- Cada nodo **contiene**:
 - color
 - valor
 - hijoIzq, hijoDer
 - padre



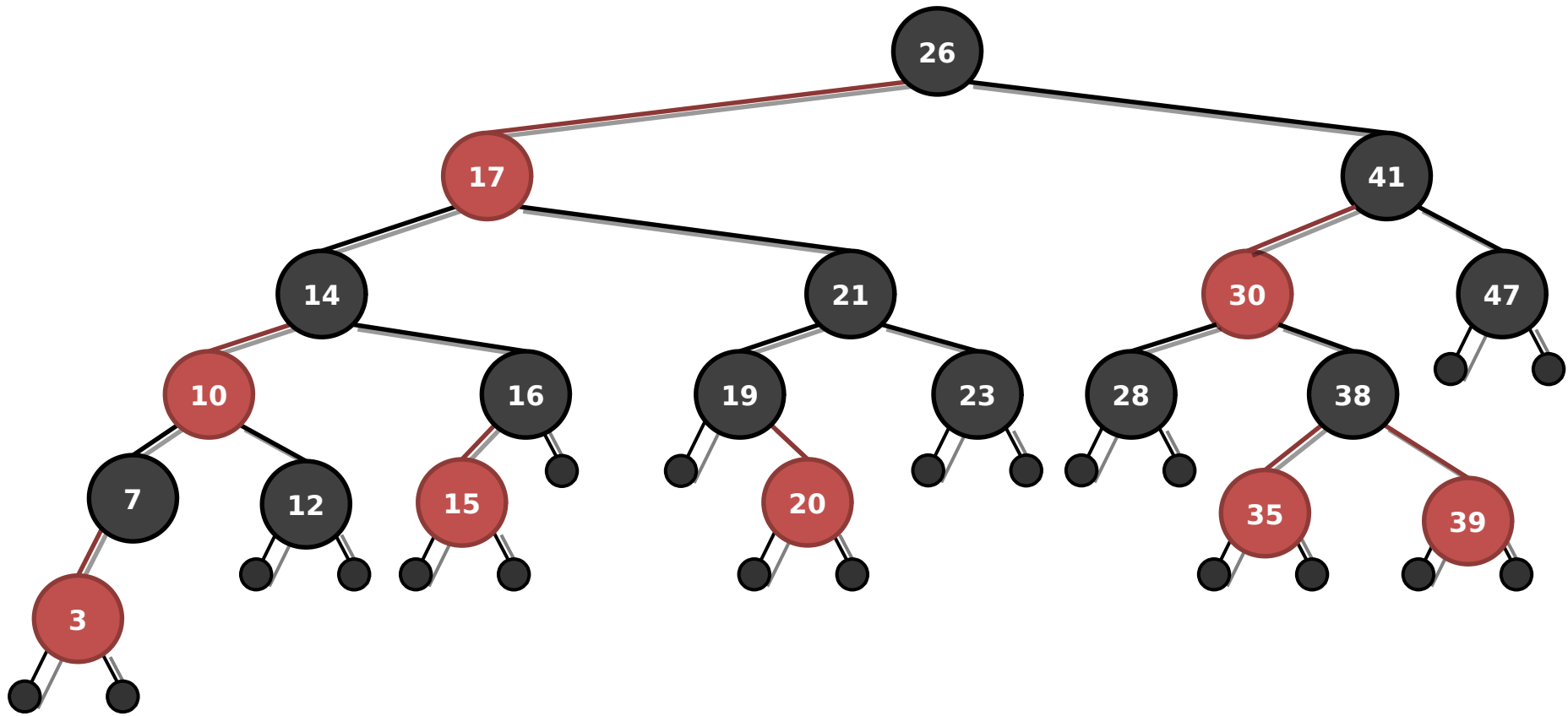
Árboles rojo-negro. Propiedades

Propiedades de los ARN

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



Árboles rojo-negro. Propiedades



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

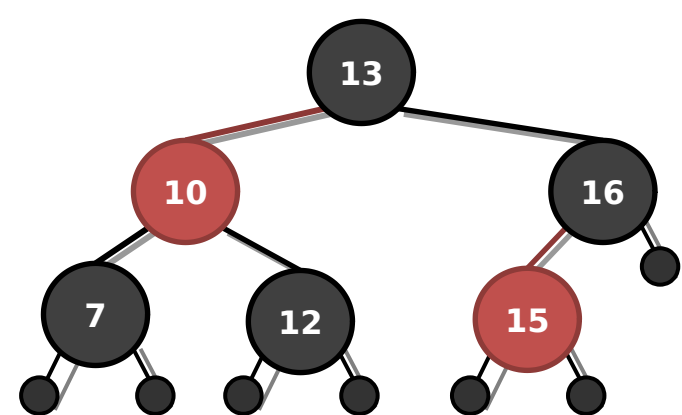
Árboles rojo-negro. Características

- **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 un elemento en un árbol rojo negro es $O(\log_2 n)$


Árboles Rojo-Negro. Inserciones

Árboles rojo-negro. Inserciones

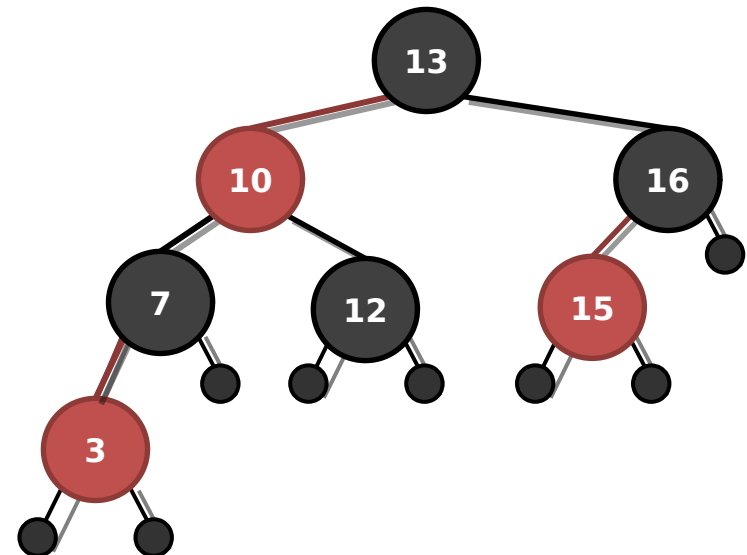
- Insertar un valor en un ARN consiste en ejecutar la inserción del ABB y colorear el nodo como rojo
 - Exceptuar la raíz que se colorea de negro
 - Los dos hijos del nuevo nodo insertado se colorean de negro (hojas nulas)
- Ejemplo: insertar el valor




Árboles rojo-negro. Inserciones

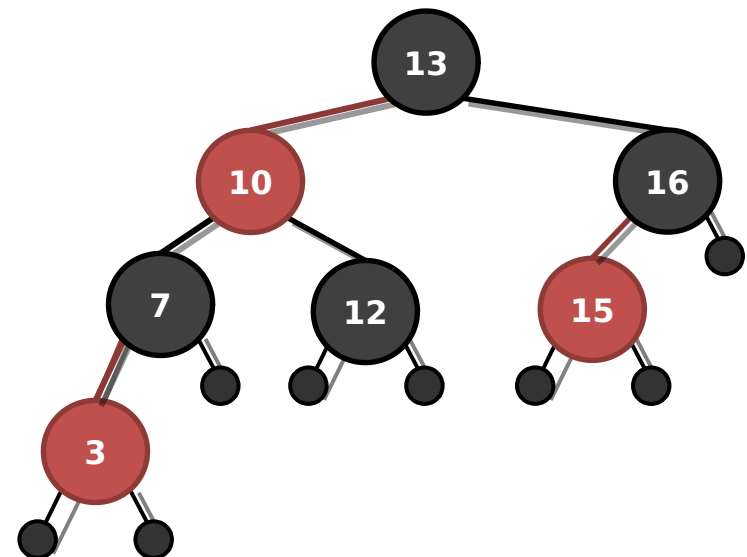
- Insertar un valor en un ARN consiste en ejecutar la **inserción del ABB** y colorear el nodo como rojo
 - Exceptuar la **raíz** que se colorea de **negro**
 - Los **dos hijos** del nuevo nodo insertado se colorean de **negro** (hojas nulas)
- Ejemplo: insertar el valor 

Si el padre del nodo insertado es negro, se satisfacen todas las propiedades



Árboles rojo-negro. Inserciones

- Insertar un valor en un ARN consiste en ejecutar la **inserción del ABB** y colorear el nodo como rojo
 - Exceptuar la **raíz** que se colorear de **negro**
 - Los **dos hijos** del nuevo nodo insertado se colorean de **negro** (hojas nulas)
- Ejemplo: insertar el valor 



Árboles rojo-negro. Inserciones

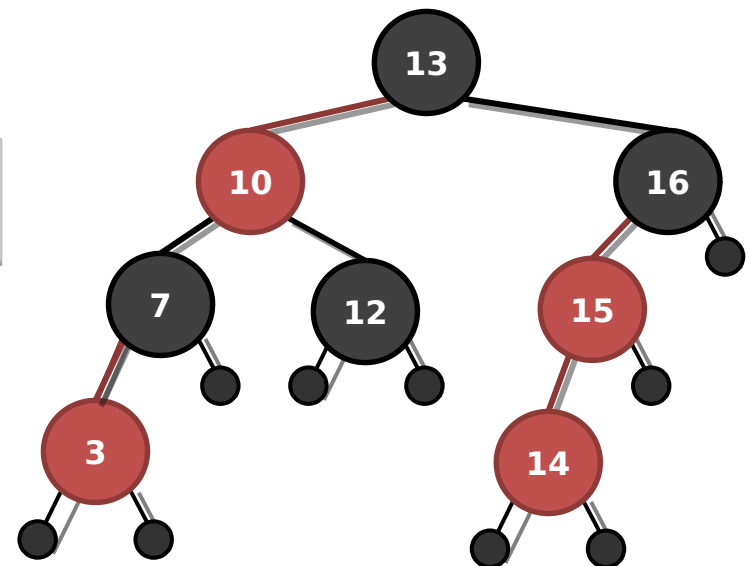
- Insertar un valor en un ARN consiste en ejecutar la **inserción del ABB** y colorear el nodo como rojo
 - Exceptuar la **raíz** que se colorea de **negro**
 - Los **dos hijos** del nuevo nodo insertado se colorean de **negro** (hojas nulas)

- Ejemplo: insertar el valor

14

Si el padre es rojo, la propiedad 3 no estaría satisfecha!!!

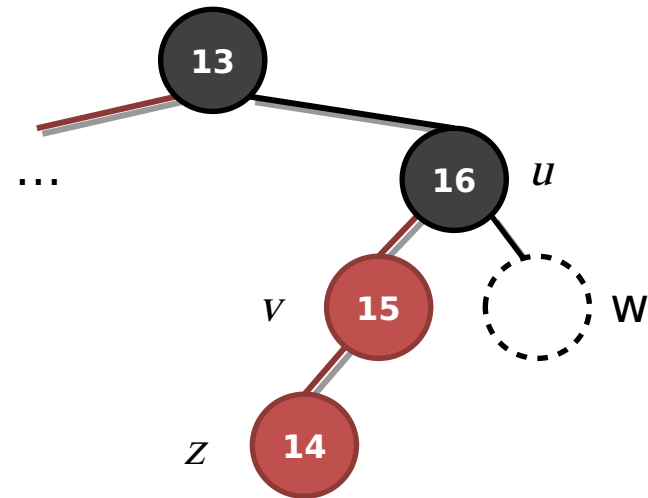
A esta violación de la propiedad interna se le llama doble rojo



ARN. Resolver doble rojo

- Para remediar un doble rojo se examinan 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

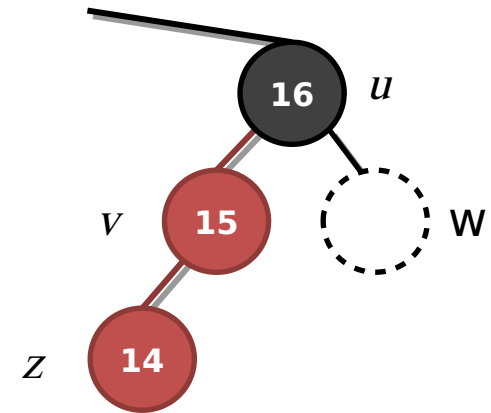


ARN. Resolver doble rojo (Caso 1)

Caso 1: el tío del nodo insertado es negro



Podemos llevar uno de los rojos, del doble rojo, a la posición del tío. Repartiendo los rojos solucionamos el problema del doble rojo.



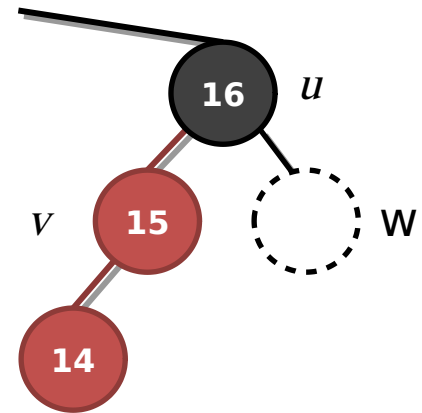
ARN. Resolver doble rojo (Caso 1)

Caso 1: el tío del nodo insertado es negro

Podemos llevar uno de los rojos, del doble rojo, a la posición del tío. Repartiendo los rojos solucionamos el problema del doble rojo.

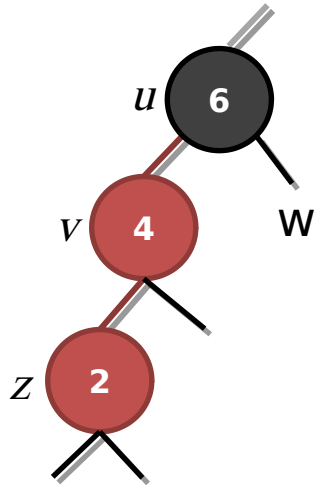
Para ello se debe:

- Aplicar una reestructuración trinodo
 - Dado el nodo insertado z , su padre v y su abuelo u , se **re-nombran** temporalmente como a (menor), b (medio), c (mayor)
 - El abuelo u se **reemplaza** por b , y se hace que los nodos a y c sean **hijos** de b
- Colorear b de **negro** y a y c de **rojo**

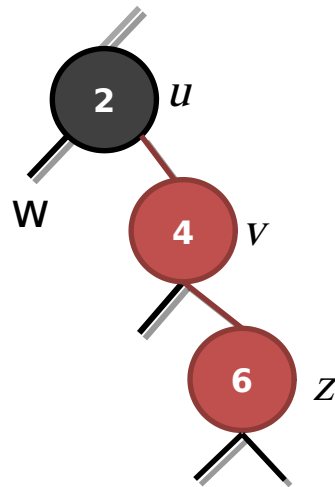


ARN. Resolver doble rojo (Caso 1)

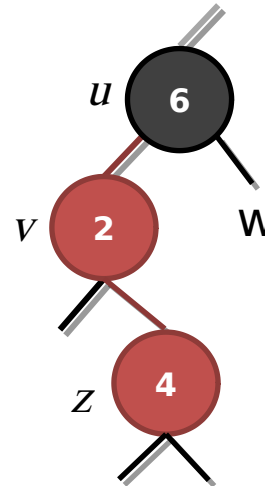
Sólo cuatro posibles configuraciones del *Caso 1*



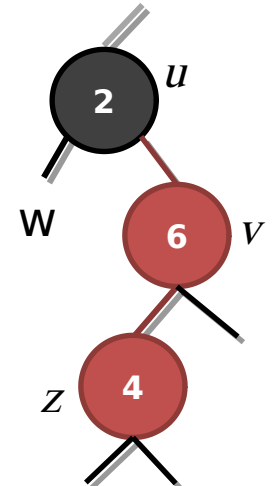
RESTRUCTURACIÓN
TRINODO



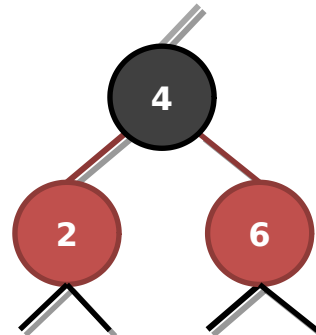
RESTRUCTURACIÓN
TRINODO



RESTRUCTURACIÓN
TRINODO




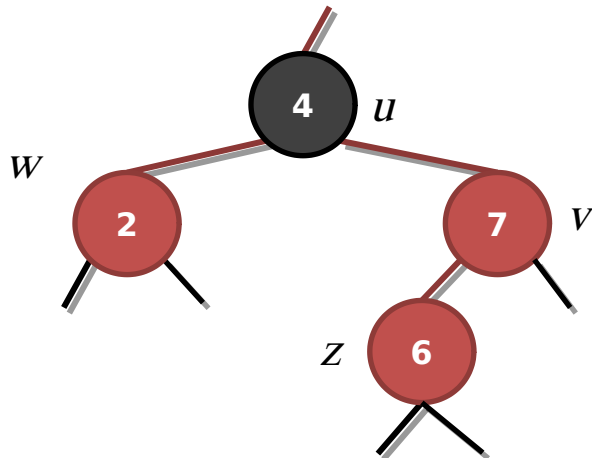
RESTRUCTURACIÓN
TRINODO



ARN. Resolver doble rojo (Caso 2)

Caso 2: el tío del nodo insertado es rojo

 Un recoloreado a negro en ambas ramas, mientras se colorea a rojo un ascendiente para no cambiar la altura negra, deshace el doble rojo.

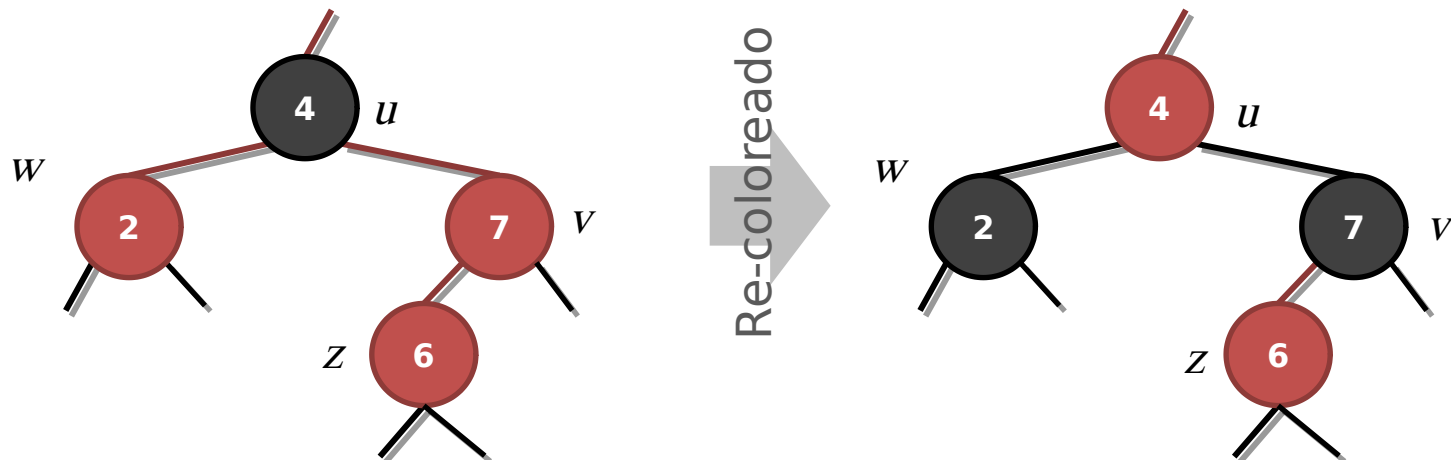


ARN. Resolver doble rojo (Caso 2)

Caso 2: el tío del nodo insertado es rojo

Un recoloreado a negro en ambas ramas, mientras se colorea a rojo un ascendiente para no cambiar la altura negra, deshace el doble rojo.

- Aplicar un **recoloreado** del árbol
- El **padre** v y el **tío** w se colorean de **negro**
- El **abuelo** u de **rojo**.
 - Si u es la **raíz** se debe mantener negro

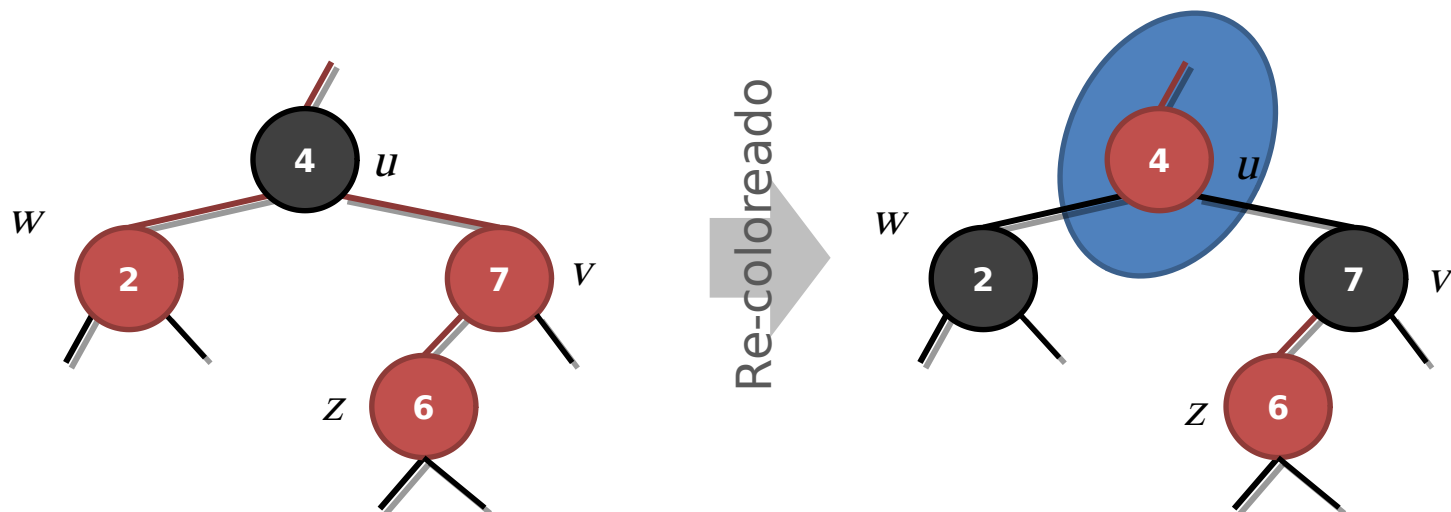


ARN. Resolver doble rojo (Caso 2)

Caso 2: el tío del nodo insertado es rojo

Tras el re-coloreado, puede aparecer el problema del doble rojo en el abuelo (u). Si aparece, aplicar el caso que le corresponda (Caso 1 o Caso 2).

El re-coloreado, o elimina el problema del doble rojo o lo propaga

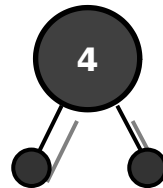


ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17

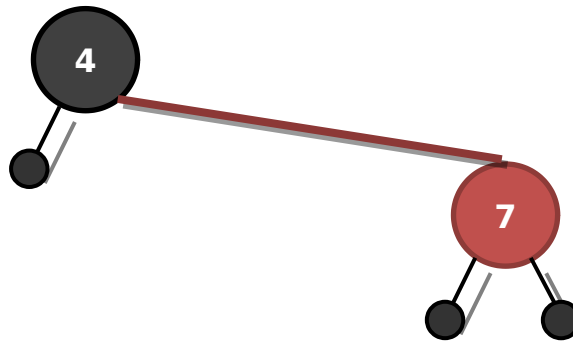
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



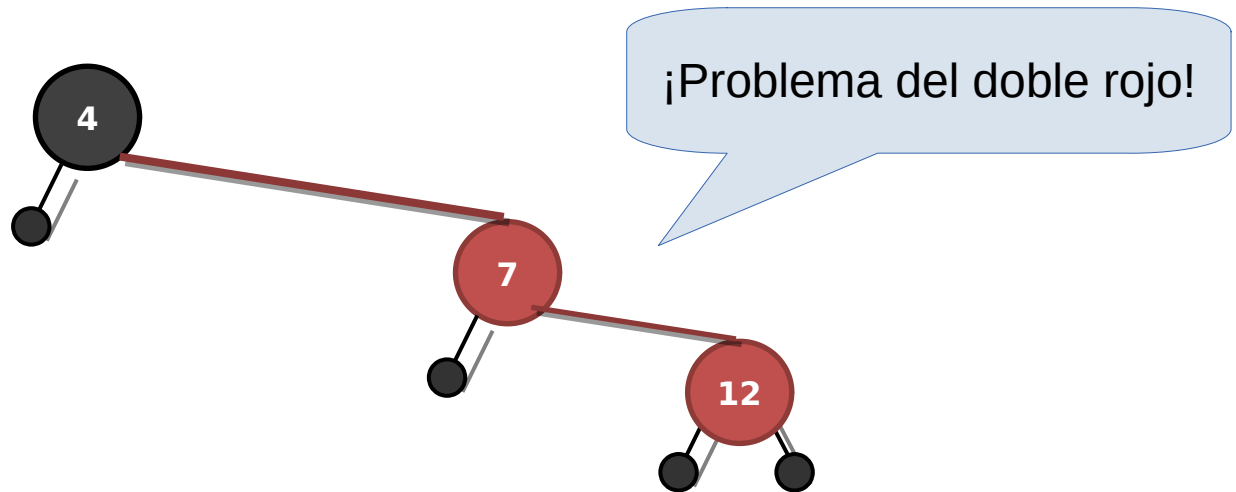
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



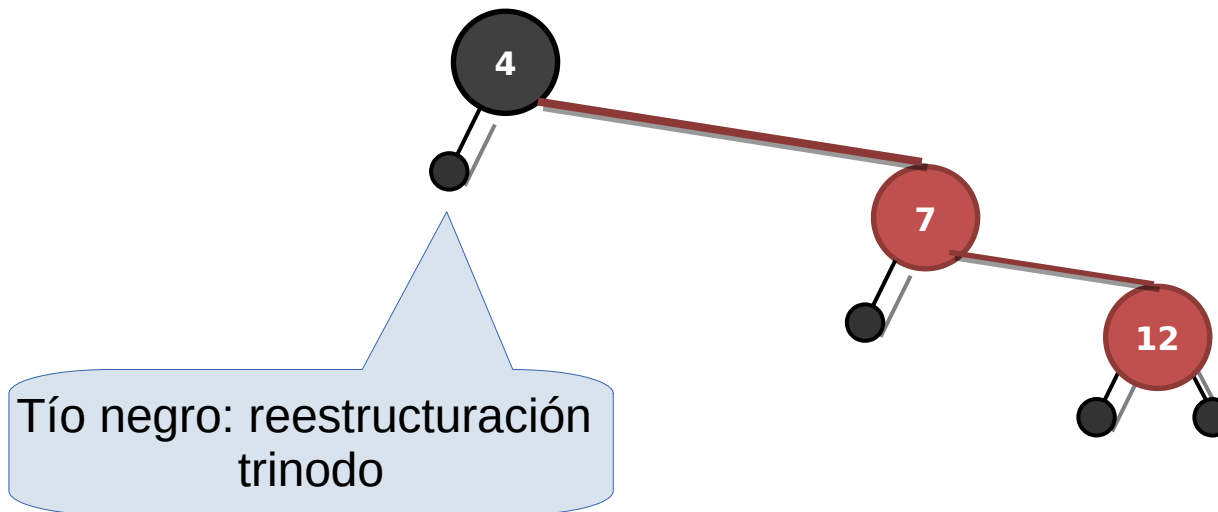
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, **12**, 15, 3, 5, 14, 18, 16, 17



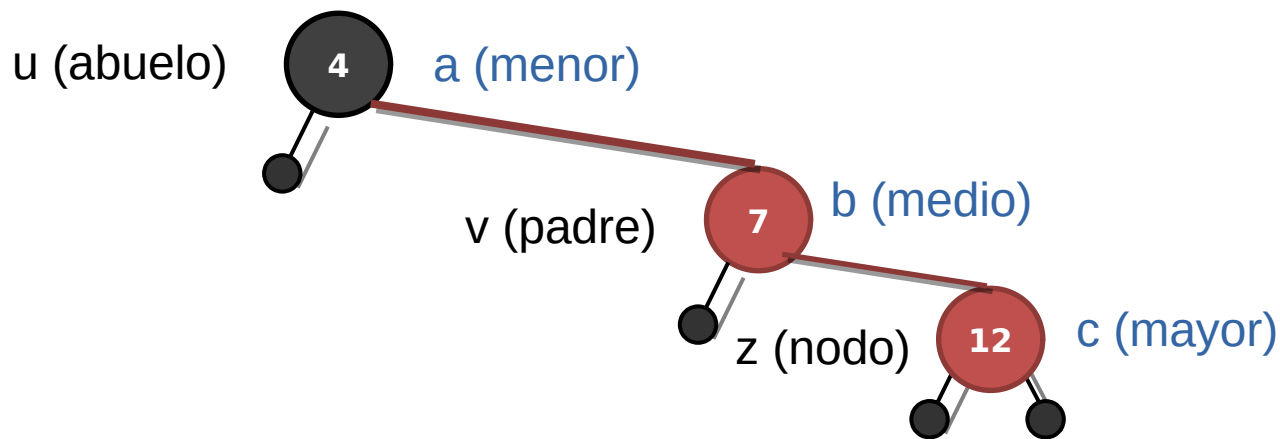
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, **12**, 15, 3, 5, 14, 18, 16, 17



ARN. Ejemplo inserciones

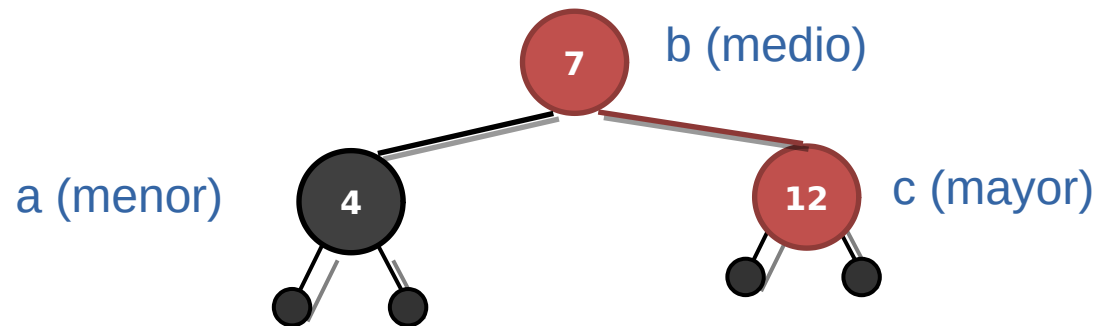
- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, **12**, 15, 3, 5, 14, 18, 16, 17



- 1) El abuelo u se **reemplaza** por b, y se hace que los nodos a y c sean **hijos** de b
- 2) Colorear b de **negro** y a y c de **rojo**

ARN. Ejemplo inserciones

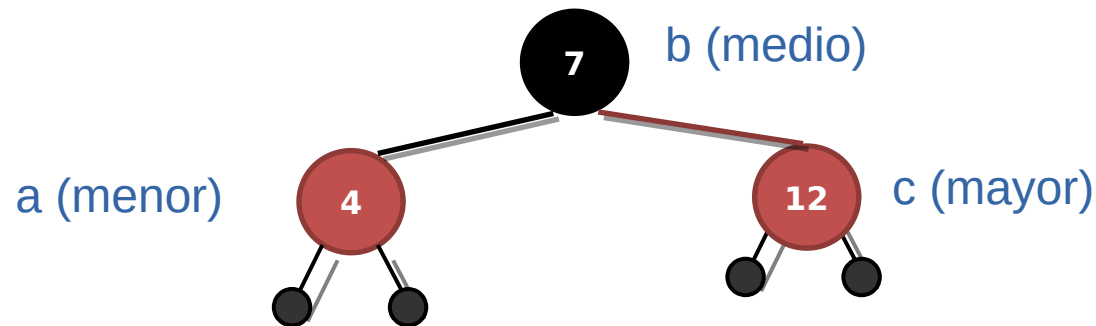
- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, **12**, 15, 3, 5, **14**, 18, 16, 17



- 1) El abuelo u se **reemplaza** por b, y se hace que los nodos a y c sean **hijos** de b

ARN. Ejemplo inserciones

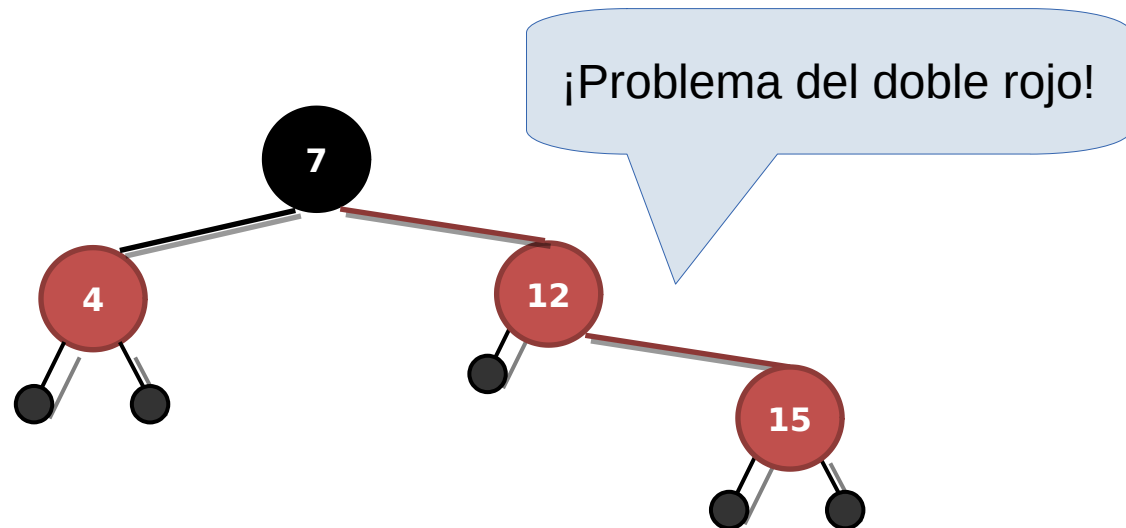
- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, **12**, 15, 3, 5, **14**, 18, 16, 17



- 1) El abuelo u se **reemplaza** por b, y se hace que los nodos a y c sean **hijos** de b
- 2) Colorear b de **negro** y a y c de **rojo**

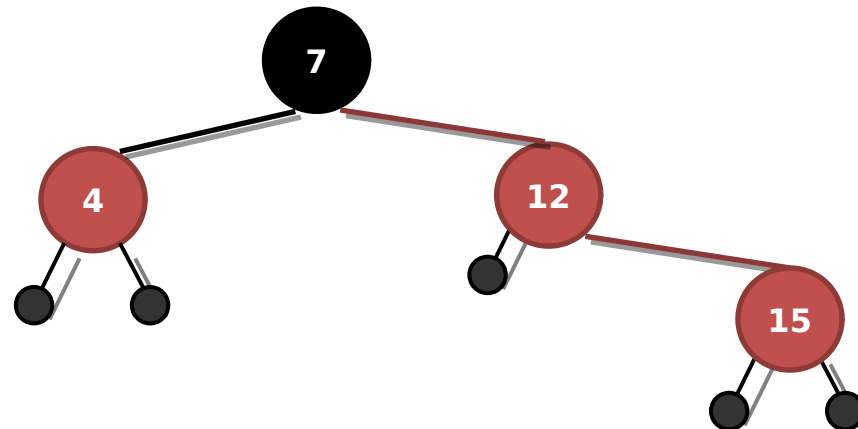
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, **15**, 3, 5, 14, 18, 16, 17



ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, **15**, 3, 5, 14, 18, 16, 17

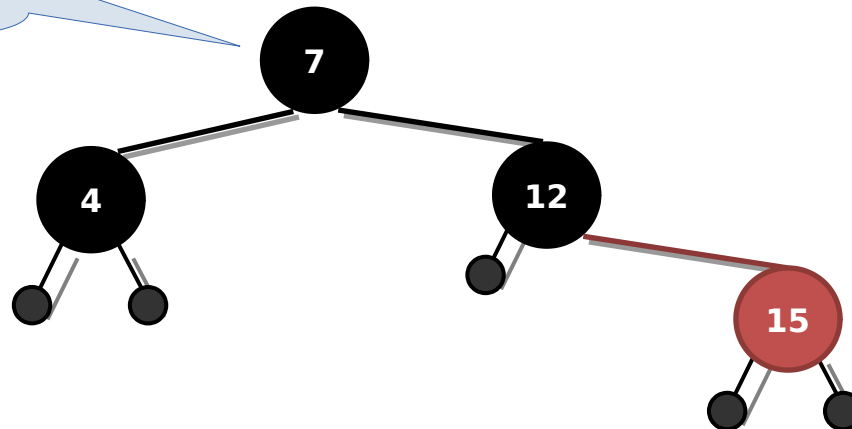


Tío rojo: recoloreado

ARN. Ejemplo inserciones

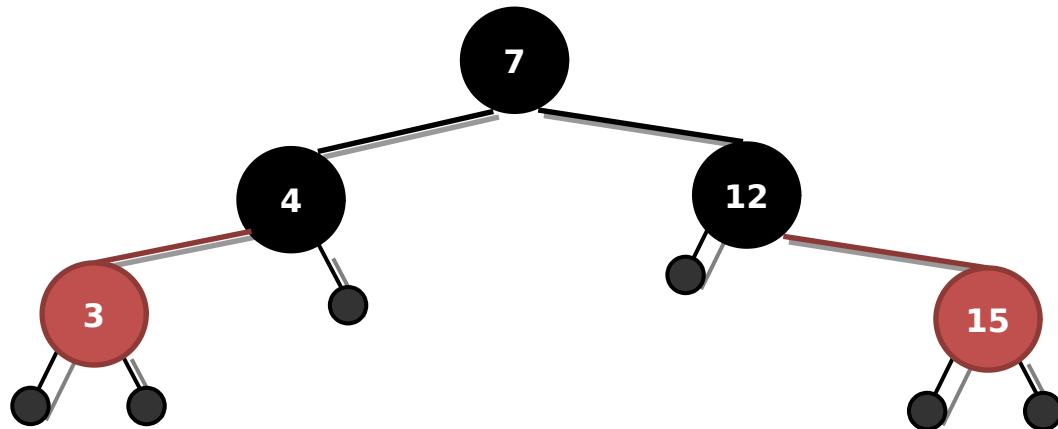
- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, **15**, 3, 5, 14, 18, 16, 17

Como el abuelo es la raíz
no puede ser rojo



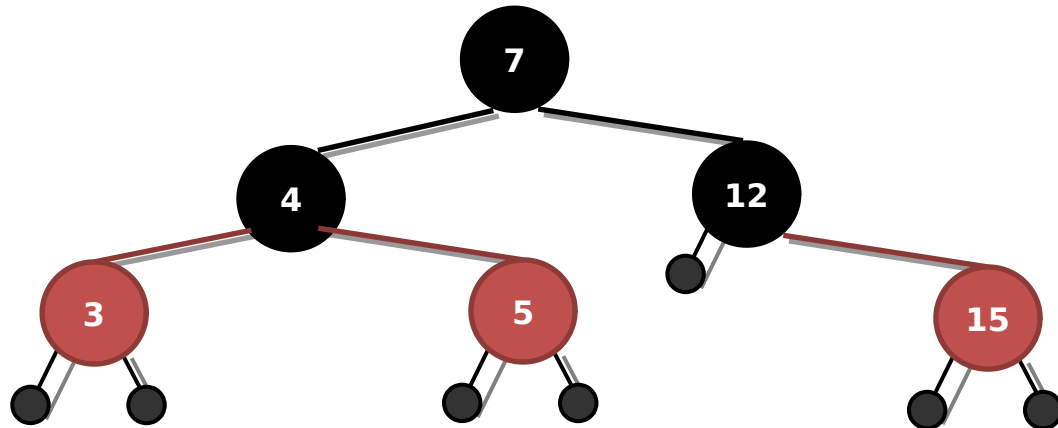
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



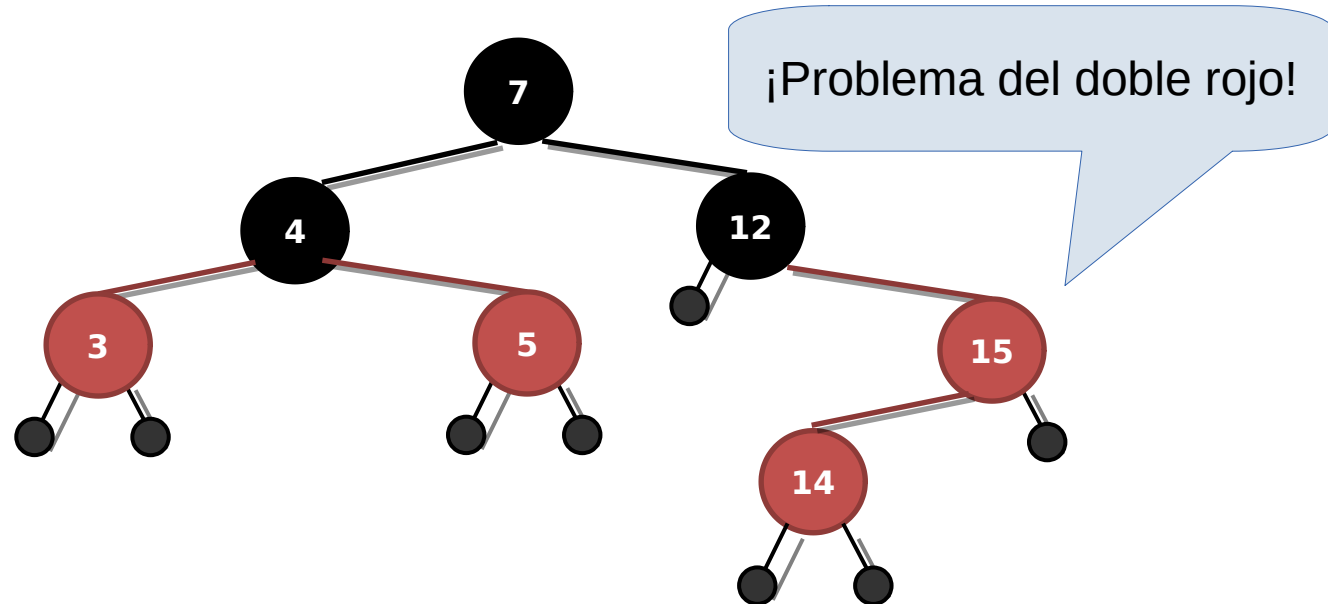
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



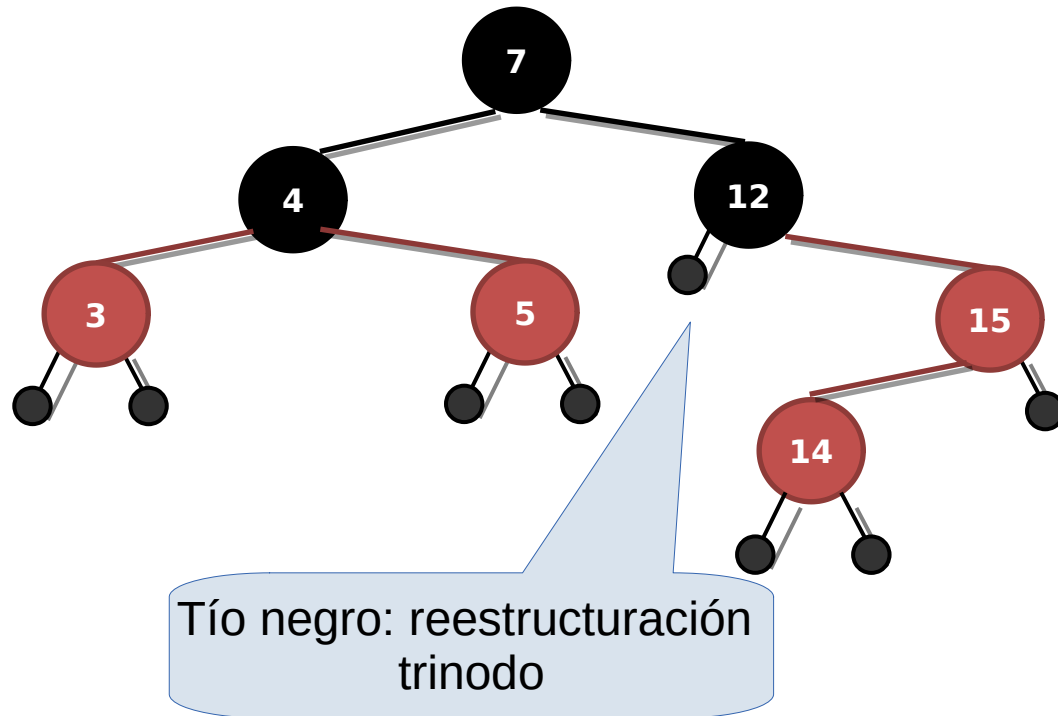
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, **14**, 18, 16, 17



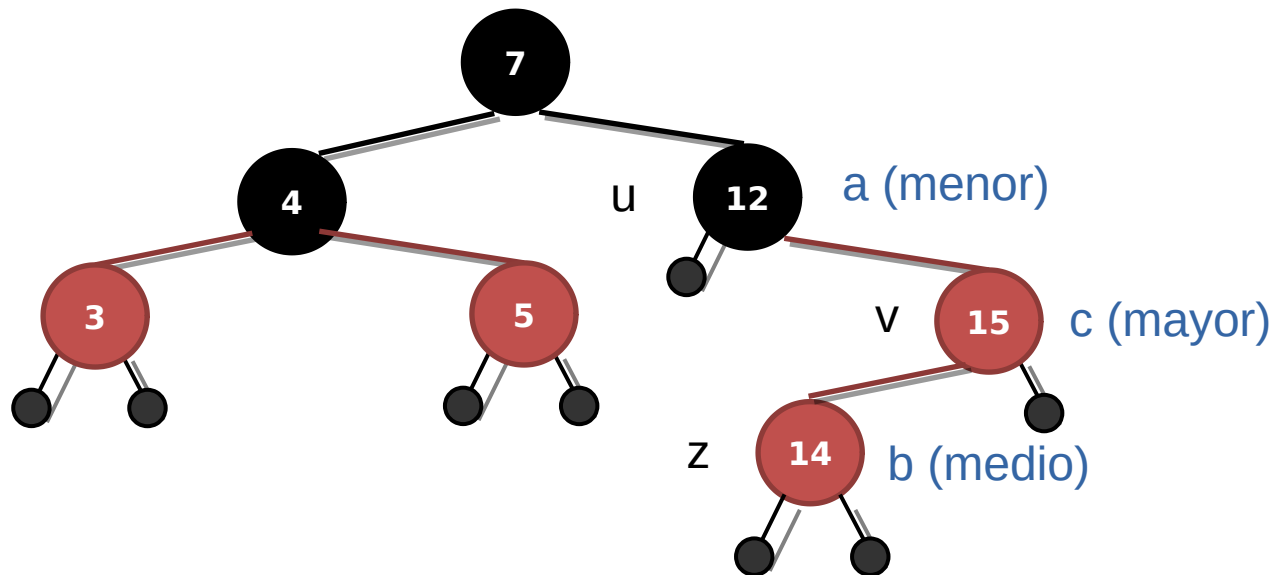
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, **14**, 18, 16, 17



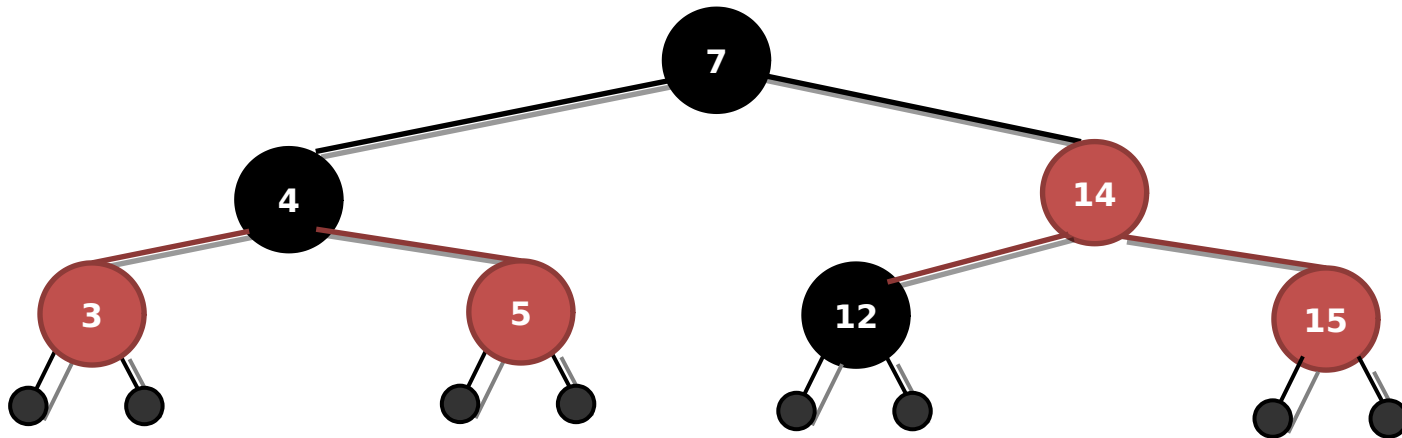
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, **14**, 18, 16, 17



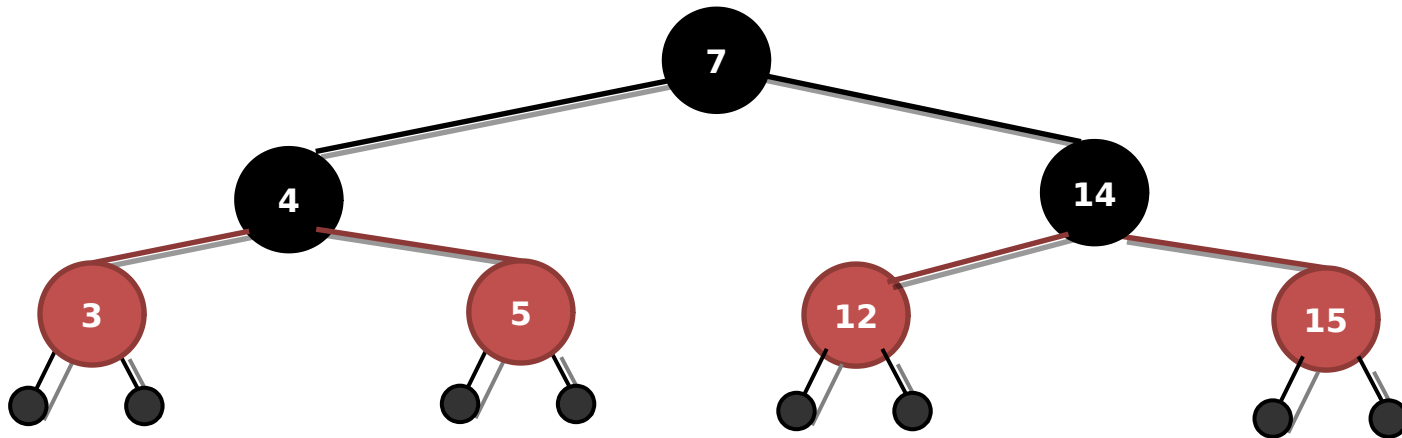
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, **14**, 18, 16, 17



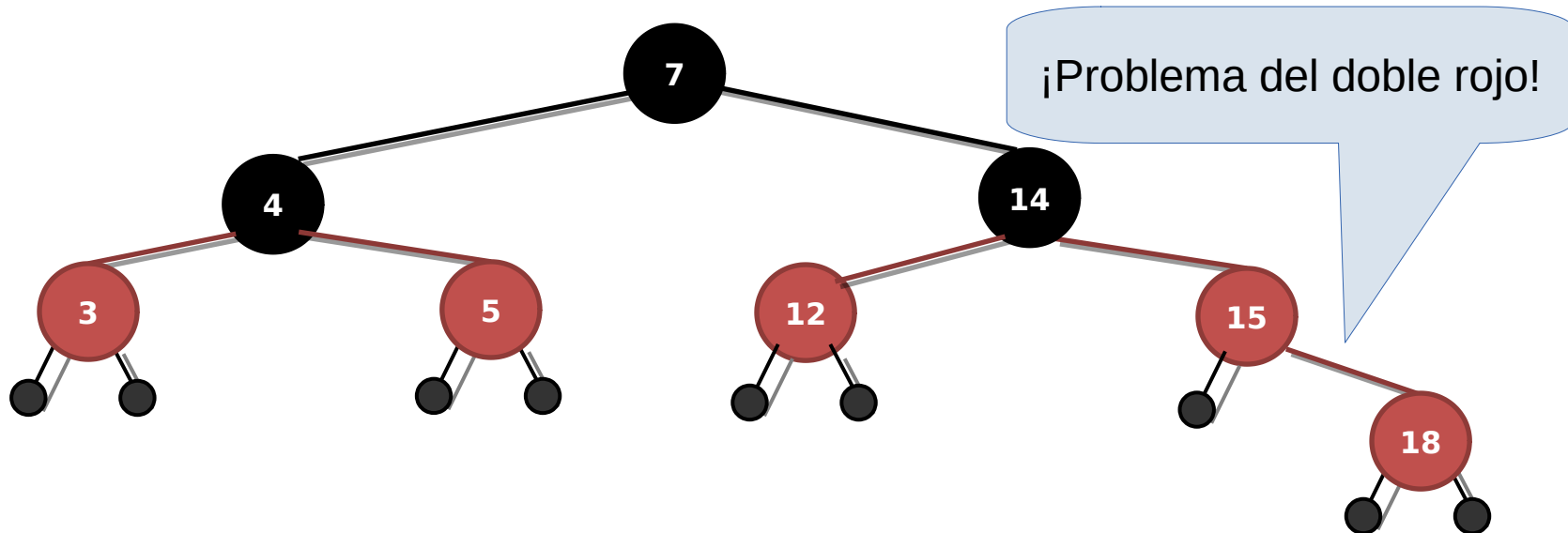
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, **14**, 18, 16, 17



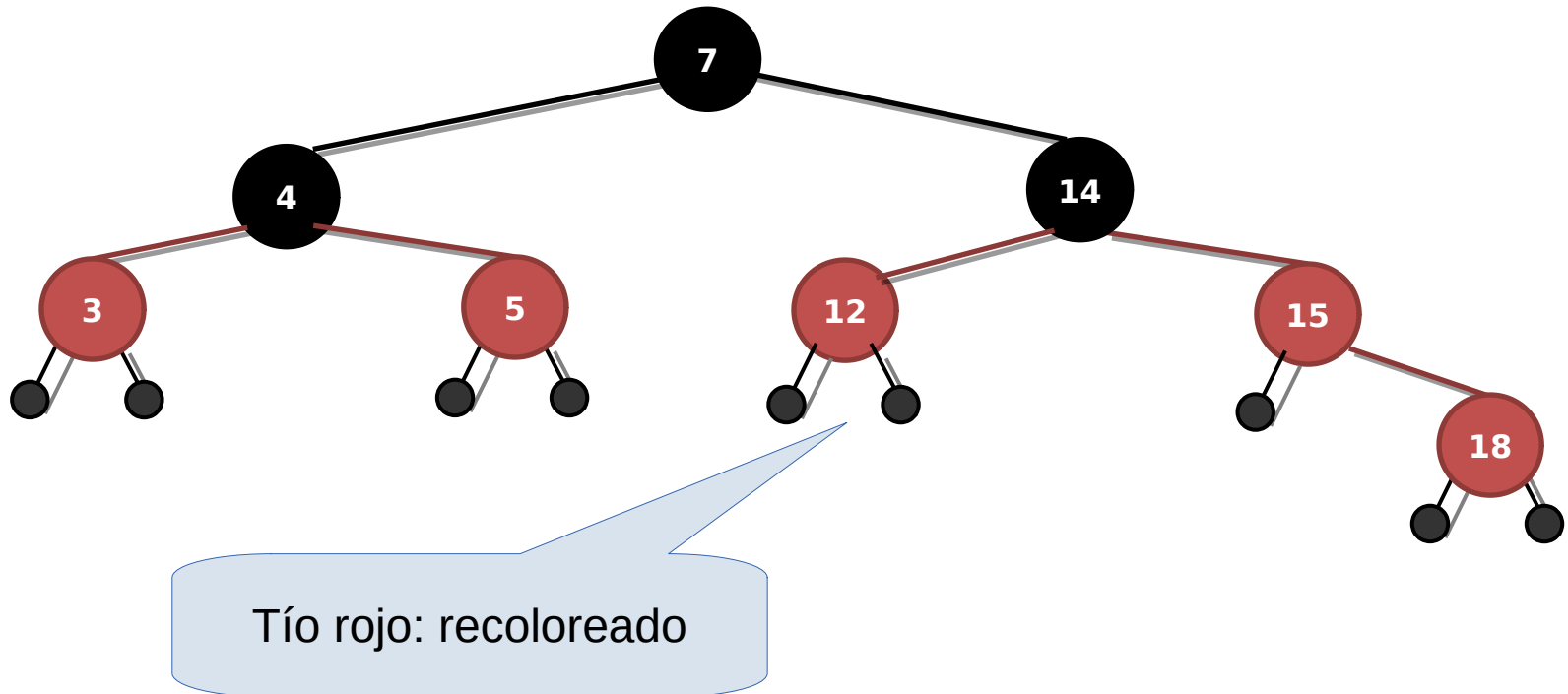
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, **18**, 16, 17



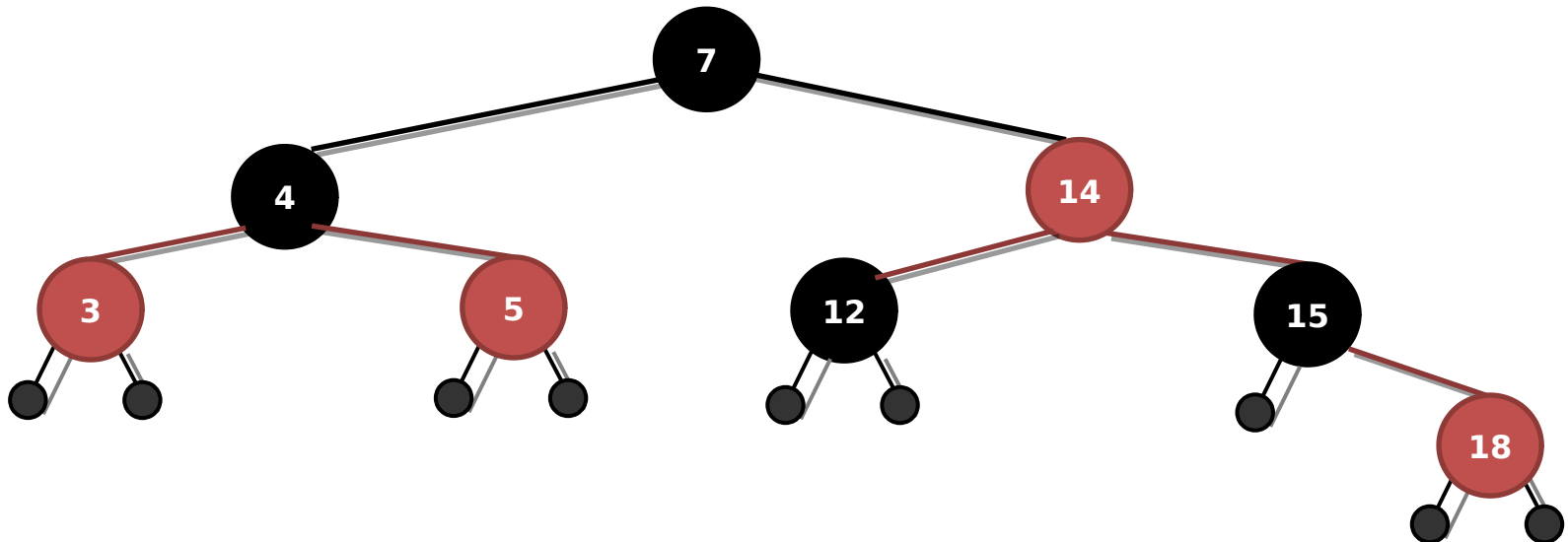
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, **18**, 16, 17



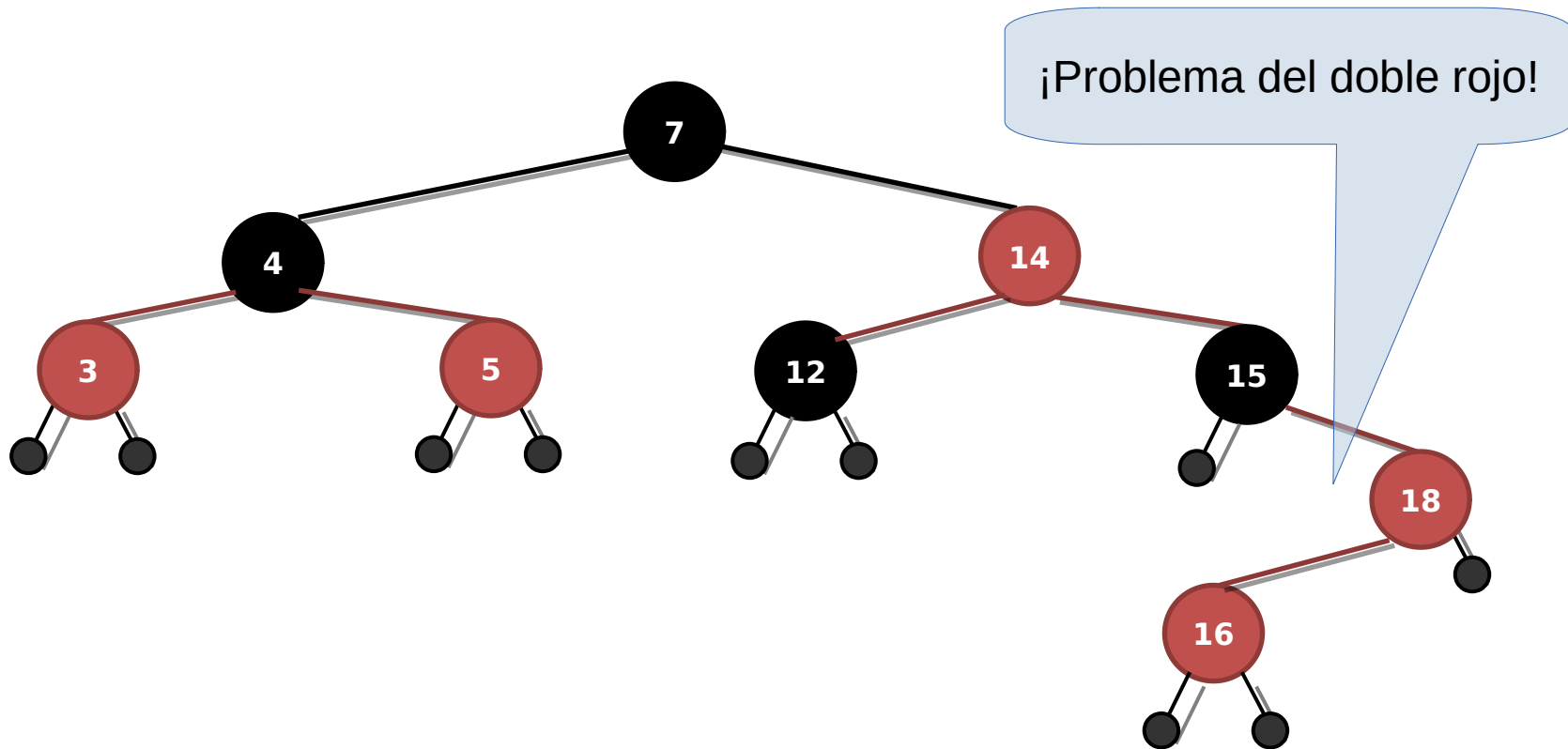
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, **18**, 16, 17



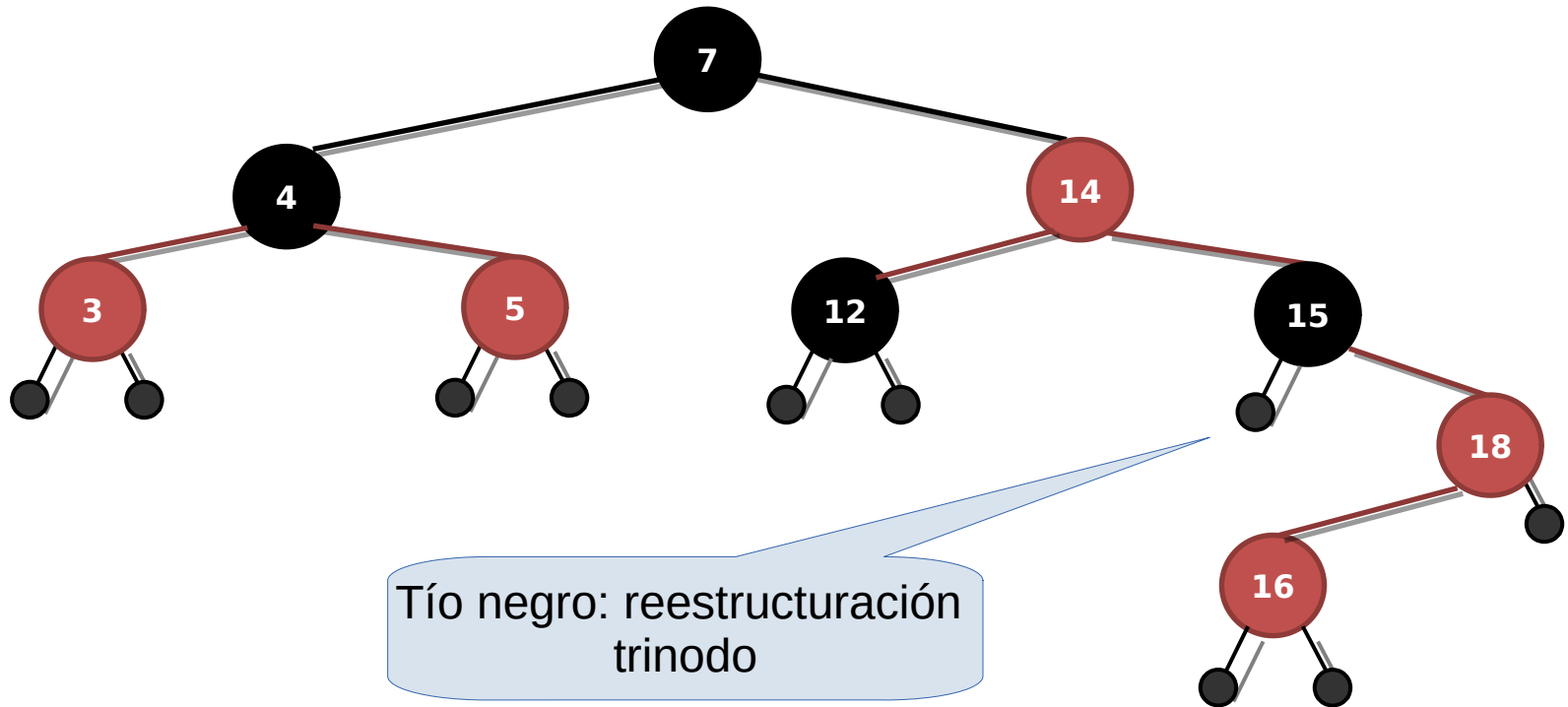
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, **16**, 17



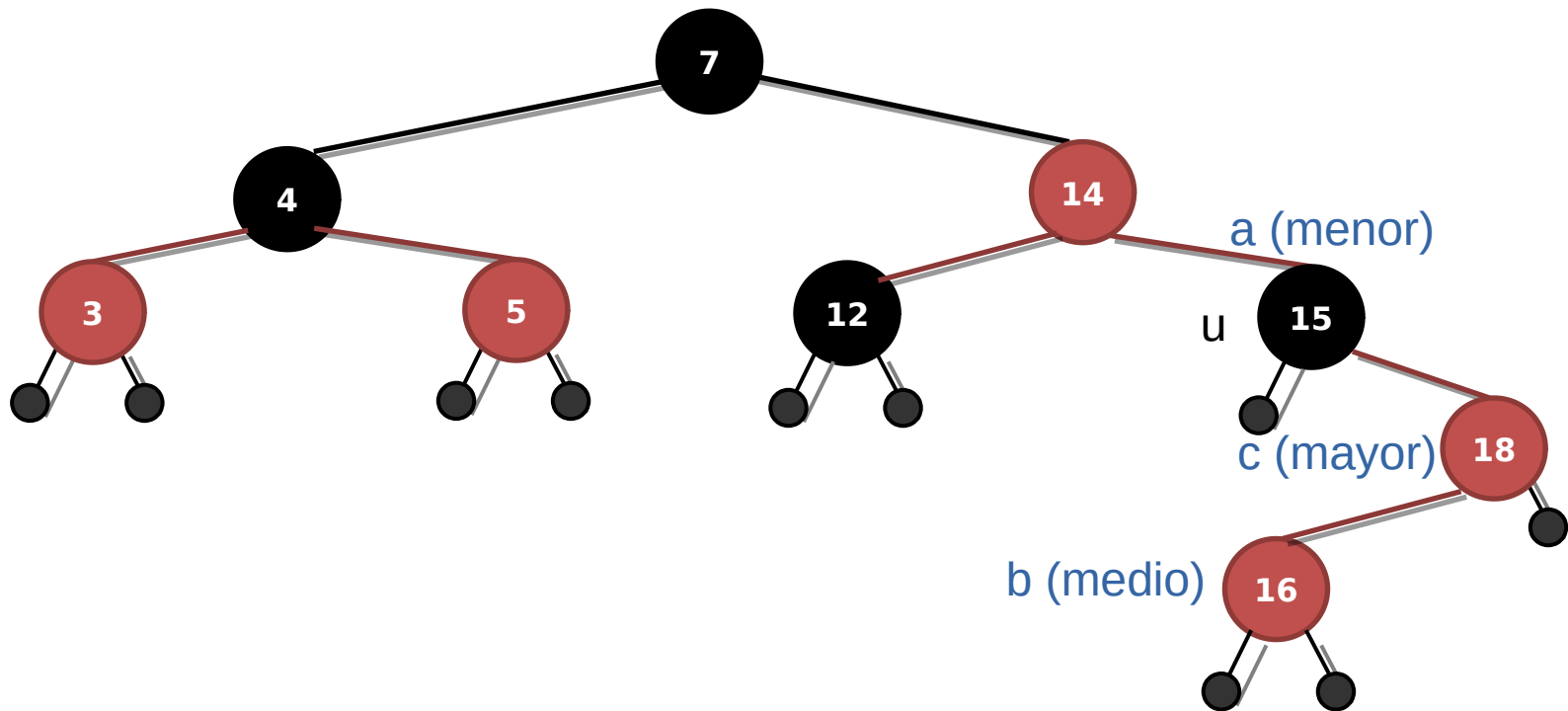
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, **16**, 17



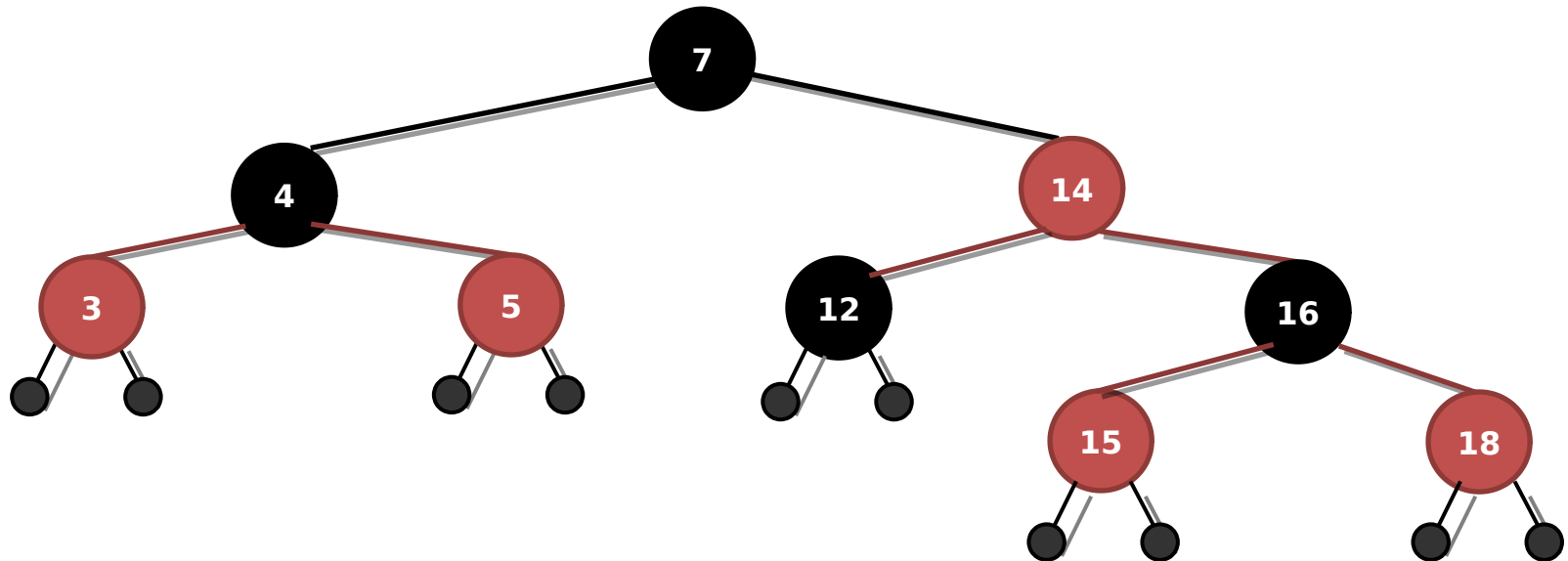
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, **16**, 17



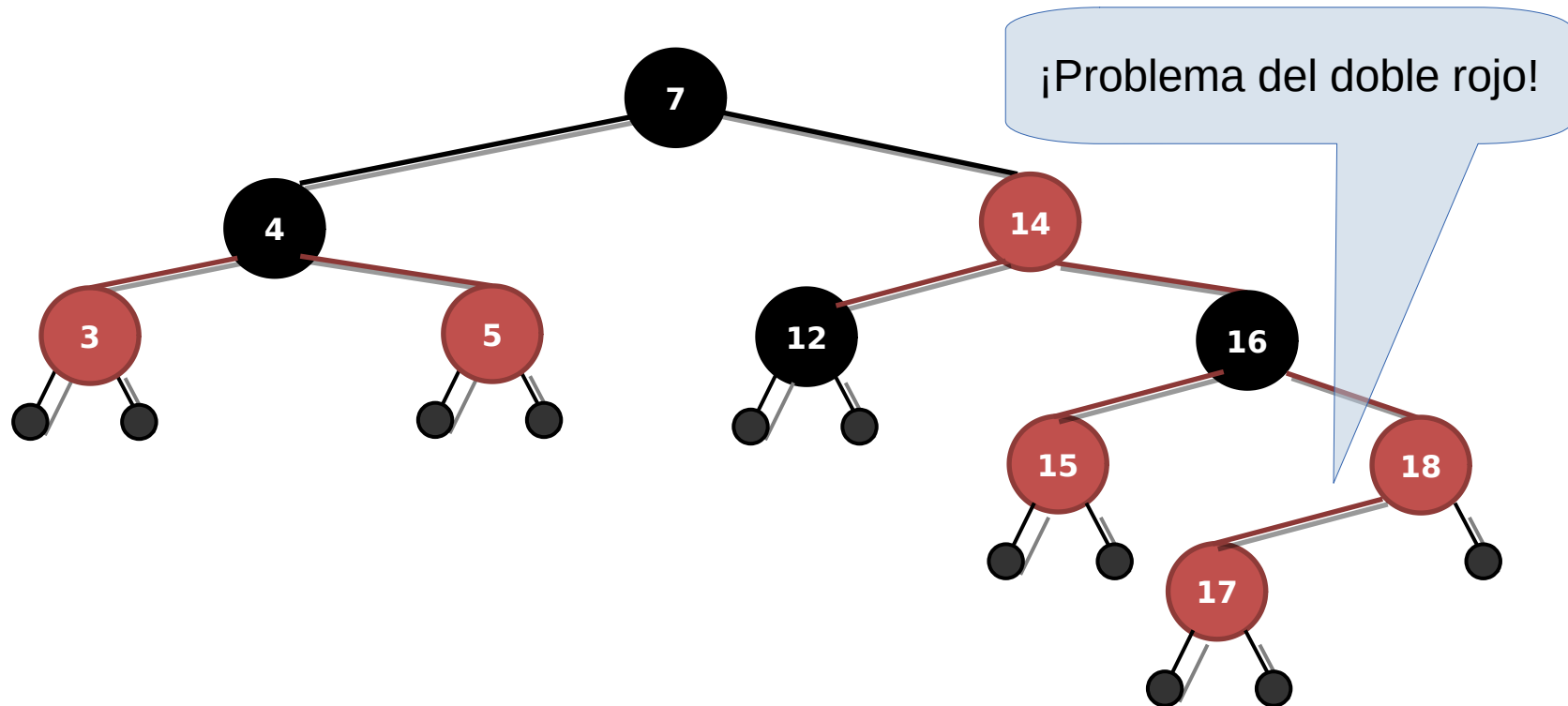
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, **16**, 17



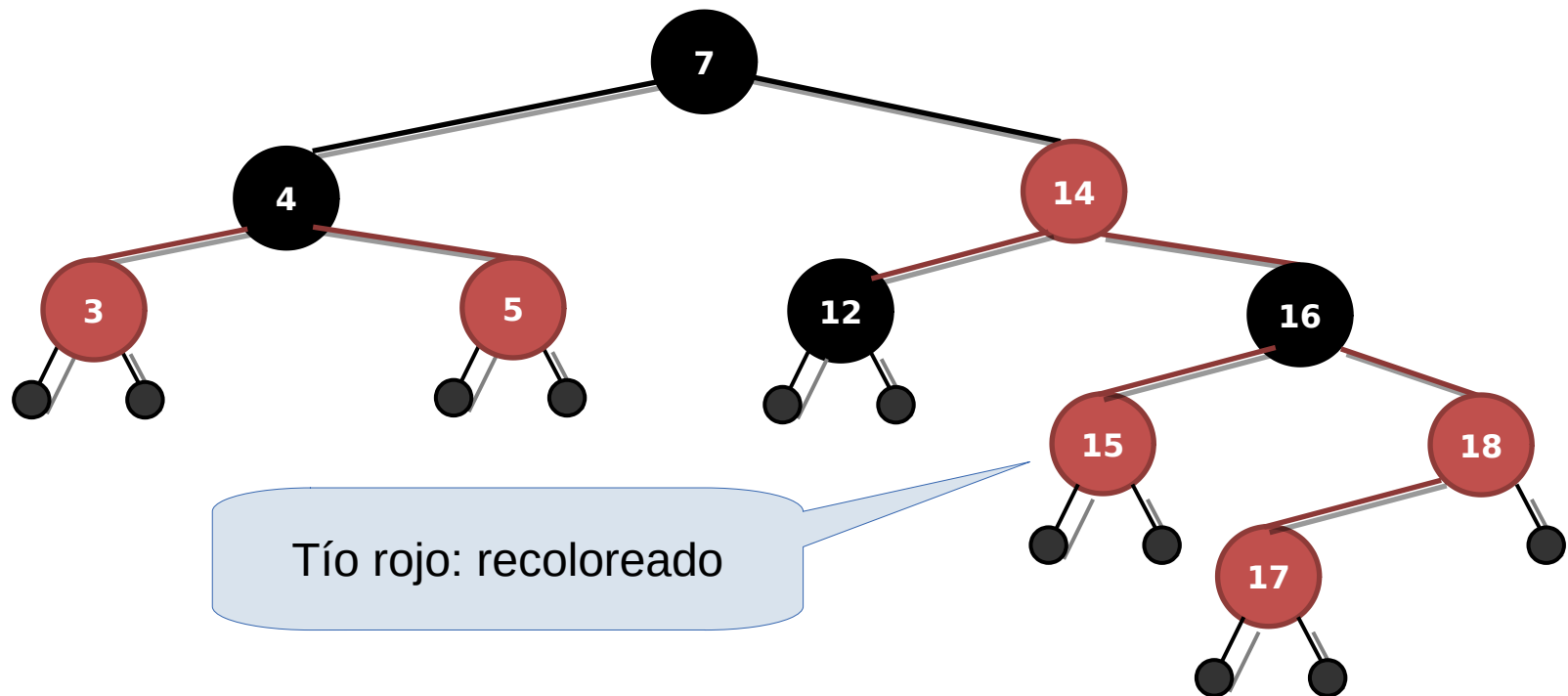
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, **17**



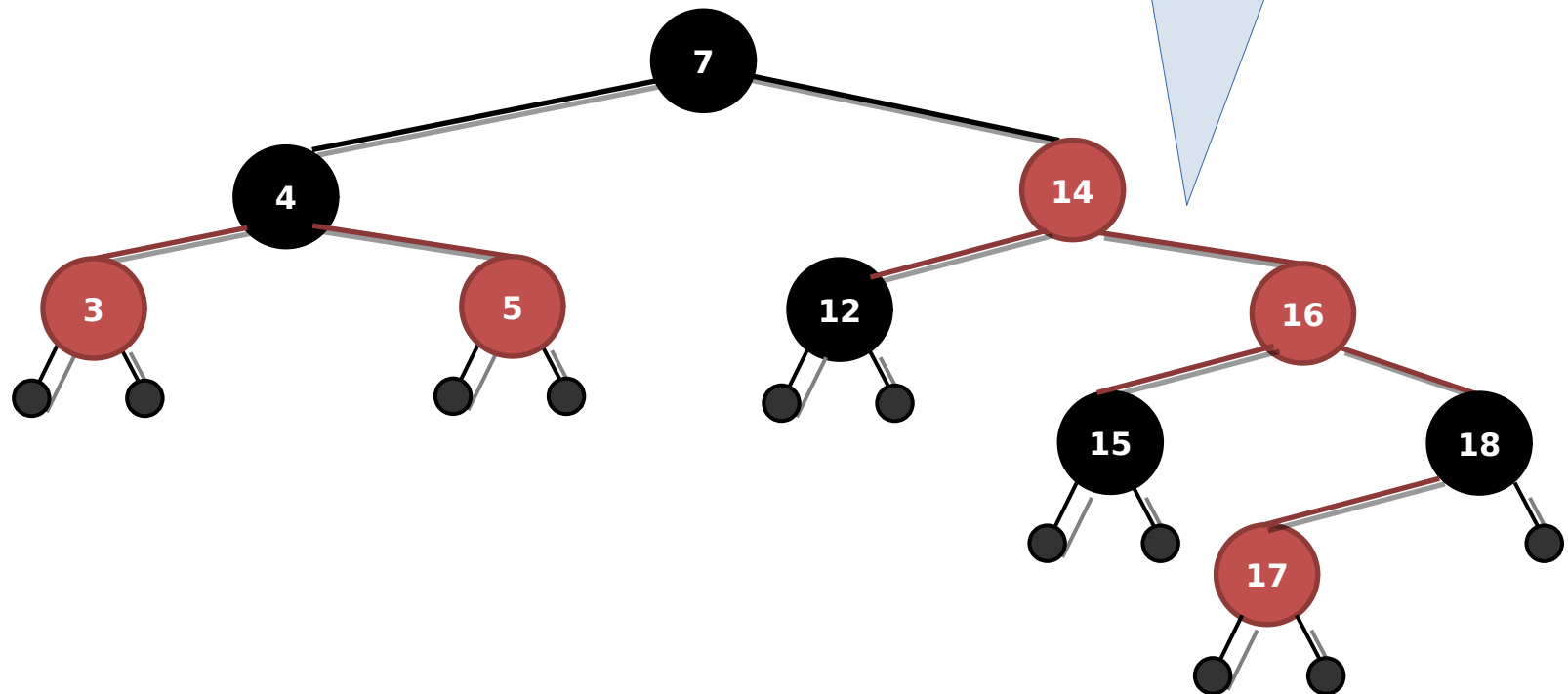
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



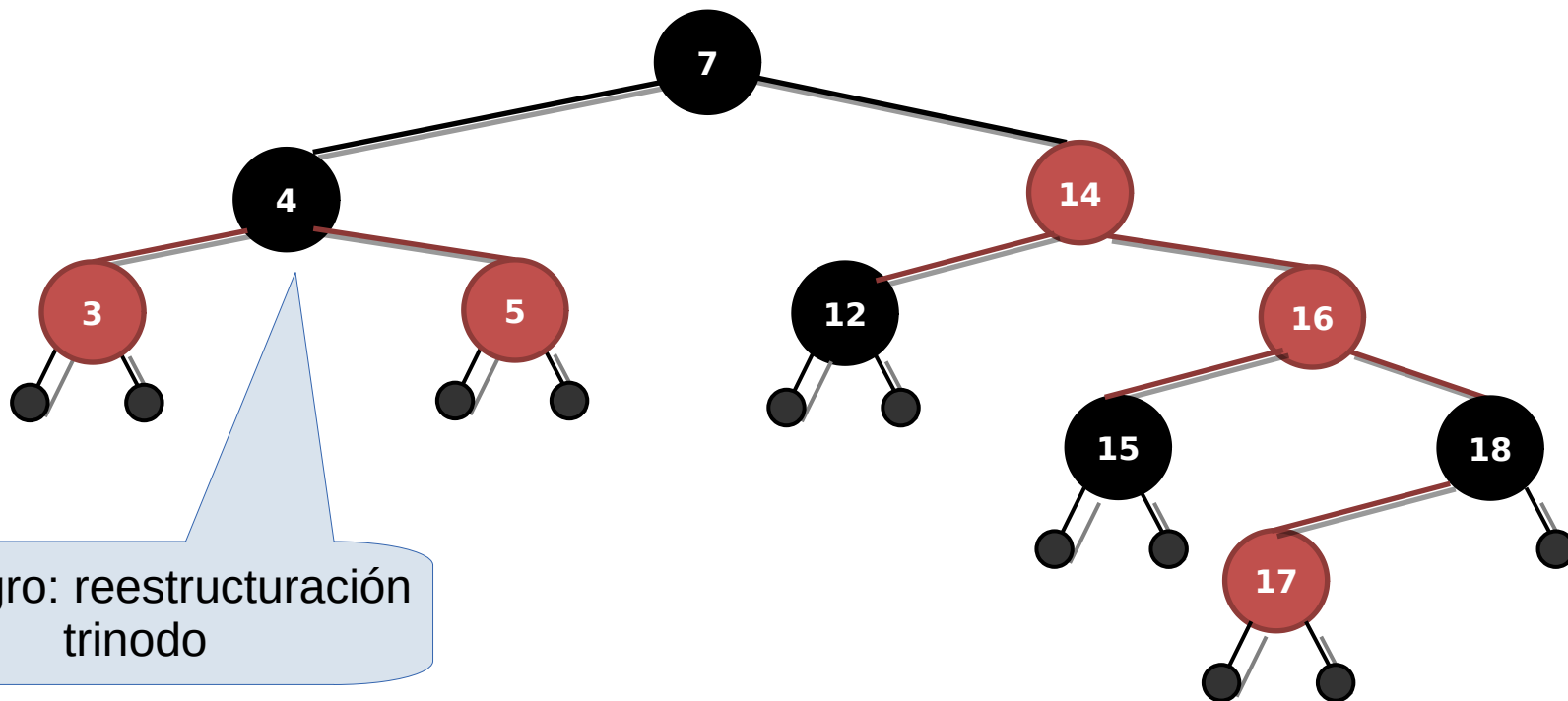
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, **17**



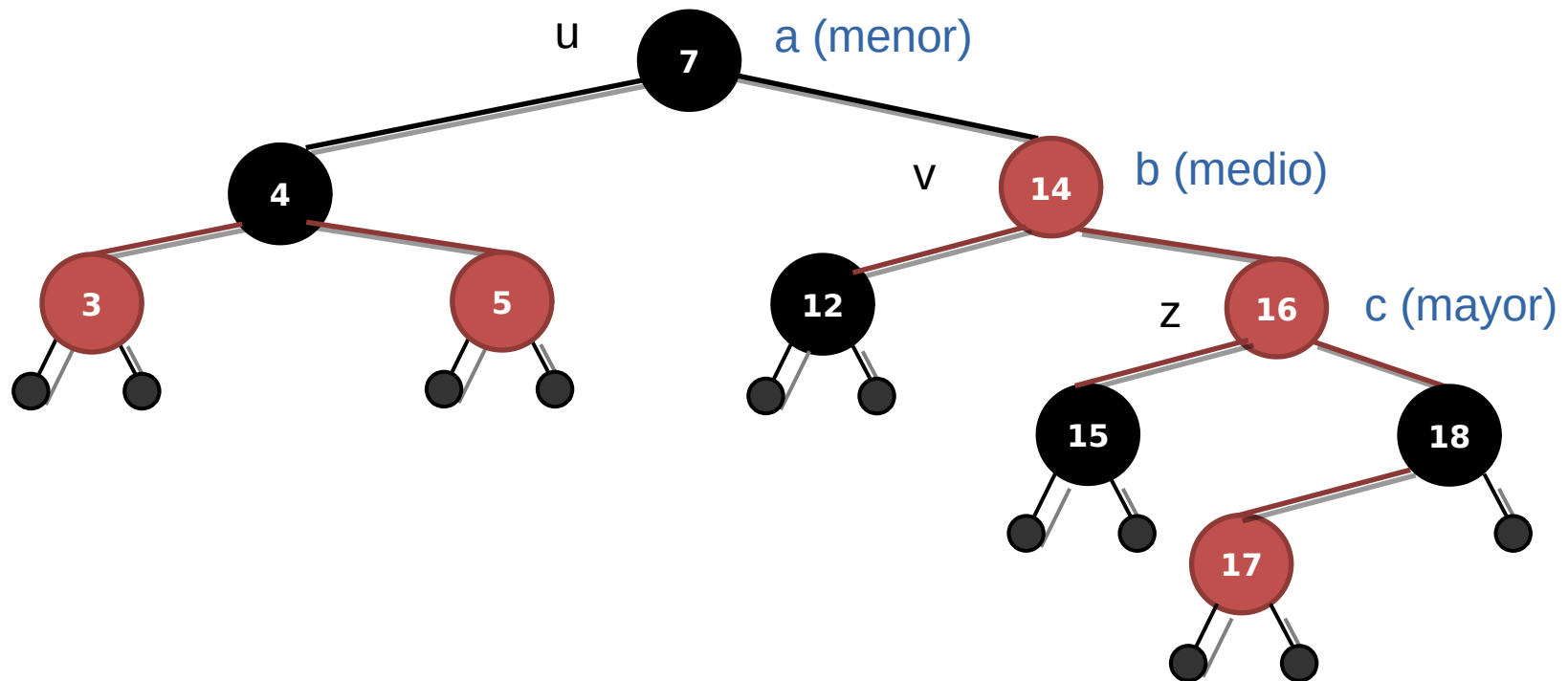
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



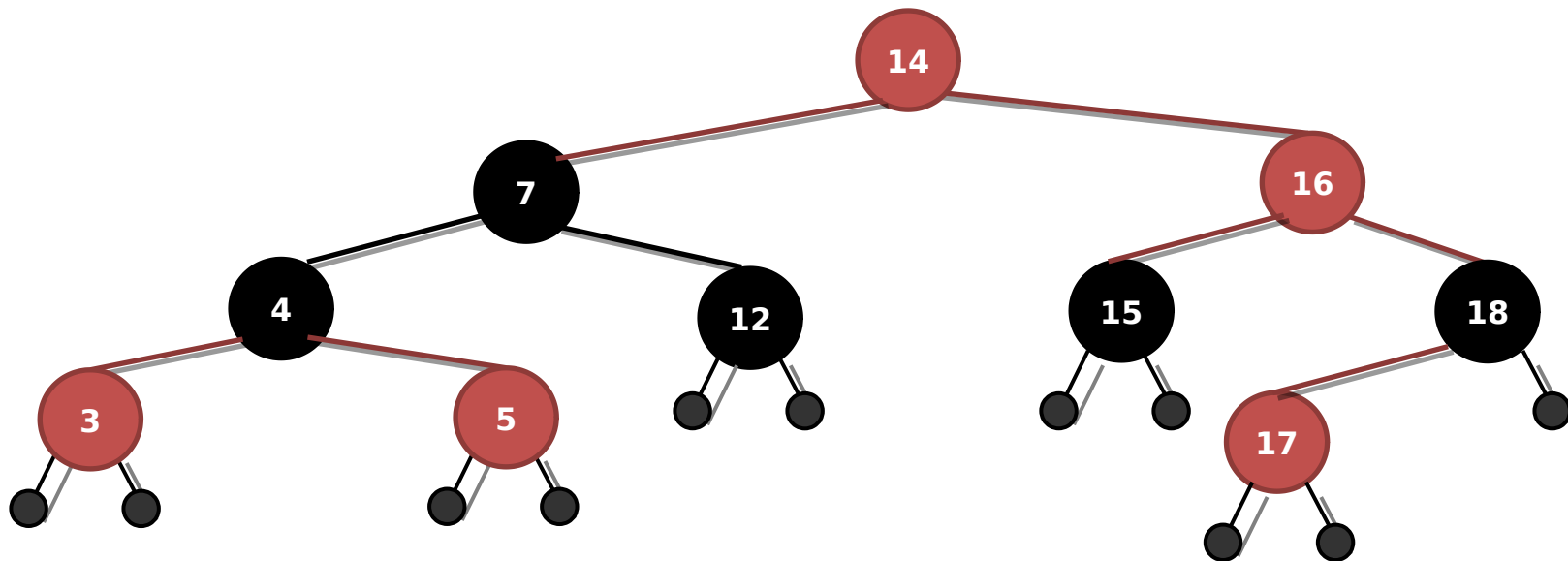
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, **17**



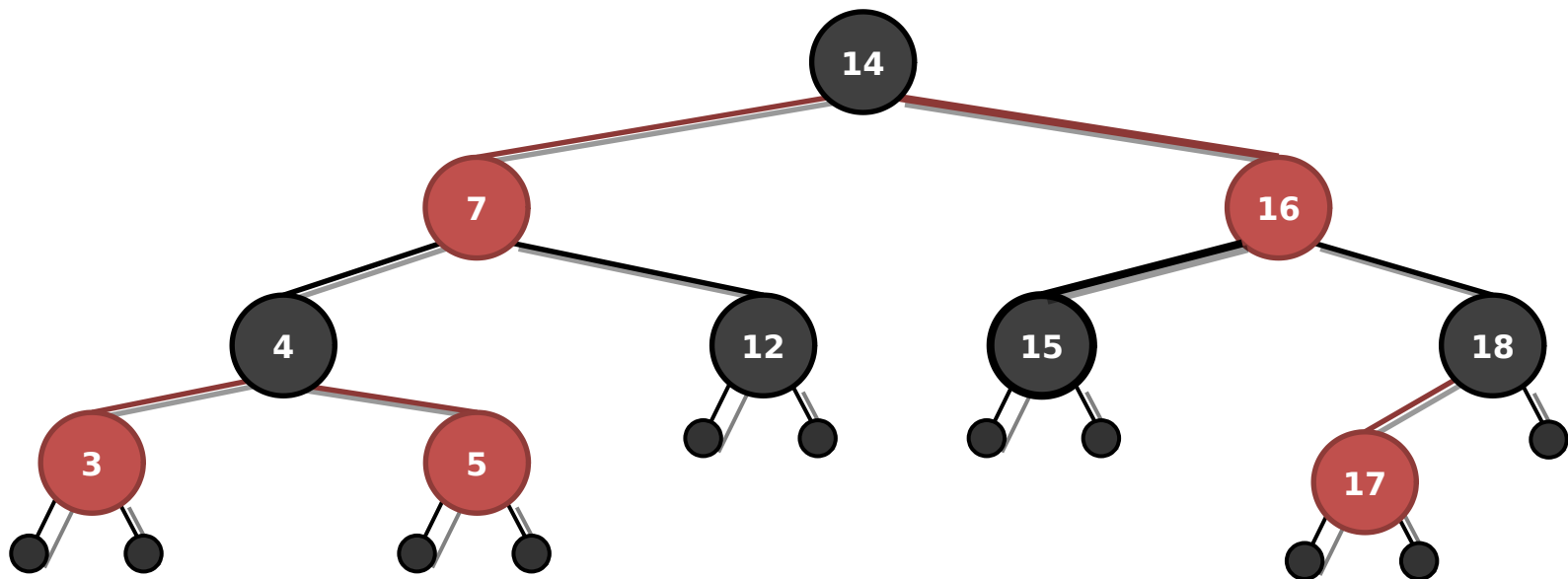
ARN. Ejemplo inserciones

- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



ARN. Ejemplo inserciones

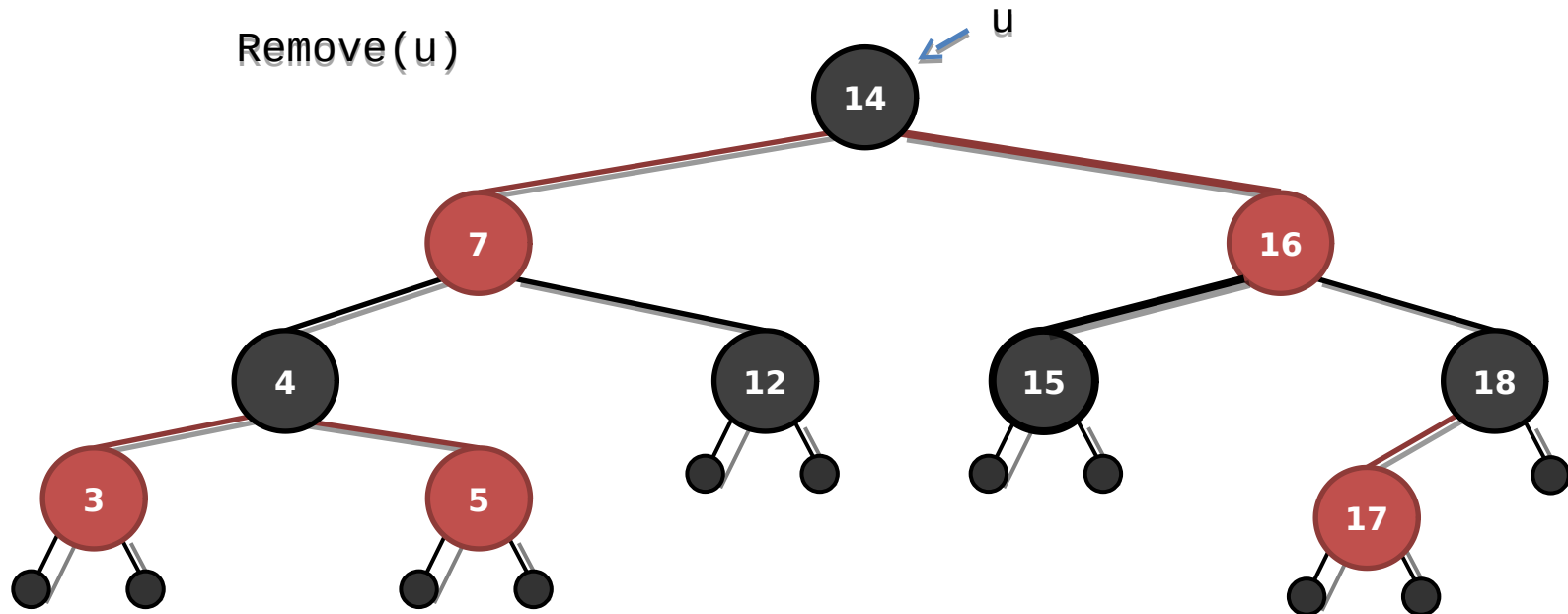
- Generar el árbol rojo-negro resultante tras realizar la siguiente secuencia de inserciones:
 - 4, 7, 12, 15, 3, 5, 14, 18, 16, 17



Árboles Rojo-Negro. Borrado

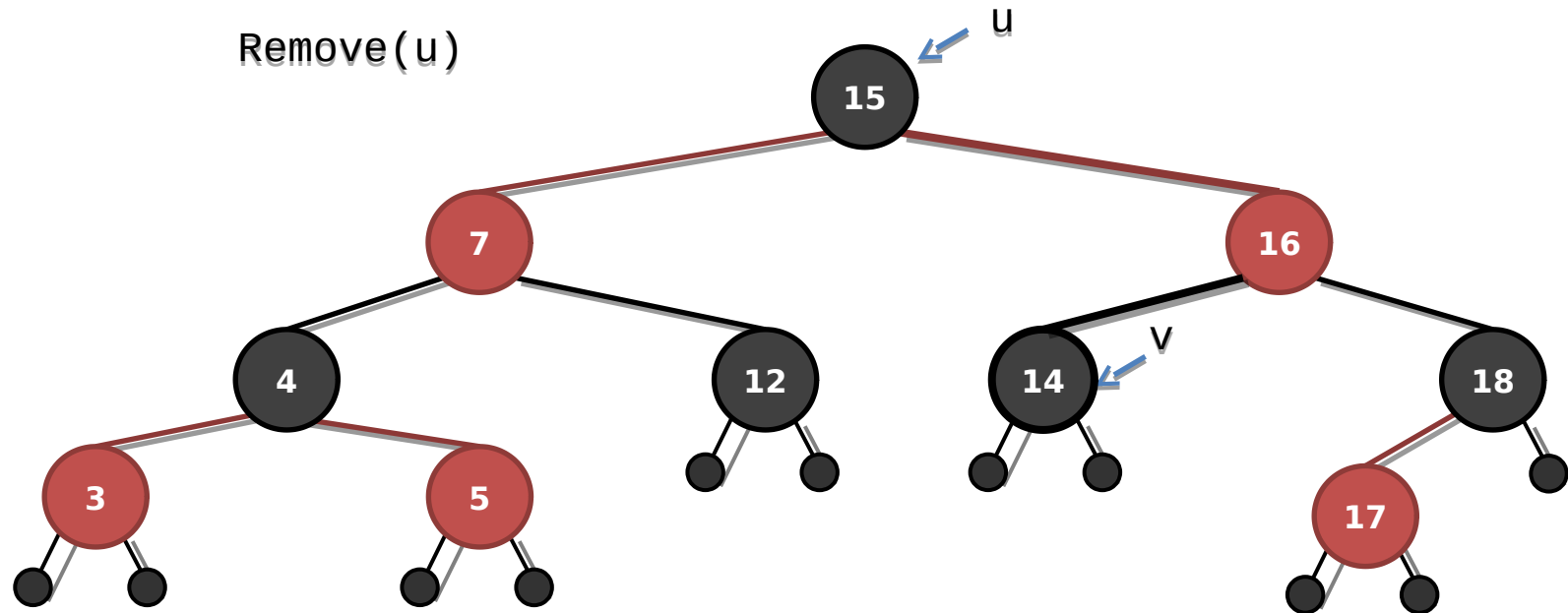
ARN. Borrado

- Inicialmente, **borrar** un valor en un ARN es igual que aplicar una operación de borrado un ABB
 - Recordatorio
 - **Aplicable únicamente si el elemento a borrar tiene un hijo nulo**
 - Si no tiene hijo nulo, se intercambia por su sucesor y entonces se borra



ARN. Borrado

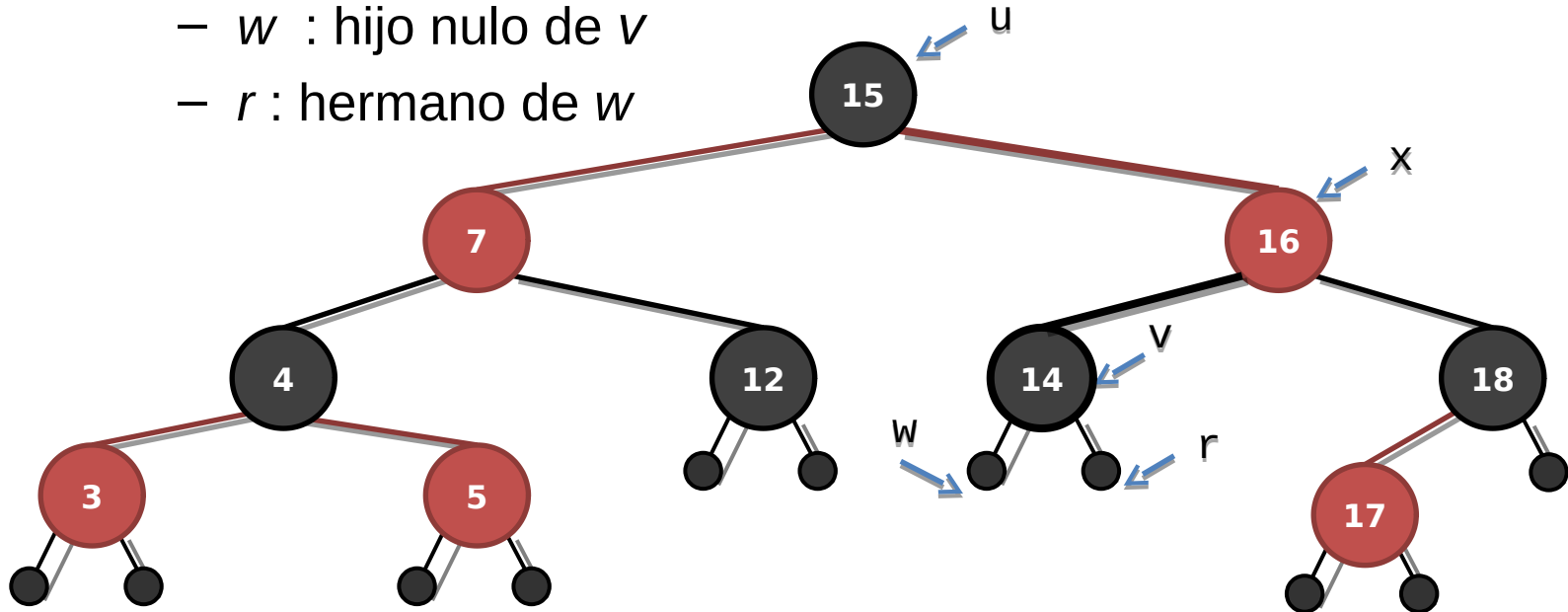
- Inicialmente, **borrar** un valor en un ARN es igual que aplicar una operación de borrado un ABB
 - Recordatorio
 - Aplicable únicamente si el elemento a borrar tiene un hijo nulo
 - **Si no tiene hijo nulo, se intercambia por su sucesor y entonces se borra**



- A continuación se procede al borrado del elemento

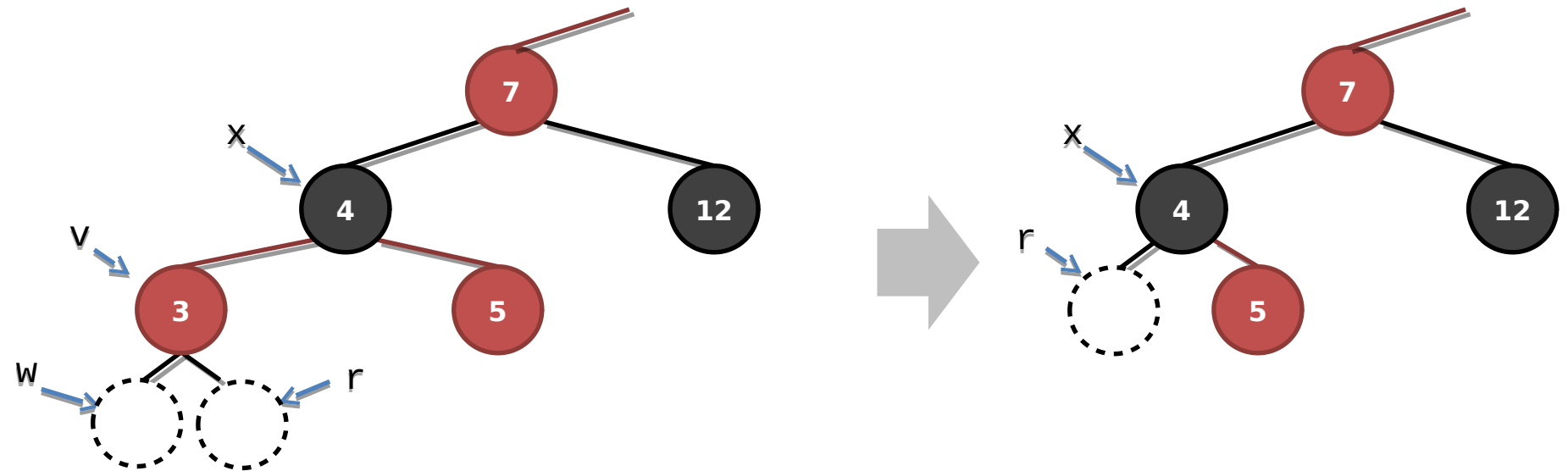
- Notación:

- v : nodo que contiene el elemento a borrar
- x : padre de v
- w : hijo nulo de v
- r : hermano de w

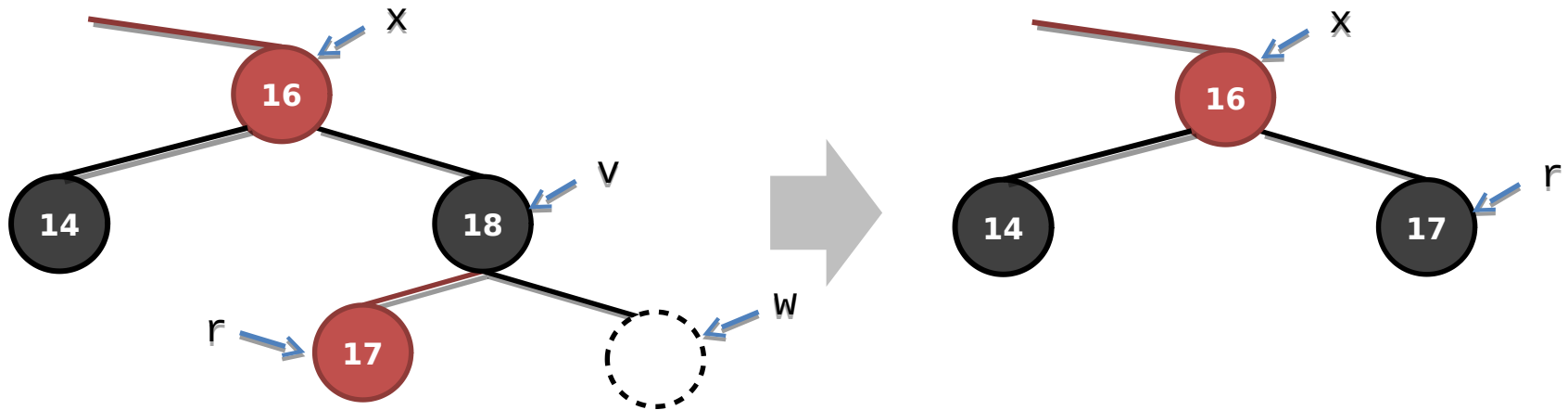


ARN. Borrado

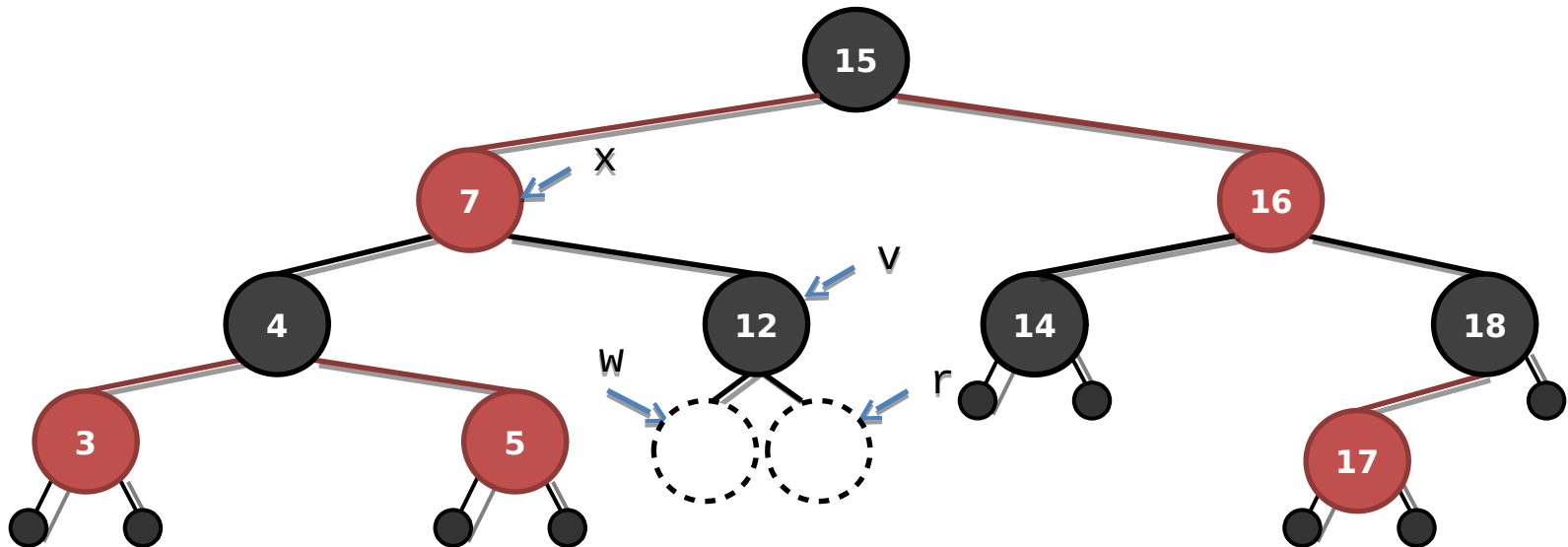
- Procedimiento:
 - Eliminar los nodos v y w . Hacer que r sea hijo de x
 - Si v o r eran **rojos**, r se colorea de negro, quedando el árbol equilibrado



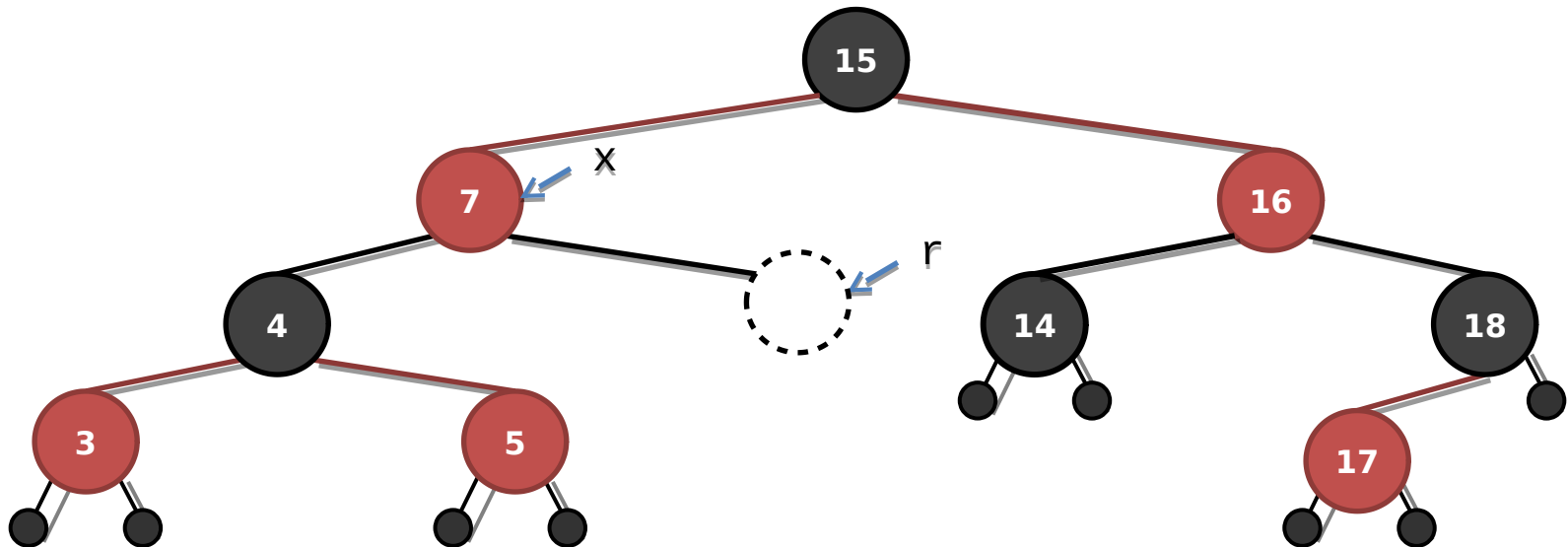
- Procedimiento:
 - Eliminar los nodos v y w . Hacer que r sea hijo de x
 - Si v o r eran **rojos**, r se colorea de negro, quedando el árbol equilibrado



- Procedimiento:
 - Eliminar los nodos v y w . Hacer que r sea hijo de x
 - Si v o r eran rojos, r se colorea de negro, quedando el árbol equilibrado
 - Si v y r eran ambos negros para preservar la propiedad 4 hay que asignar a r un doble negro



- Procedimiento:
 - Eliminar los nodos v y w . Hacer que r sea hijo de x
 - Si v o r eran rojos, r se colorea de negro, quedando el árbol equilibrado
 - Si v y r eran ambos negros para preservar la propiedad 4 hay que asignar a r **un doble negro**

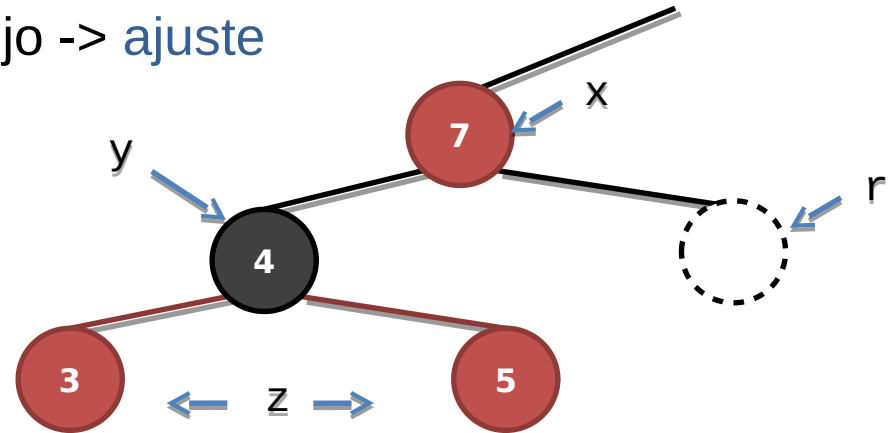


ARN. Resolver doble negro

- 3 posibles situaciones – 3 posibles soluciones
 - *Caso 1*: el hermano de r es negro y uno de sus hijos (sobrino de r) es rojo -> **reestructuración**
 - *Caso 2*: el hermano de r es negro y sus dos hijos (sobrinos de r) son negros -> **recoloración**
 - *Caso 3*: el hermano de r es rojo -> **ajuste**

- Notación:

- r : nodo doble negro
- x : padre de r
- y : hermano de r
- z : sobrino de r (hijo de y)

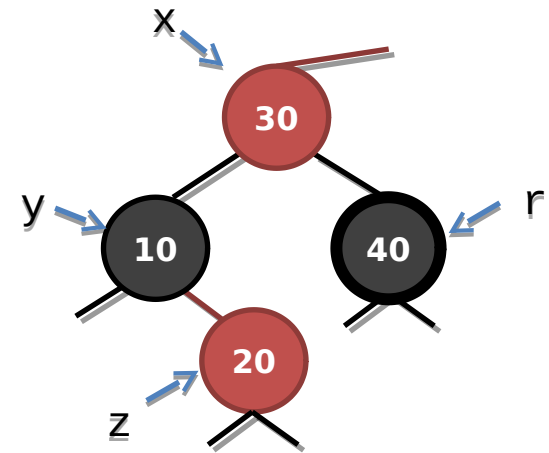


ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo



Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.



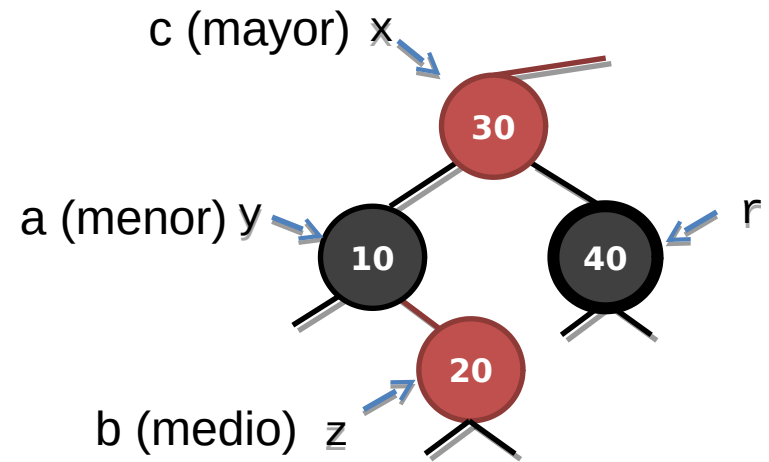
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- **Aplicar reestructuración trinodo en el nodo:** identificar de los nodos y , z y x quien es el que mayor valor tiene, el valor medio y el valor menor



ARN. Resolver doble negro (Caso 1)

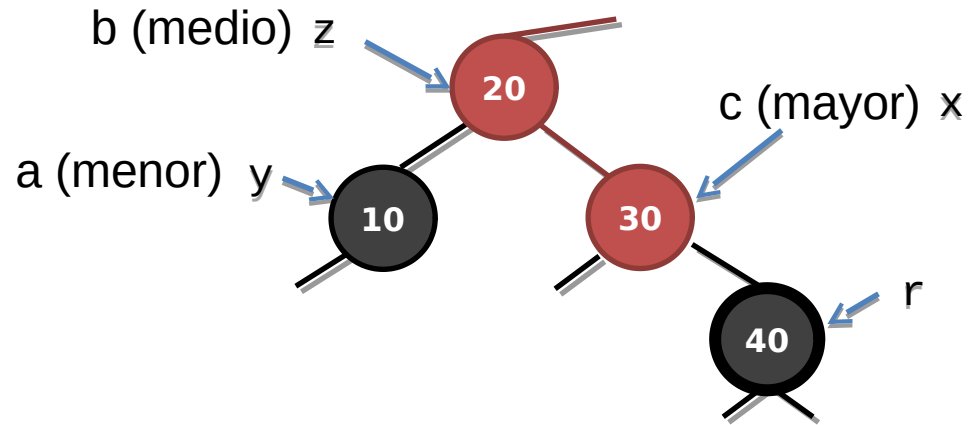
Caso 1: el hermano y de r es negro y un sobrino z es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- **Aplicar restructuración**

trinodo en el nodo: identificar de los nodos y , z y x quien es el que mayor valor tiene, el valor medio y el valor menor



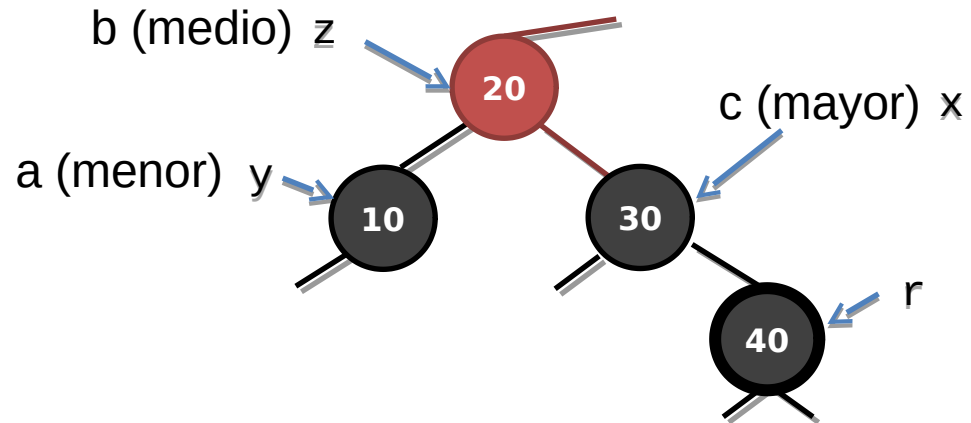
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano **y** de **r** es negro y un sobrino **z** es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- Aplicar reestructuración trinodo en el nodo: identificar de los nodos **y**, **z** y **x** quien es el que mayor valor tiene, el valor medio y el valor menor
- Colorear **a** y **c** de negro



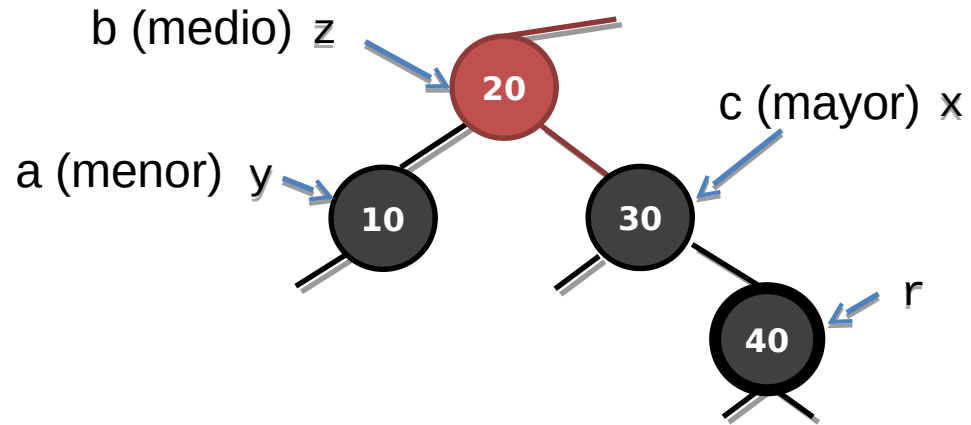
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano **y** de **r** es negro y un sobrino **z** es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- Aplicar restructuración trinodo en el nodo: identificar de los nodos y, z y x quien es el que mayor valor tiene, el valor medio y el valor menor
- Colorear a y C de negro
- Asignar a **b** el color de que tenía **x**



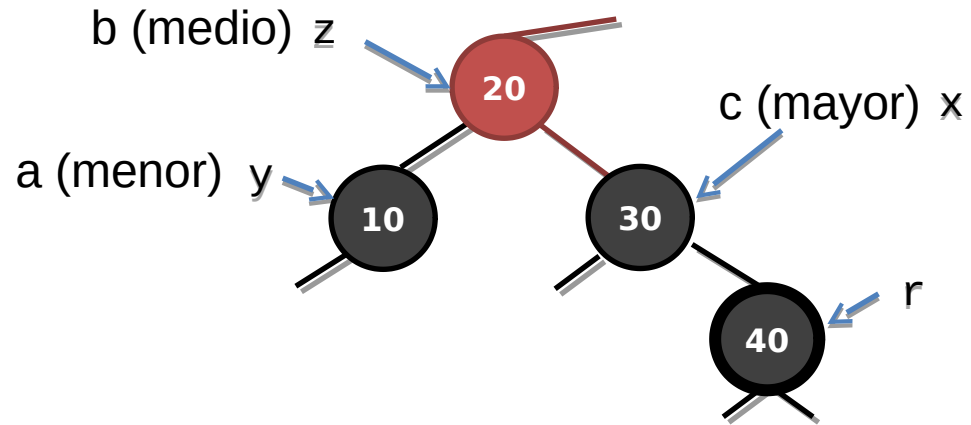
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano **y** de **r** es negro y un sobrino **z** es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- Aplicar reestructuración trinodo en el nodo: identificar de los nodos y, z y x quien es el que mayor valor tiene, el valor medio y el valor menor
- Colorear a y C de negro
- Asignar a b el color de que tenía X
- Colorear **r** de negro



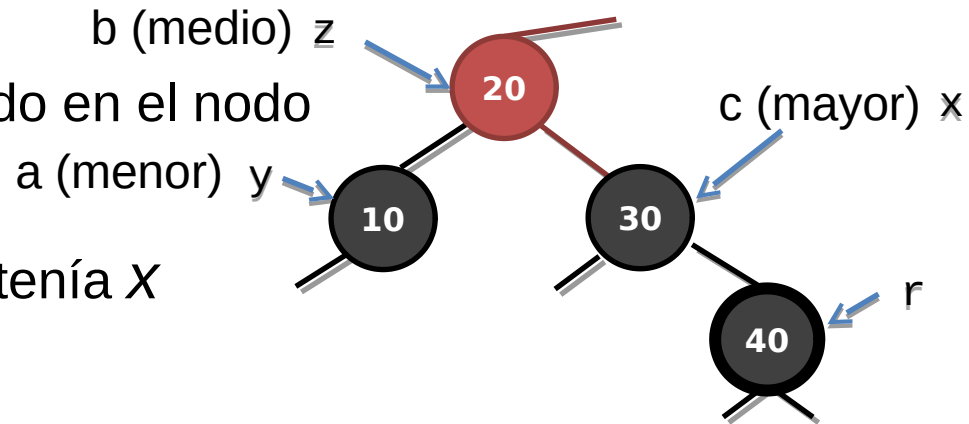
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo

Convertir el sobrino rojo en negro y traerlo encima del nodo doble negro para resolver el problema.

Para ello:

- Aplicar restructuración trinodo en el nodo
- Colorear a y c de negro
- Asignar a b el color de que tenía X
- Colorear r de negro

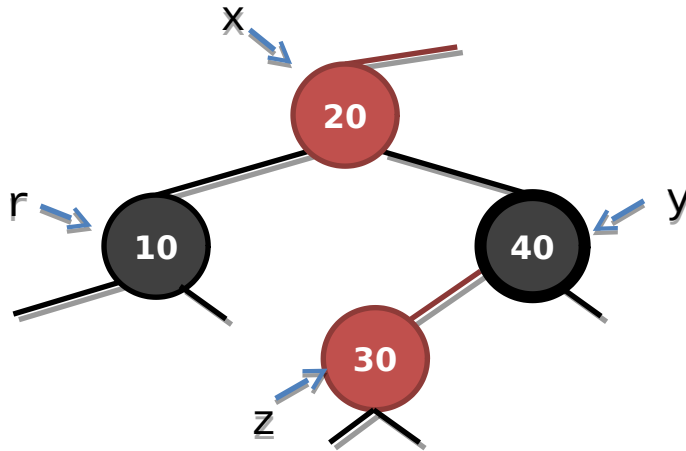


Tras la restructuración y el coloreado el árbol queda equilibrado

- Como mucho se ejecuta una operación de restructuración

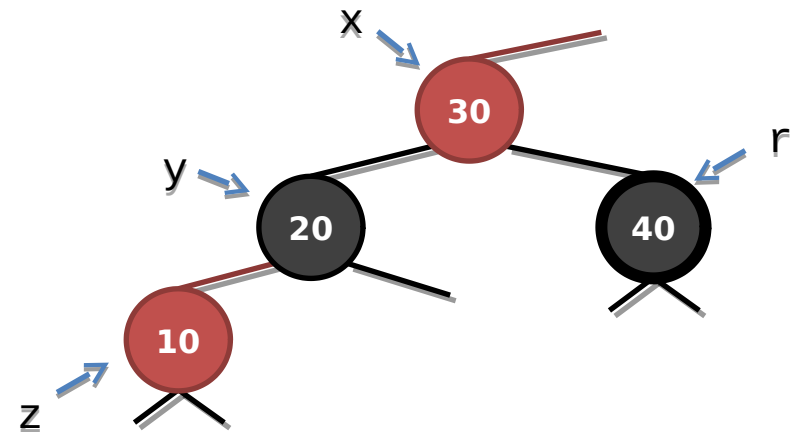
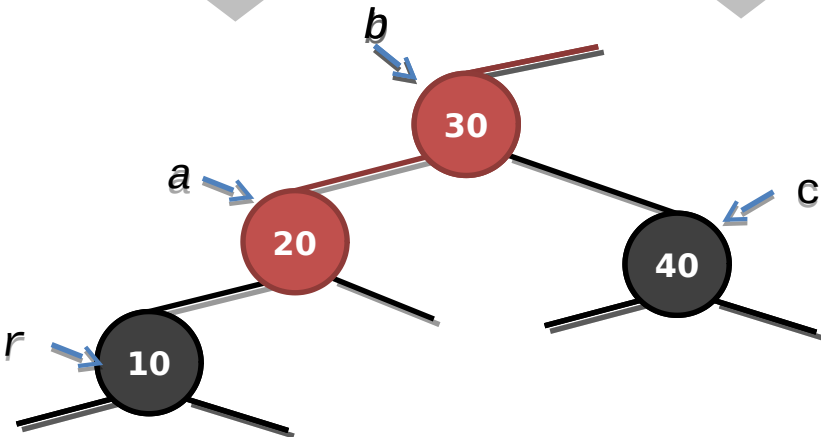
ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo



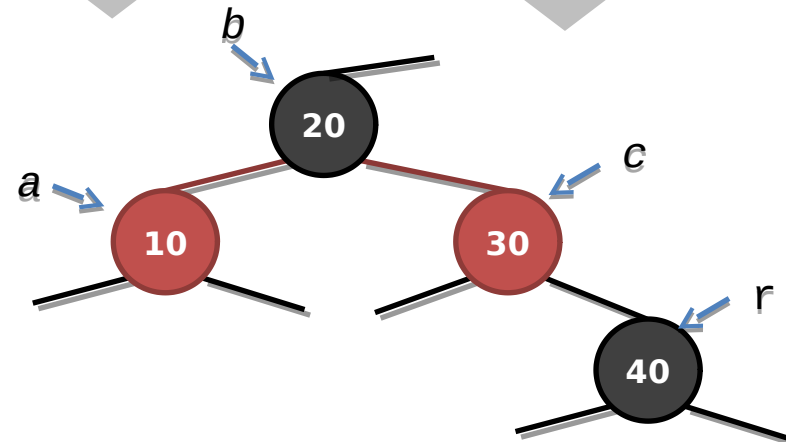
RESTRUCTURACIÓN
TRINODO

RESTRUCTURACIÓN
TRINODO



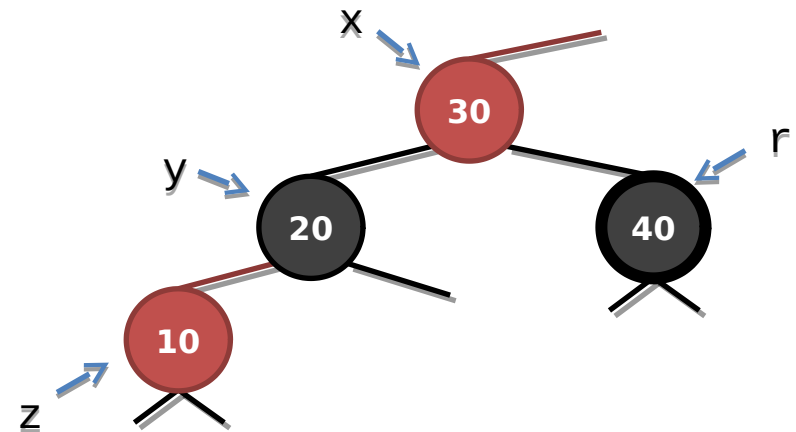
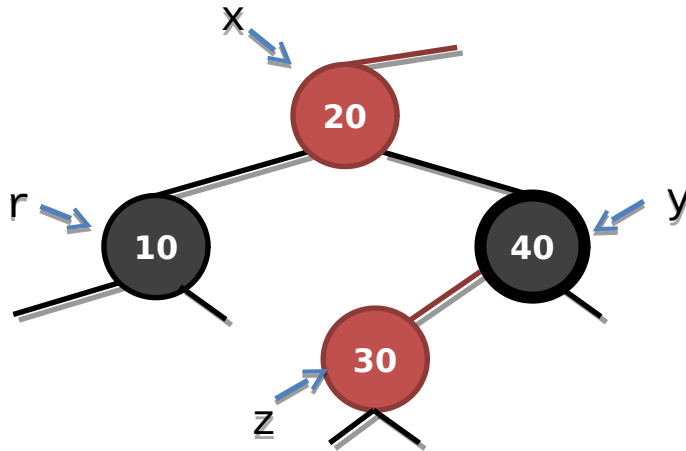
RESTRUCTURACIÓN
TRINODO

RESTRUCTURACIÓN
TRINODO

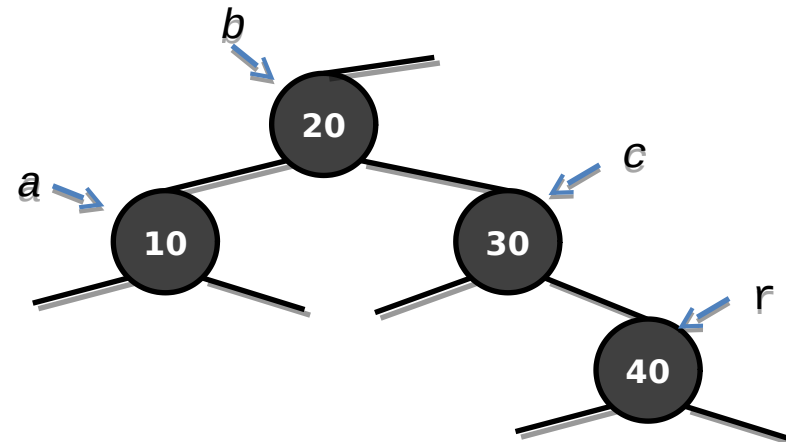
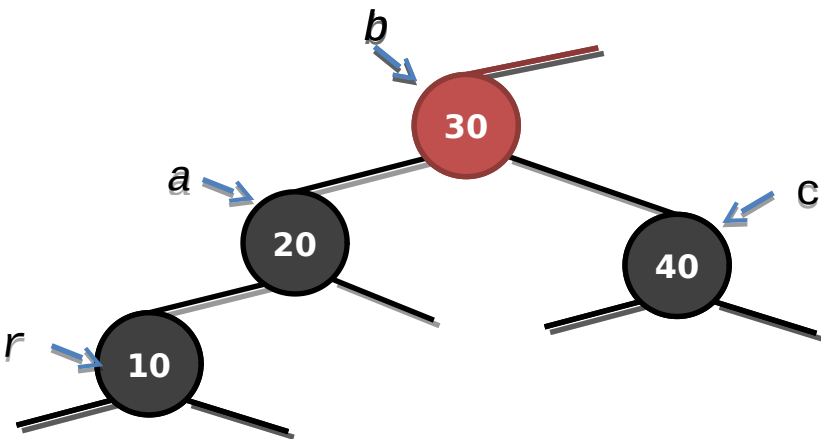


ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo

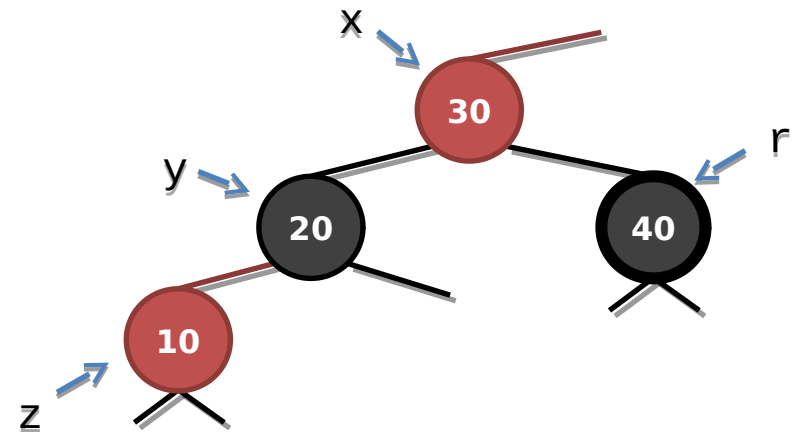
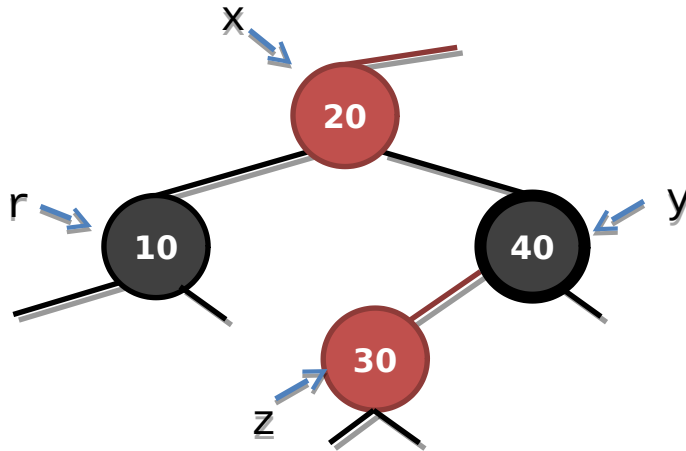


Colorear a y c de negro, asignar a b el color de que tenía x, colorear r de negro

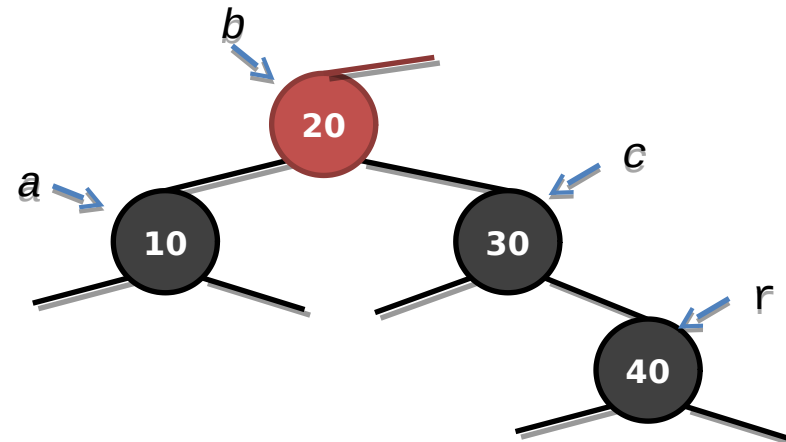
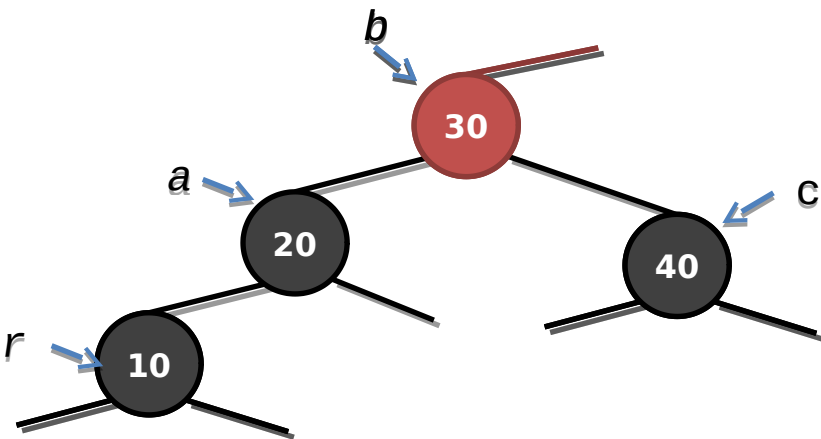


ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo

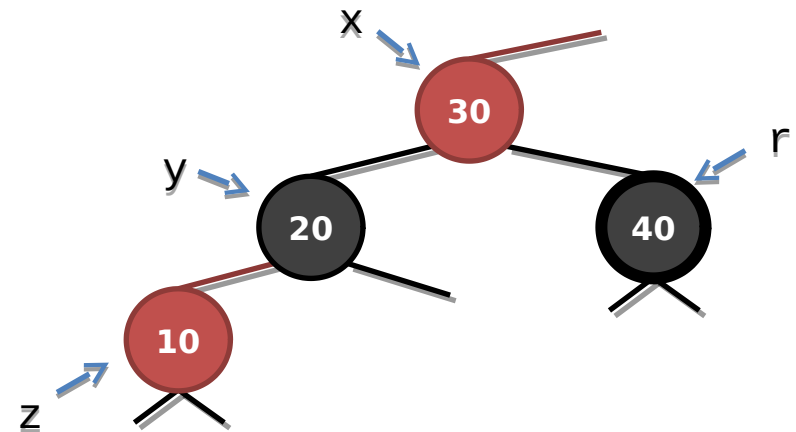
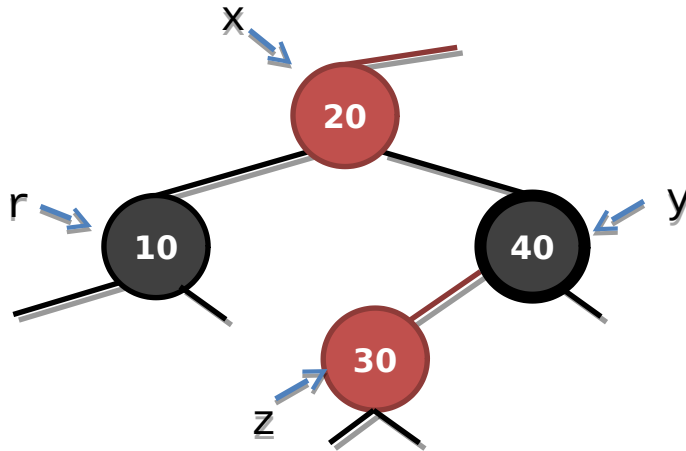


Colorear a y c de negro, asignar a b el color de que tenía x , colorear r de negro

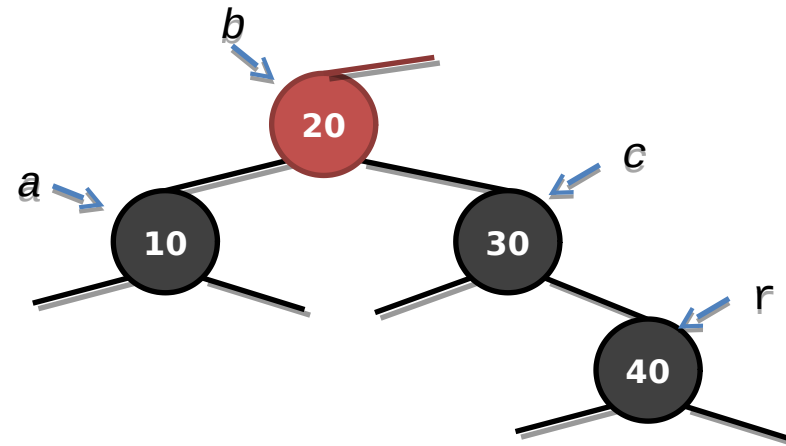
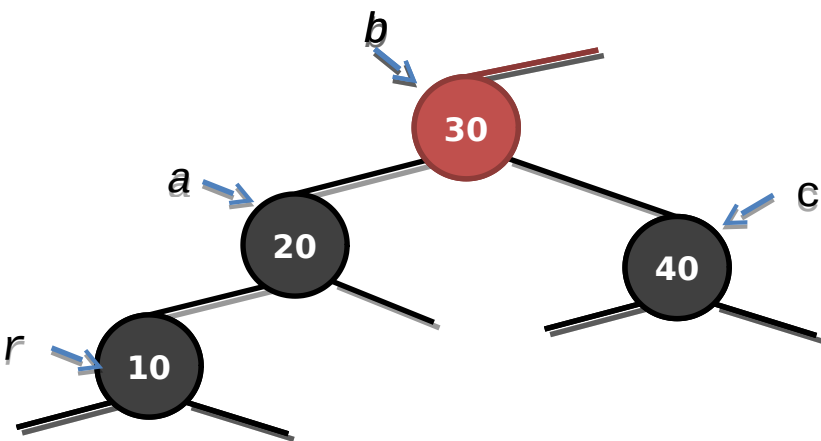


ARN. Resolver doble negro (Caso 1)

Caso 1: el hermano y de r es negro y un sobrino z es rojo



Colorear a y c de negro, asignar a b el color de que tenía x , **colorear r de negro**

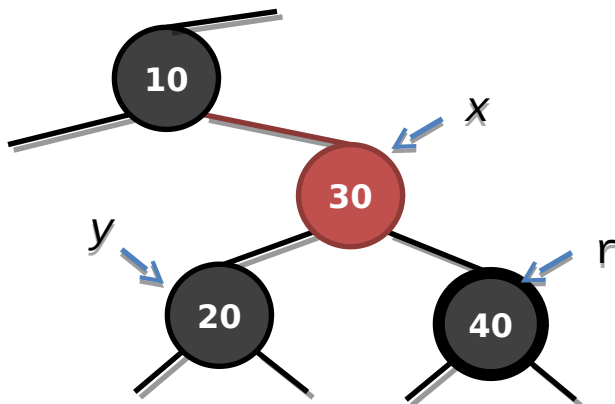


ARN. Resolver doble negro (Caso 2)

Caso 2: el hermano y de r es negro y sus dos hijos (sobrinos de r) son negros



Convertir a negro el padre de r mientras y se pasa a rojo mantiene la altura negra y soluciona el doble negro.



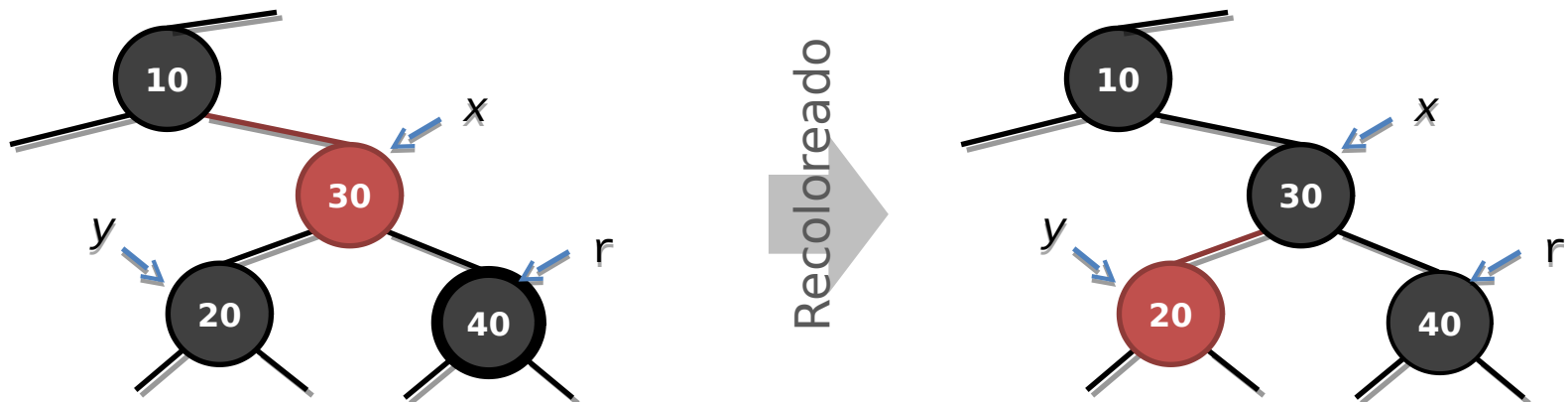
ARN. Resolver doble negro (Caso 2)

Caso 2: el hermano y de r es negro y sus dos hijos (sobrinos de r) son negros

Convertir a negro el padre de r mientras y se pasa a rojo mantiene la altura negra y soluciona el doble negro.

Para ello:

- Recoloración de r como negro e y como rojo
- Si x es rojo, se pasa a negro

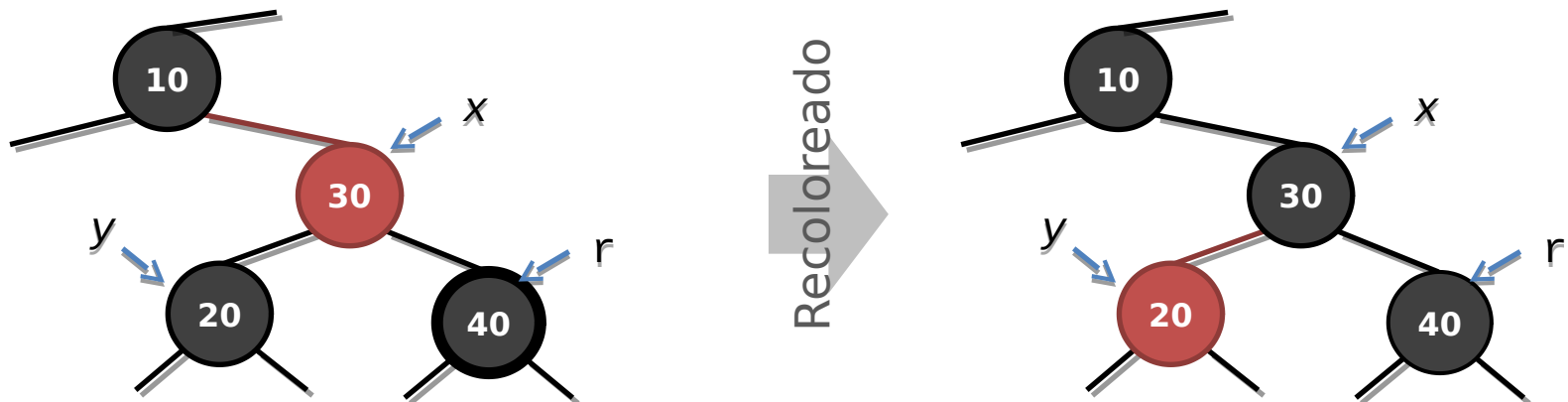


ARN. Resolver doble negro (Caso 2)

Caso 2: el hermano y de r es negro y sus dos hijos (sobrinos de r) son negros

Convertir a negro el padre de r mientras y se pasa a rojo mantiene la altura negra y soluciona el doble negro.

Si x no era rojo, se ha trasladado el problema a x . En ese caso se vuelve a aplicar el caso que corresponda (Caso 1,2 o 3).



ARN. Resolver doble negro (Caso 2)

Caso 2: el hermano y de r es negro y sus dos hijos (sobrinos de r) son negros

Convertir a negro el padre de r mientras y se pasa a rojo mantiene la altura negra y soluciona el doble negro.

Si x no era rojo, se ha trasladado el problema a x . En ese caso se vuelve a aplicar el caso que corresponda (Caso 1,2 o 3).

Observaciones:

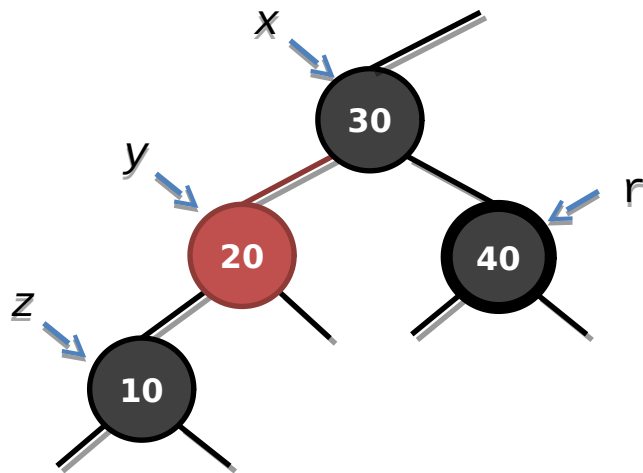
- Número máximo de recoloreados causado por un borrado no es mayor que $\log(n+1)$
- El proceso termina forzosamente porque en la raíz no tiene sentido el problema de doble negro.

ARN. Resolver doble negro (Caso 3)

Caso 3: el hermano y de r es rojo



Traer el nodo rojo como padre de r para que el hermano sea un nodo negro y se transforme a caso 1 o caso 2.



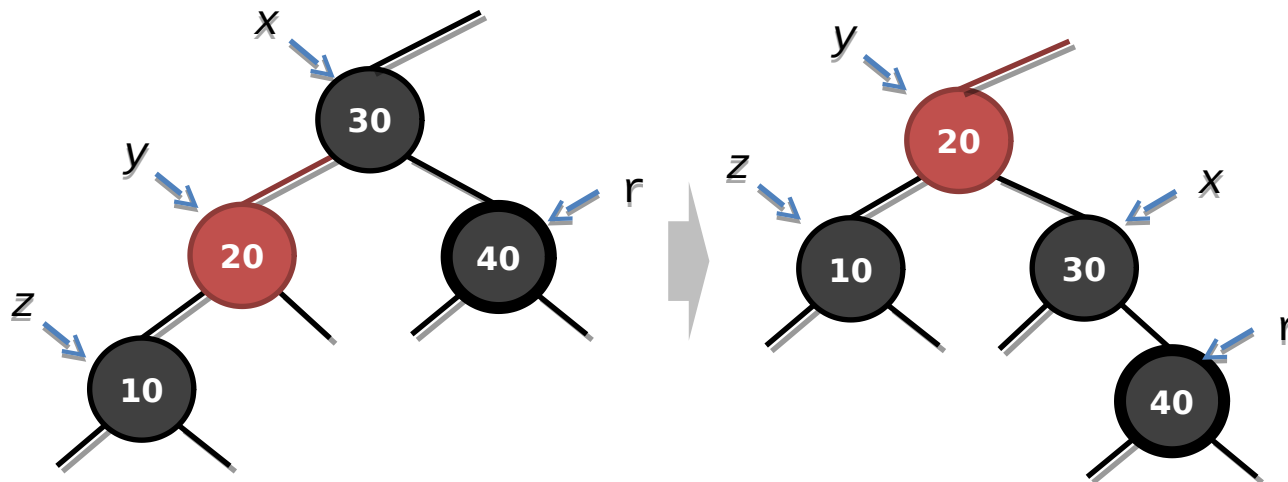
ARN. Resolver doble negro (Caso 3)

Caso 3: el hermano y de r es rojo

Traer el nodo rojo como padre de r para que el hermano sea un nodo negro y se transforme a caso 1 o caso 2.

Para ello se realiza el ajuste siguiente:

- Si y es el h_der de x , z es el h_der de y
 - Si y es el h_izda de x , z es el h_izda de y
 - Colorear y de negro y x de rojo
- ➔ r. trinodo en z



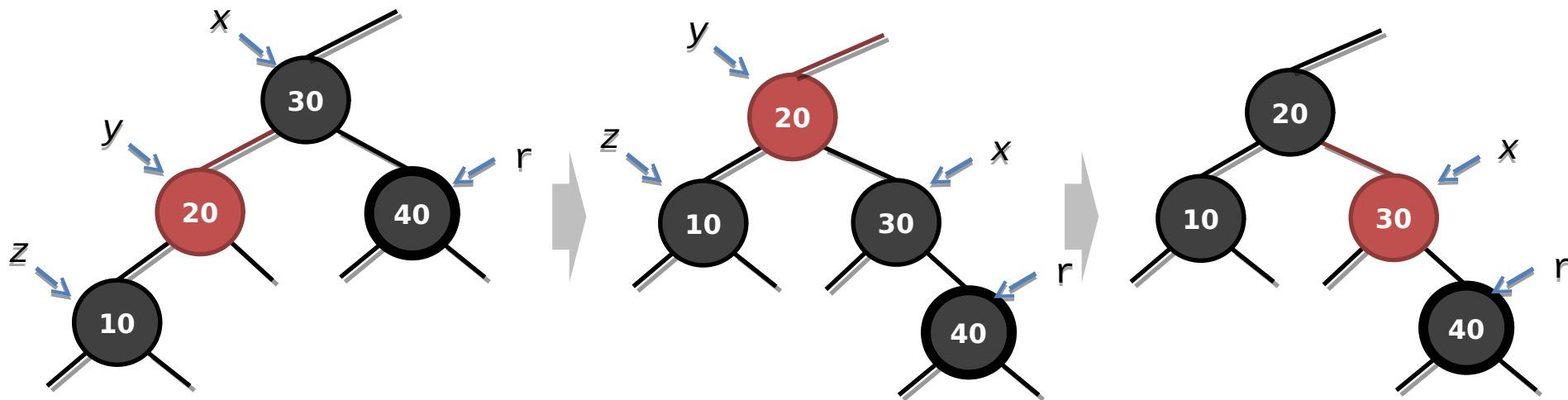
ARN. Resolver doble negro (Caso 3)

Caso 3: el hermano y de r es rojo

Traer el nodo rojo como padre de r para que el hermano sea un nodo negro y se transforme a caso 1 o caso 2.

Para ello se realiza el ajuste siguiente:

- Si y es el h_der de x , z es el h_der de y
 - Si y es el h_izda de x , z es el h_izda de y
 - Colorear y de negro y x de rojo
- ➔ r . trinodo en z



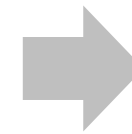
ARN. Resolver doble negro (Caso 3)

Caso 3: el hermano y de r es rojo

Traer el nodo rojo como padre de r para que el hermano sea un nodo negro y se transforme a caso 1 o caso 2.

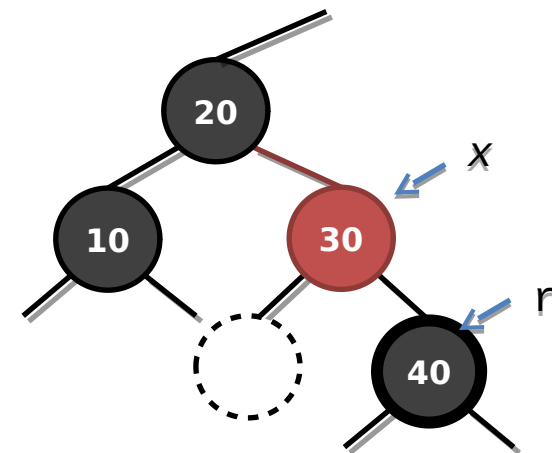
Para ello se realiza el ajuste siguiente:

- Si y es el h_der de x , z es el h_der de y
- Si y es el h_izda de x , z es el h_izda de y
- Colorear y de negro y x de rojo



r . trinodo en z

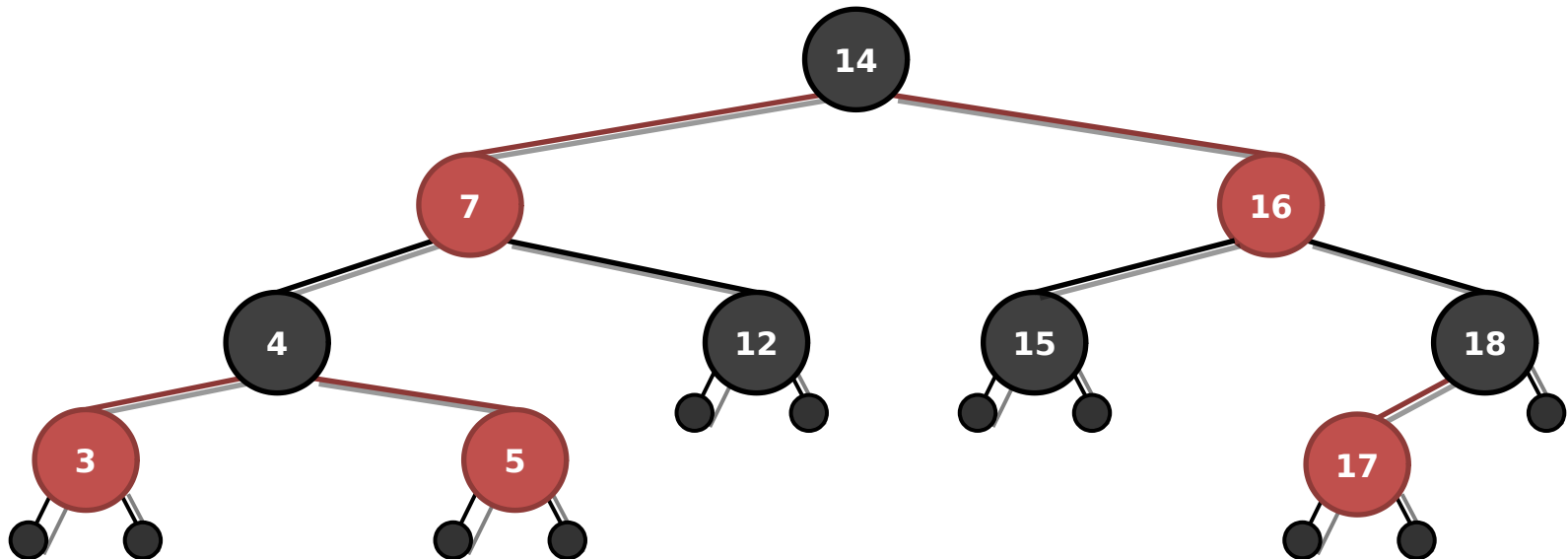
Obsérvese que tras el ajuste del caso 3 el problema del doble de negro no desaparece, pero el hermano de r será un nodo negro y por tanto se transforma en caso 1 o 2.



ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

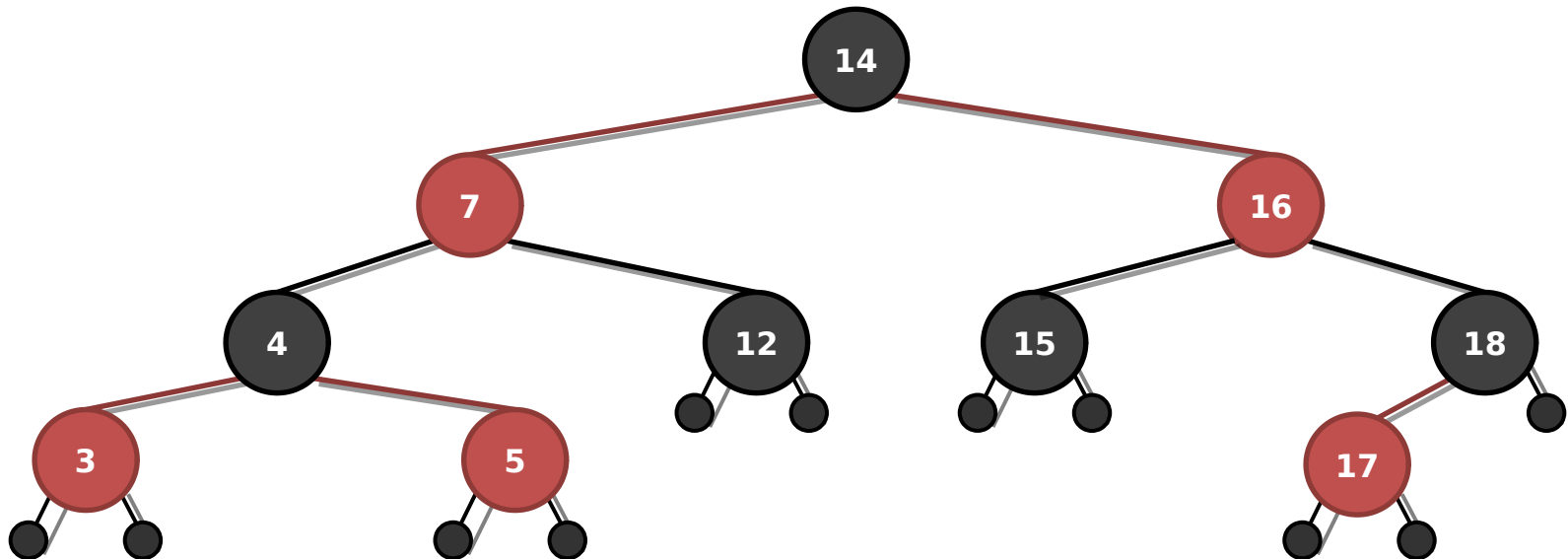
– 3, 12, 17, 18, 15, 16



ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

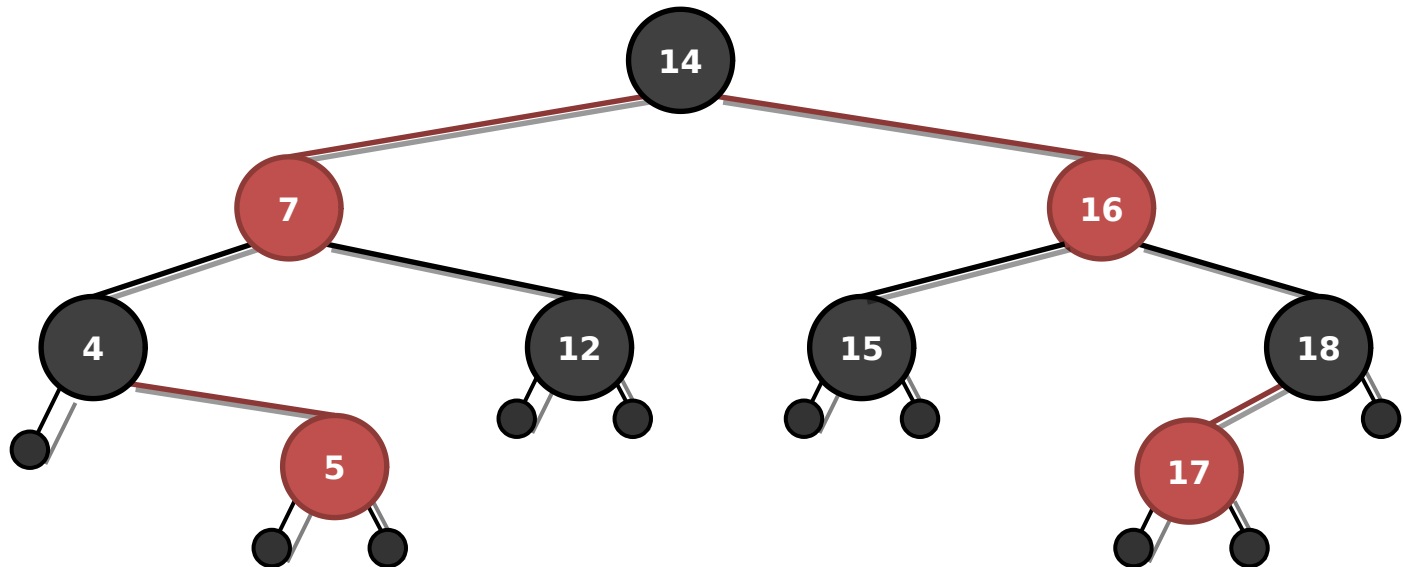


El nodo 3 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

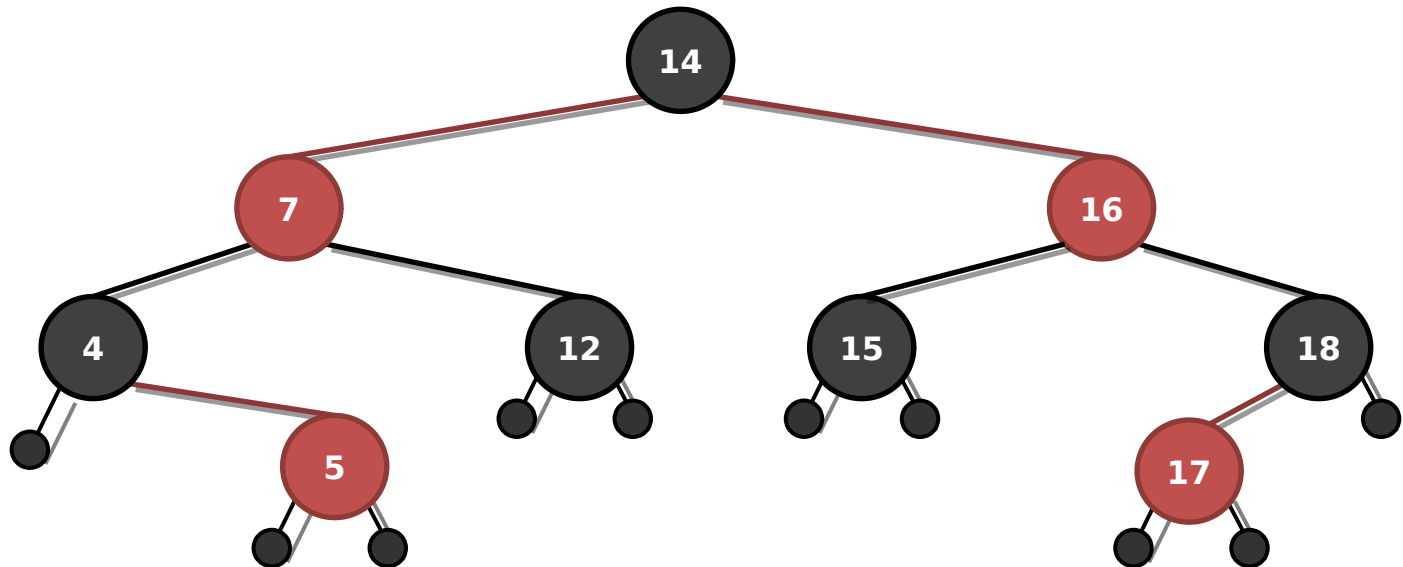


La altura negra
está equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

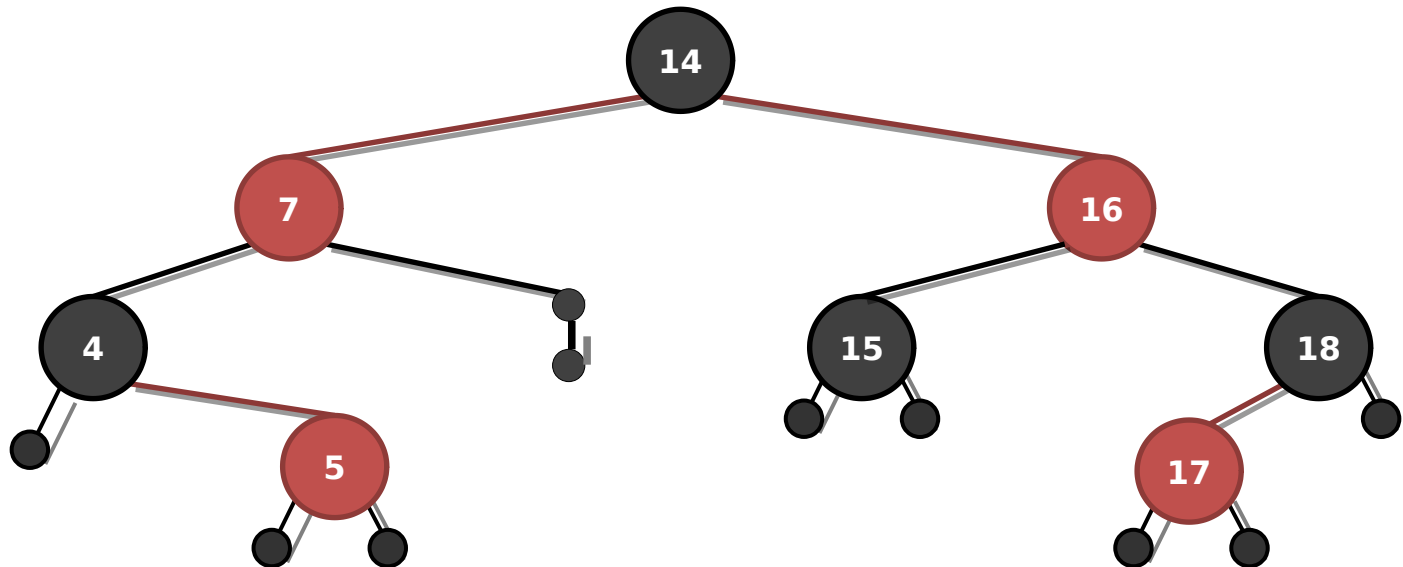


El nodo 12 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

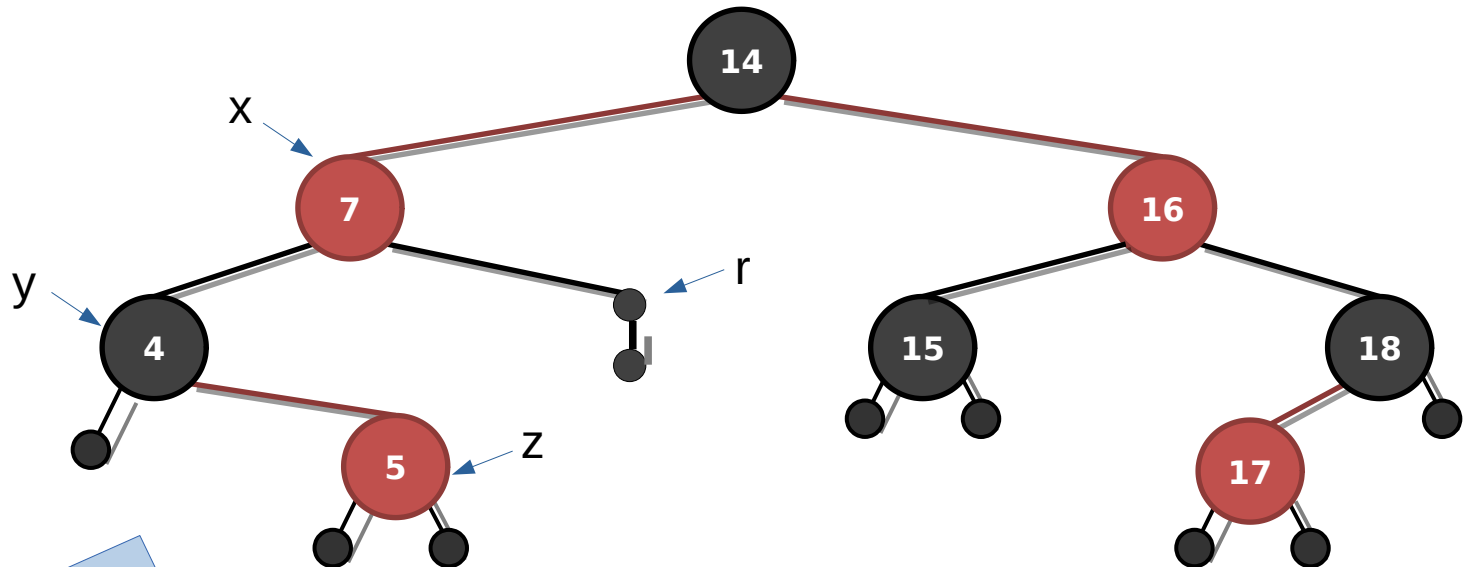


Necesitamos doble negro para que la altura negra esté equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

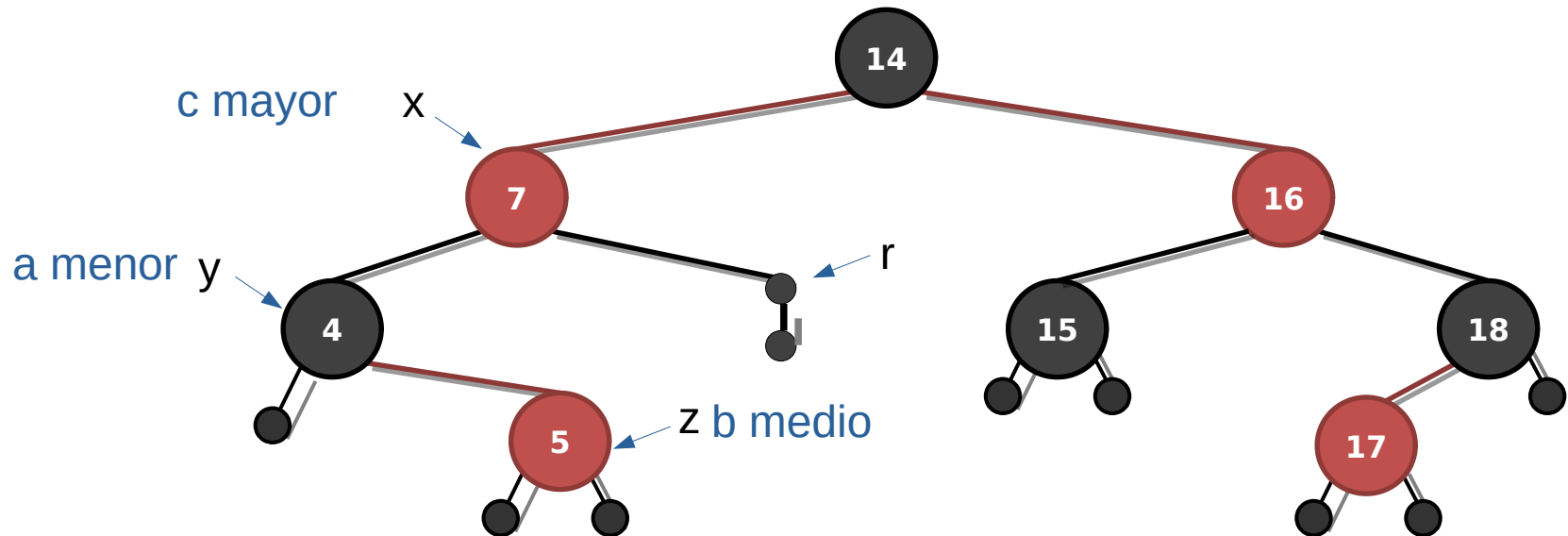


Caso 1: y, hermano de r, es negro y tiene un hijo rojo, z

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16



Aplicar reestructuración trinodo

Colorear *a* y *c* de negro

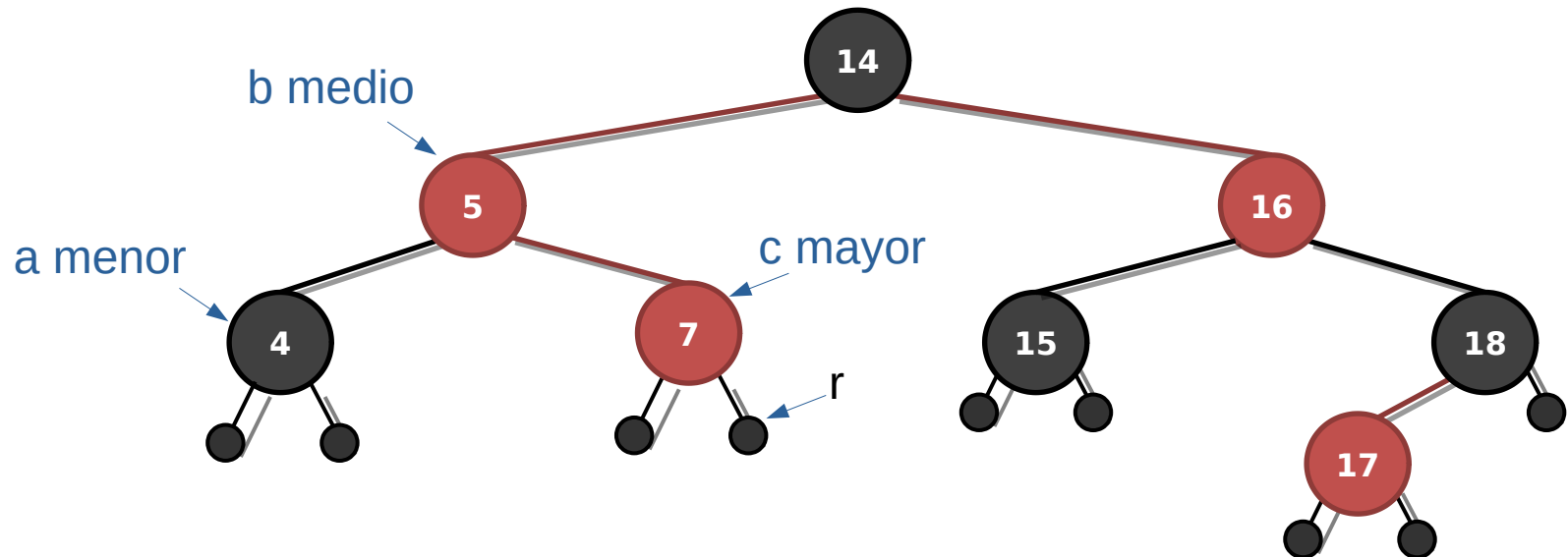
Asignar a *b* el color de que tenía *x*

Colorear *r* de negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16



Aplicar reestructuración trinodo

Colorear a y c de negro

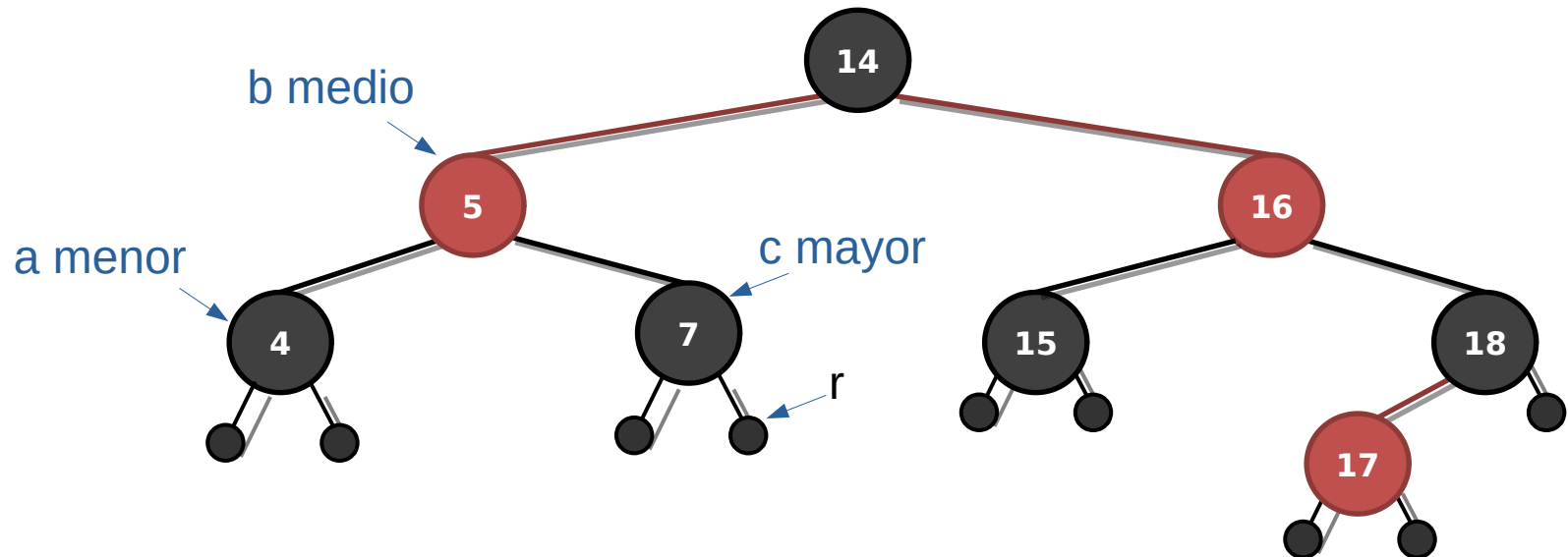
Asignar a *b* el color de que tenía *x*

Colorear *r* de negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16



Aplicar reestructuración trinodo

Colorear a y c de negro

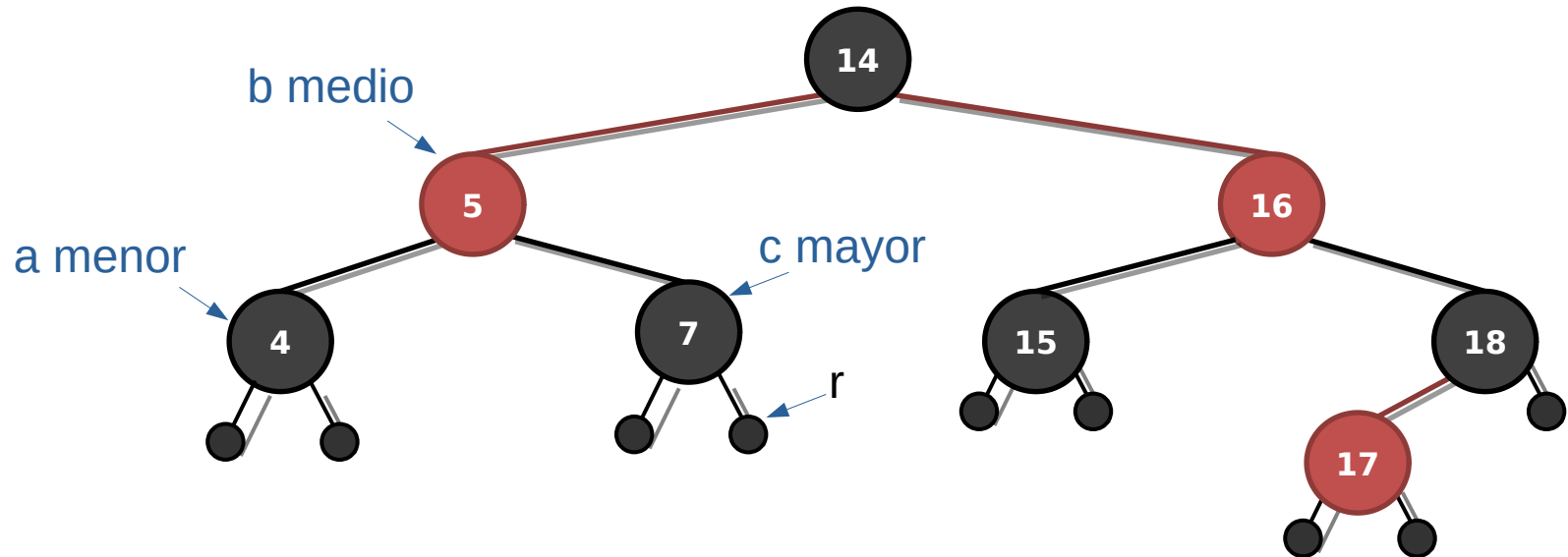
Asignar a b el color de que tenía x

Colorear r de negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16



Aplicar reestructuración trinodo

Colorear a y c de negro

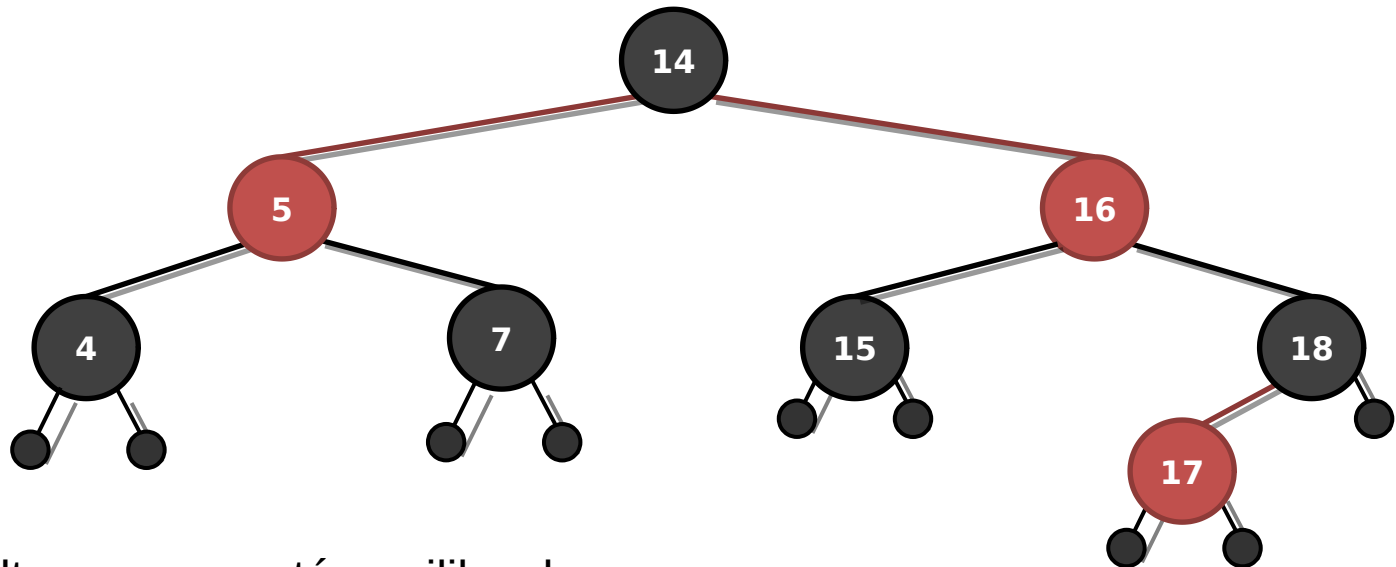
Asignar a b el color de que tenía x

Colorear r de negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

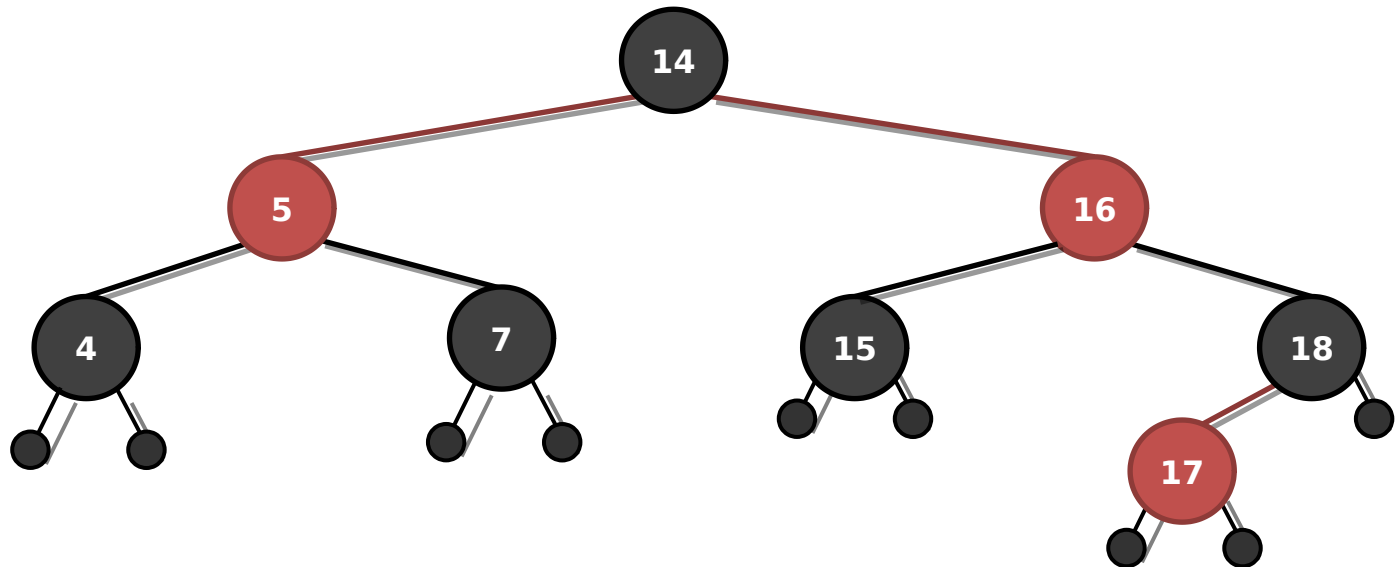


La altura negra está equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, **17**, 18, 15, 16

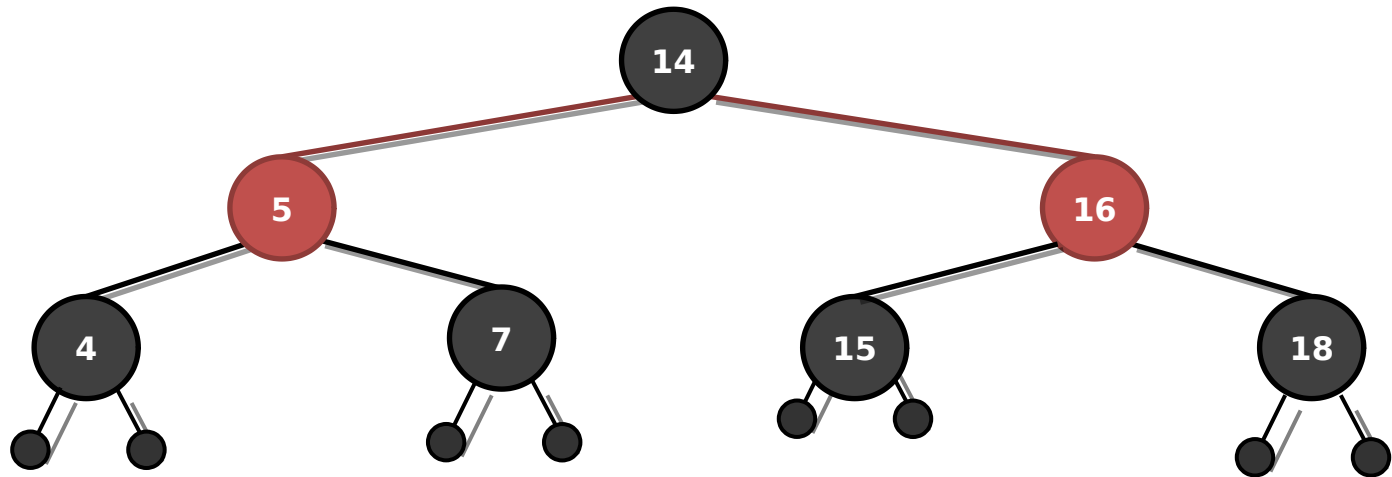


El nodo 17 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, 16

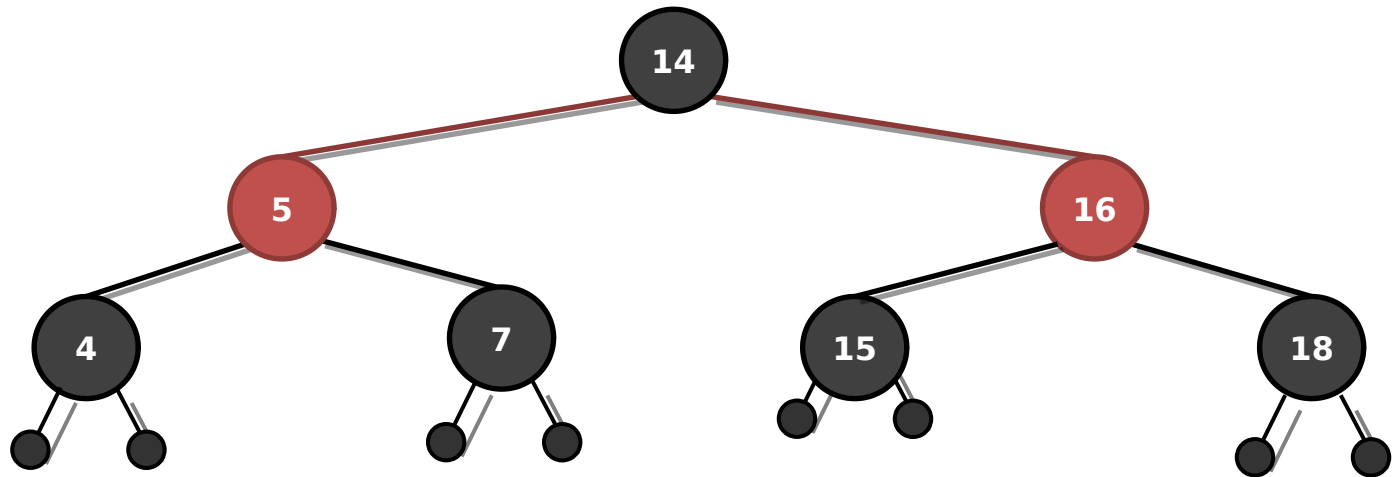


La altura negra está equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16

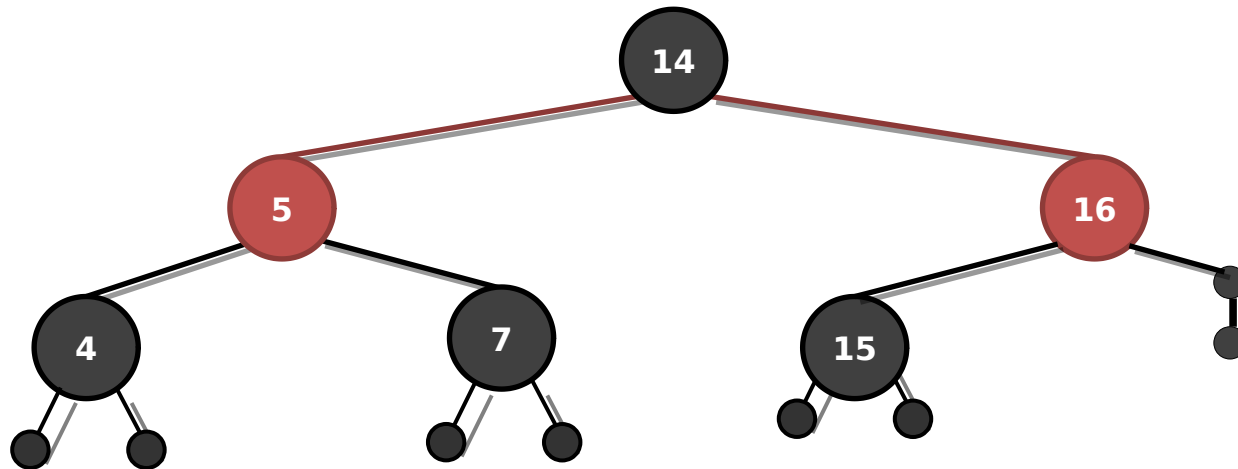


El nodo 18 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16

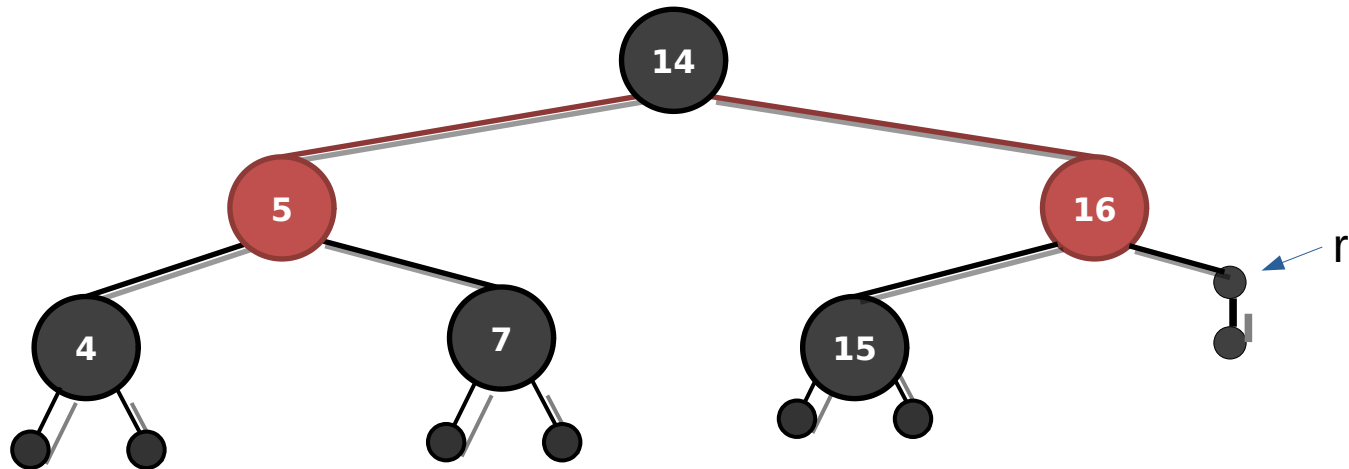


Necesitamos doble negro para que la altura negra esté equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16

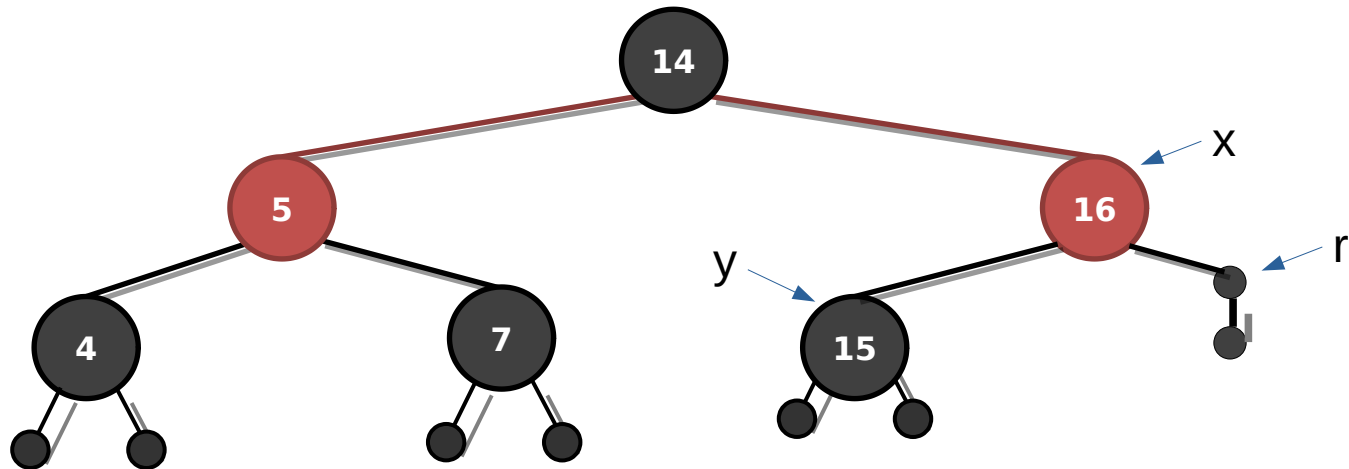


Caso 2: y, hermano de r, es negro y sus hijos también

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16



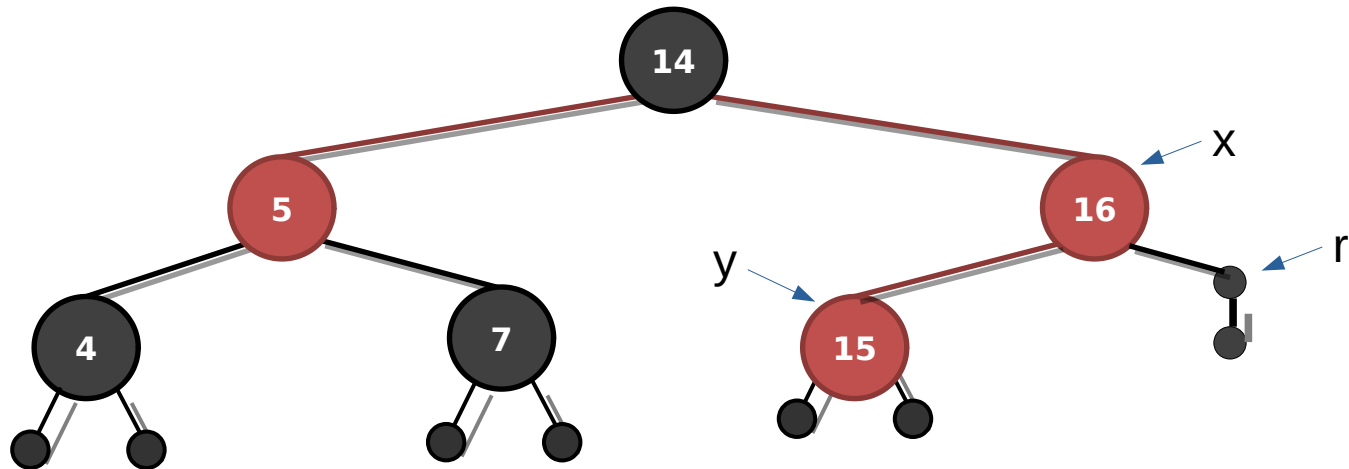
Recoloración de r como negro e y como rojo

Si x es rojo, se pasa a negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16



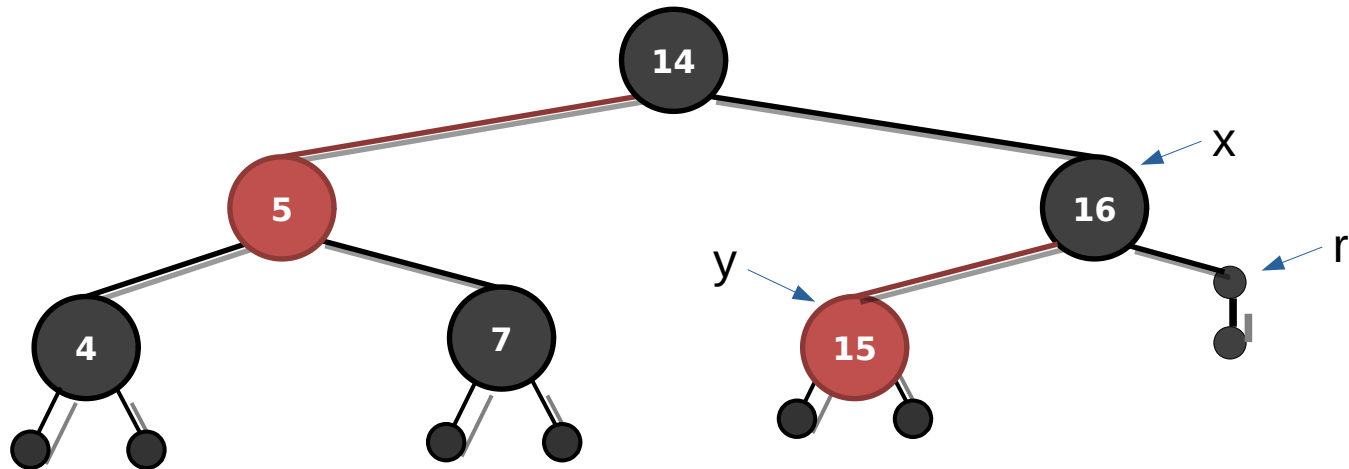
Recoloración de r como negro e y como rojo

Si x es rojo, se pasa a negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16



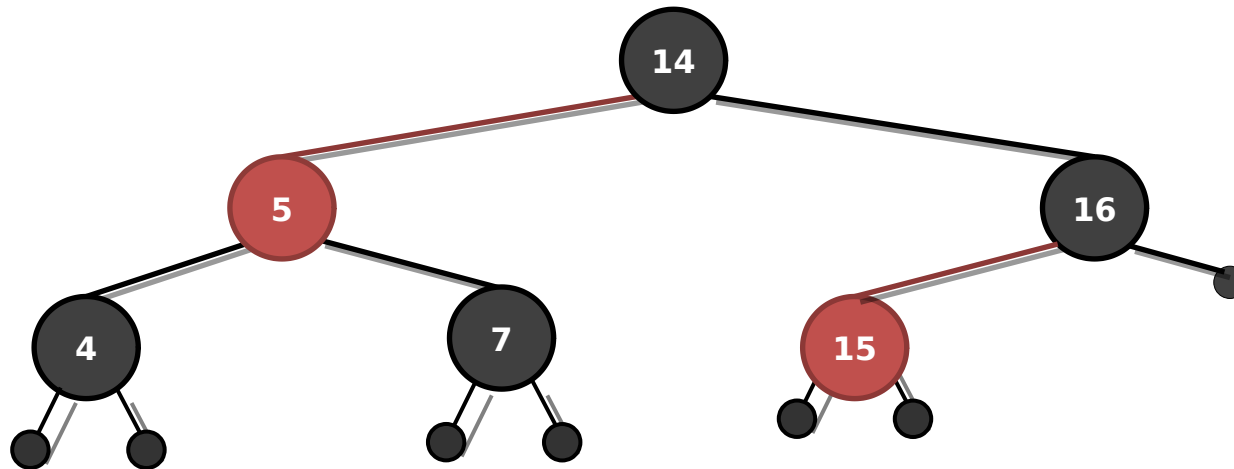
Recoloración de r como negro e y como rojo

Si x es rojo, se pasa a negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, **18**, 15, 16

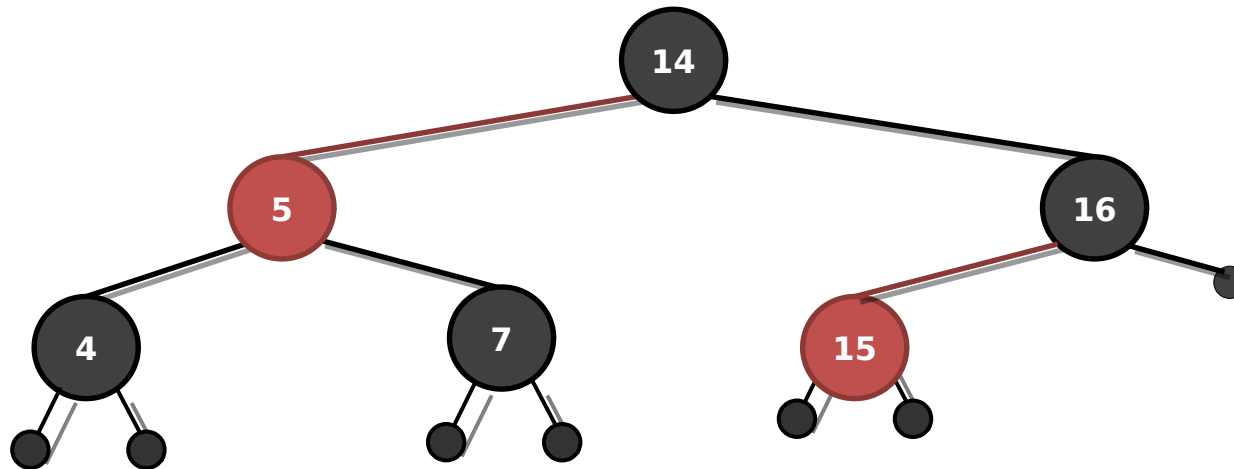


La altura negra está equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, **15**, 16

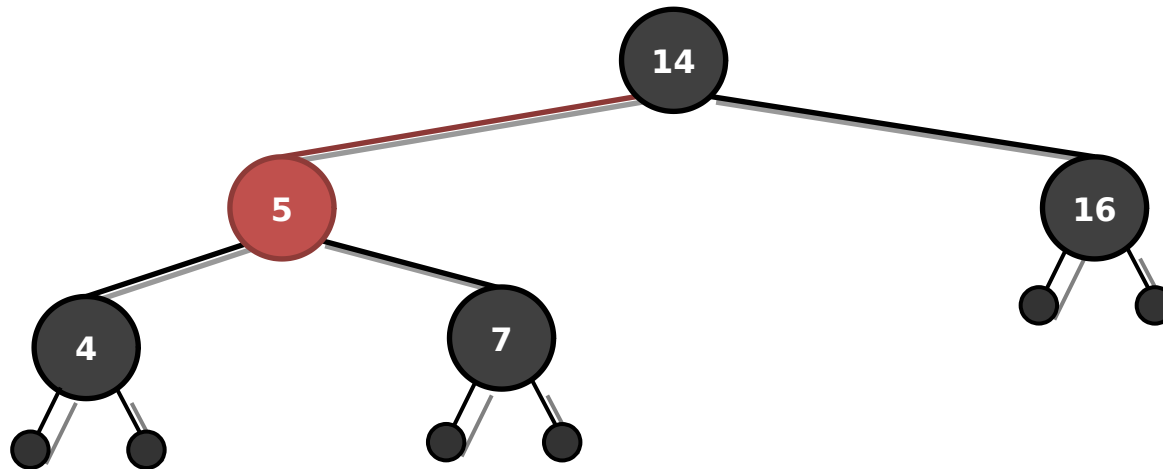


El nodo 15 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, **15**, 16

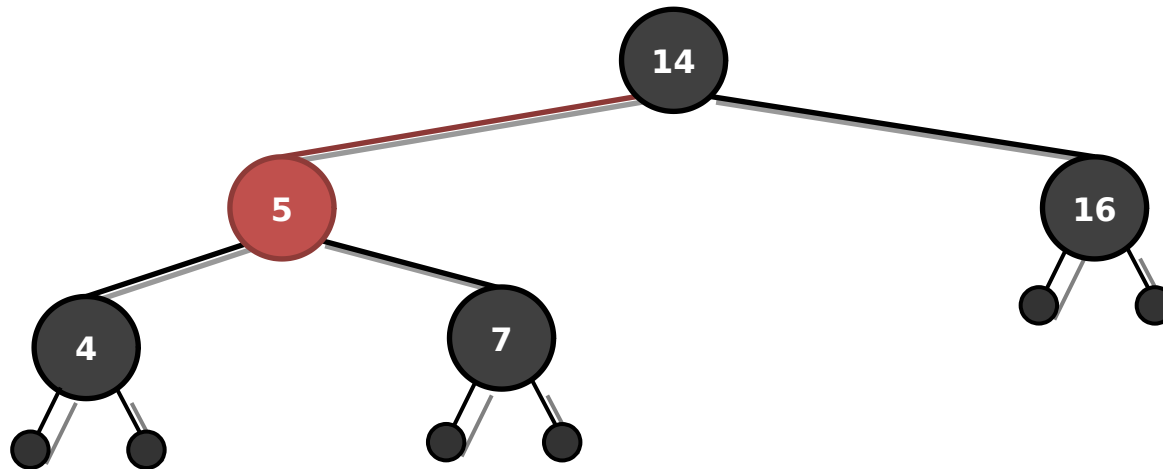


La altura negra está equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

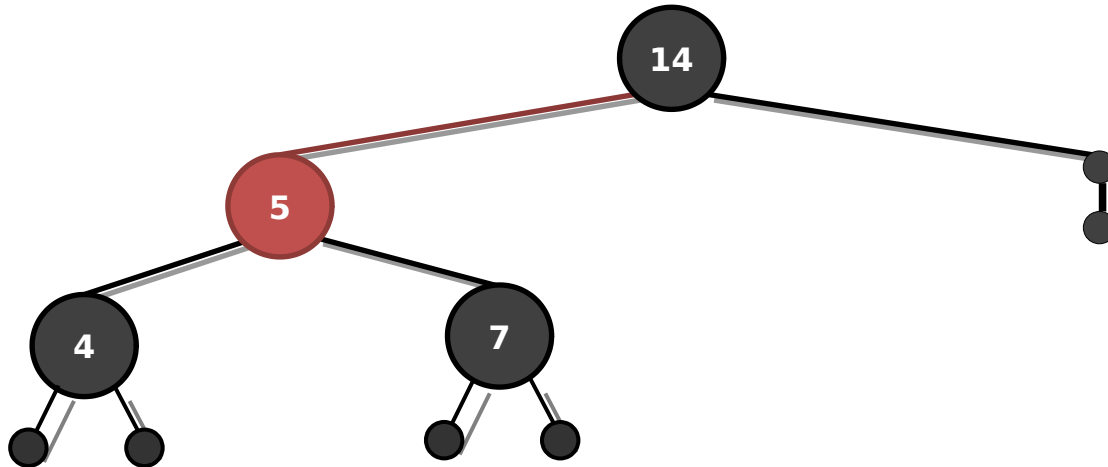


El nodo 16 es hoja,
se borra sin más

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

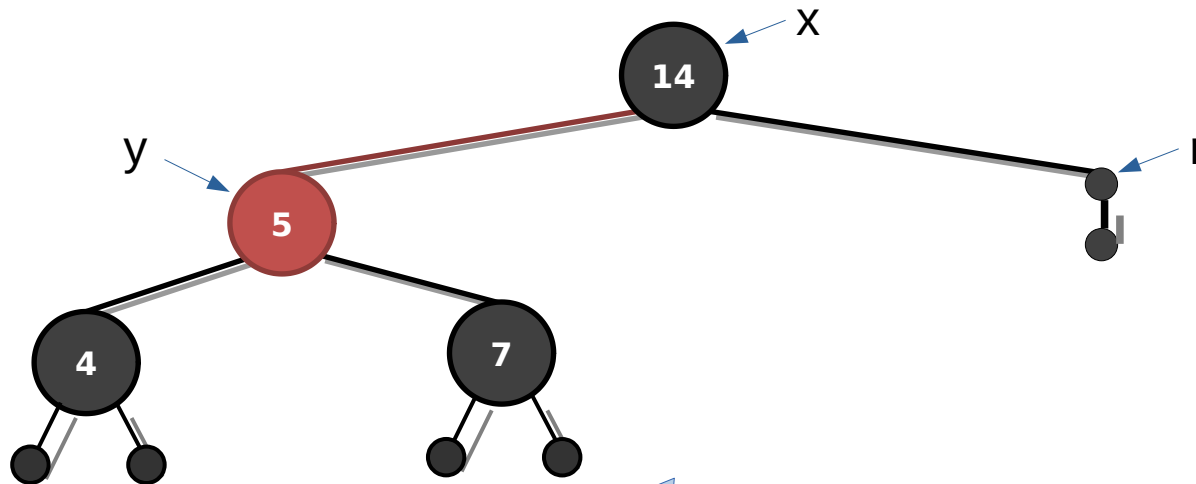


Necesitamos doble negro para que la altura negra esté equilibrada

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

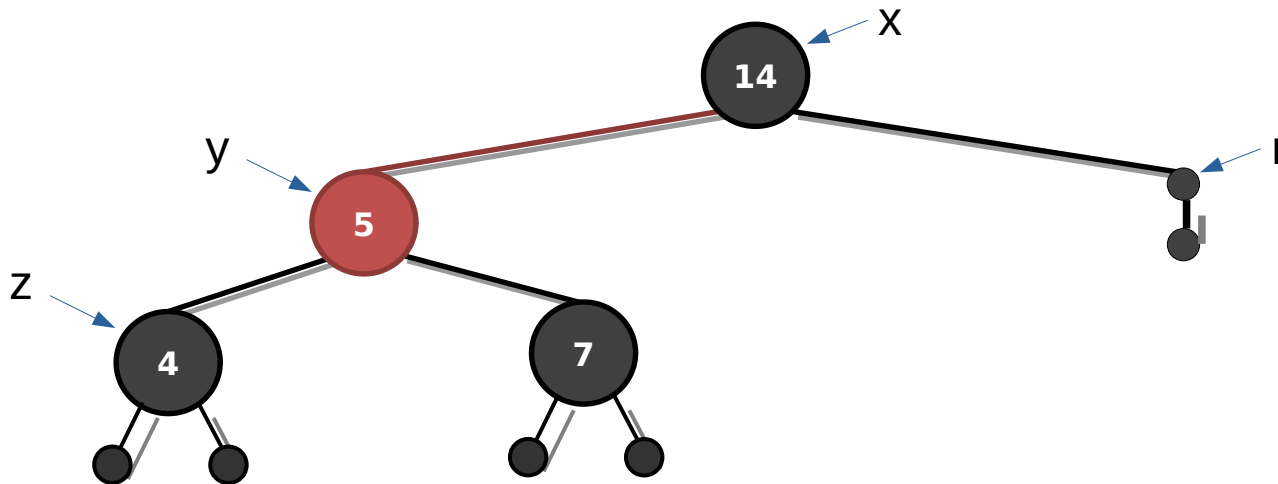


Caso 3: y, hermano de r,
es rojo

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**



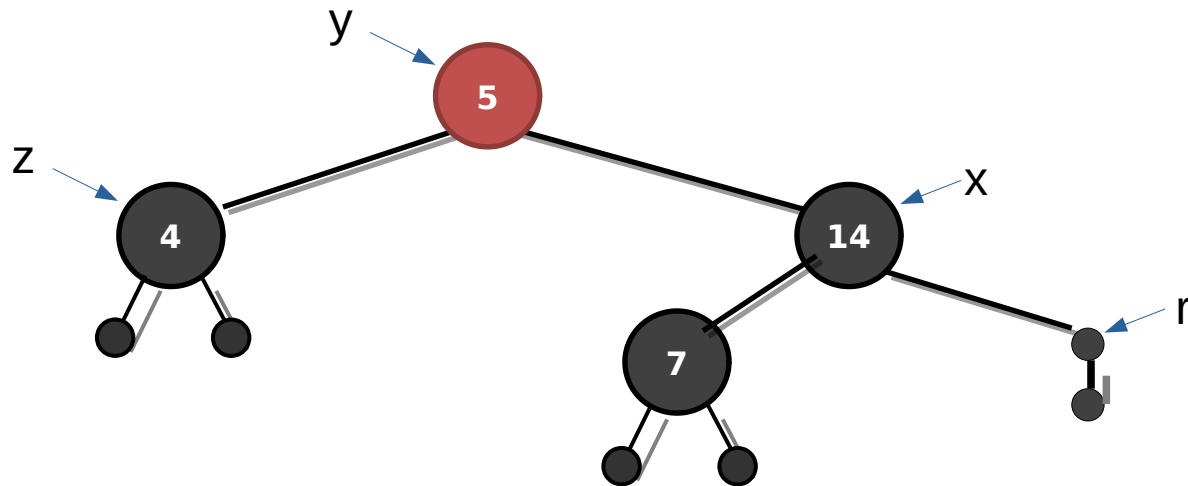
Si y es el h_izda de x, z es el h_izda de y, rotación z

Colorear y de negro y x de rojo

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**



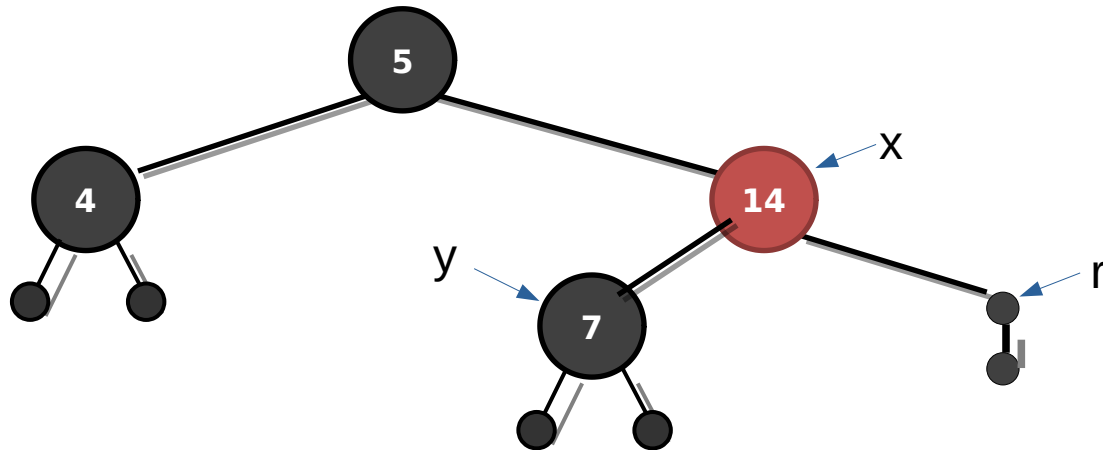
Si y es el h_izda de x, z es el h_izda de y, rotación z

Colorear y de negro y x de rojo

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

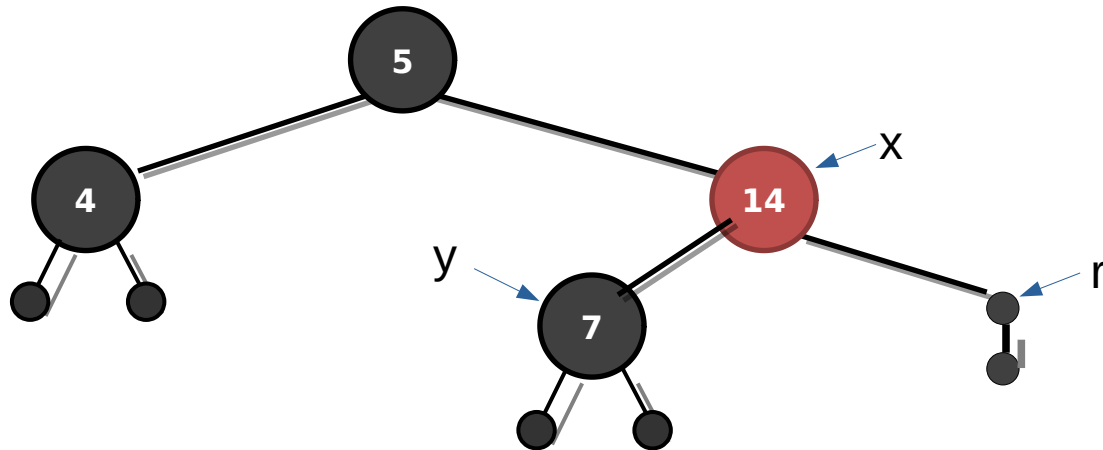


Caso 2: y, hermano de r,
es negro y sus hijos también

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

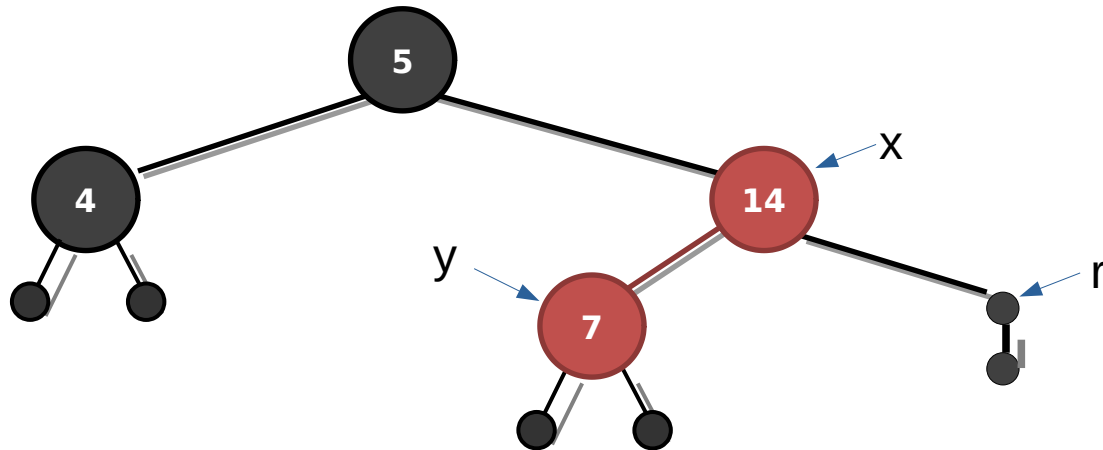


Recoloración de r como negro e y como rojo
Si x es rojo, se pasa a negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

– 3, 12, 17, 18, 15, **16**

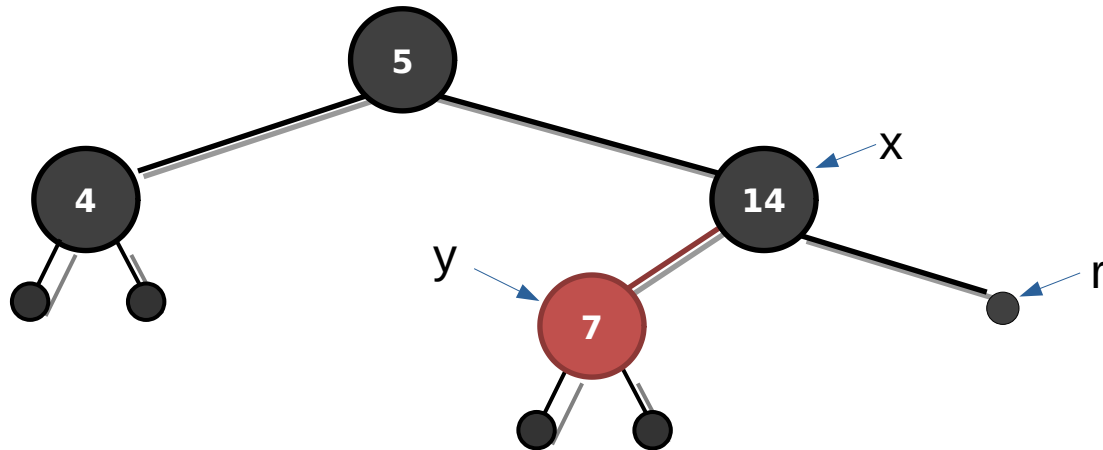


Recoloración de r como negro e y como rojo
Si x es rojo, se pasa a negro

ARN. Ejemplo borrado

Realizar la siguiente secuencia de borrados en el ARN que se presenta a continuación:

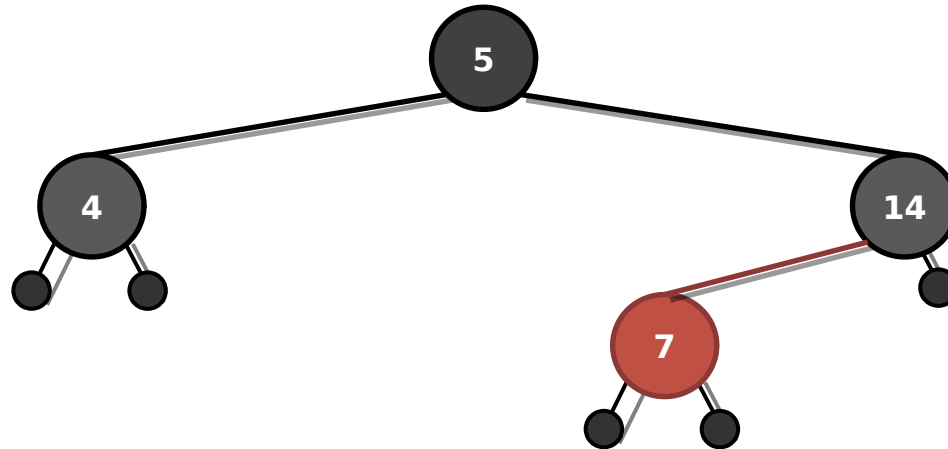
– 3, 12, 17, 18, 15, **16**



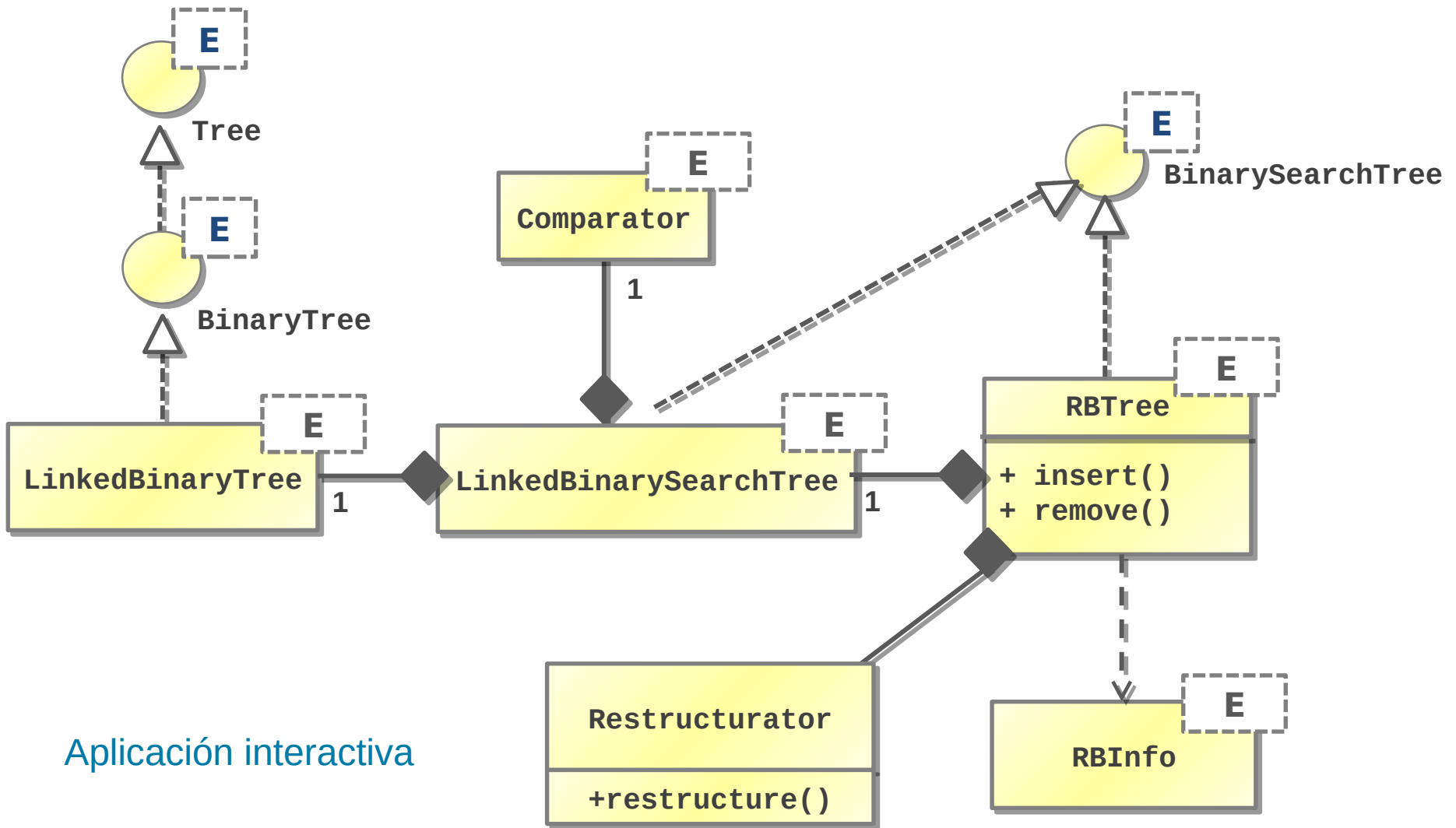
La altura negra está equilibrada

ARN. Ejemplo borrado

ARN resultante tras eliminar la secuencia de valores
– 3, 12, 17, 18, 15, 16

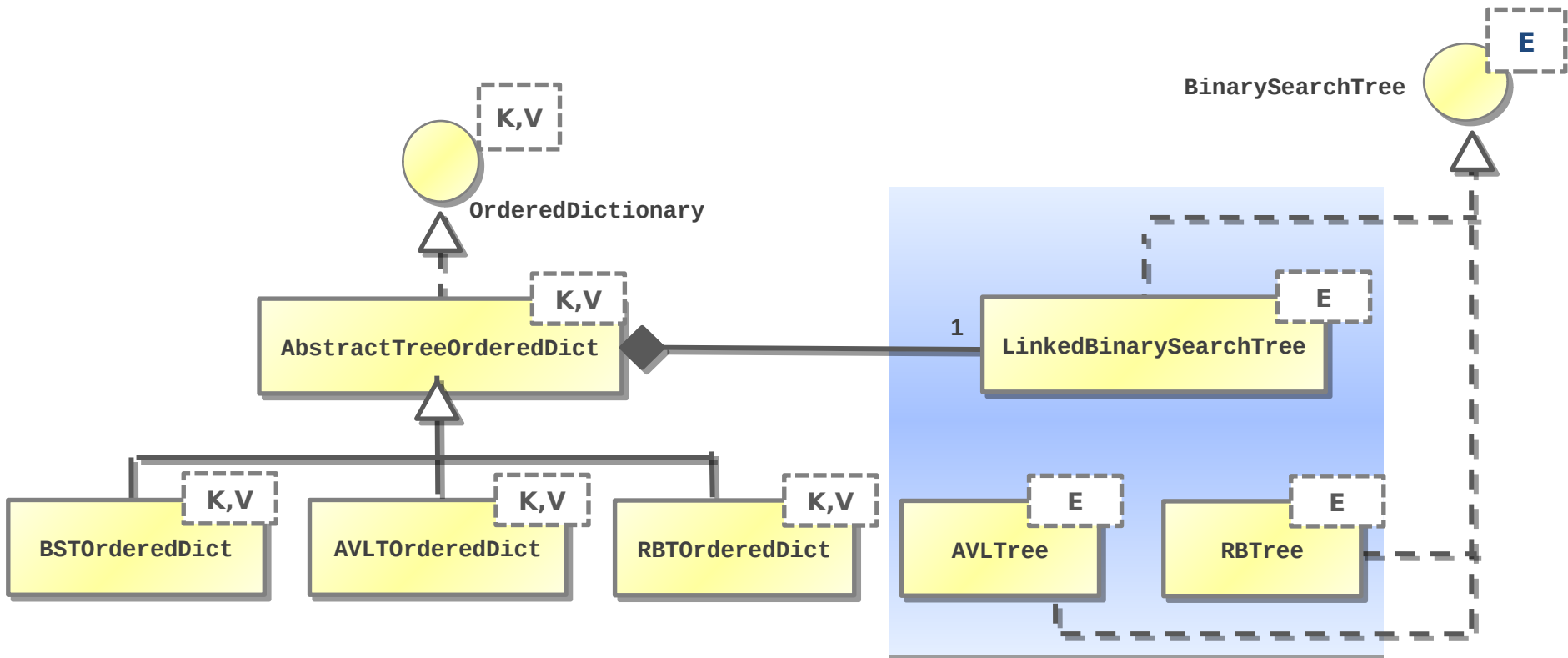


ARN. Implementación Java



Diccionario ordenado sobre ABB, ARN o AVL

Diccionario ordenado sobre ABB, ARN o AVL



Análisis del Equilibrados por AVL o Rojo Negro

- Una **reestructuración** es $O(1)$
 - Asumiendo una implementación como `LinkedBinaryTree`
- **Encontrar** un elemento es $O(\log n)$
 - Altura del árbol. No requiere reestructuración
- **Insertar** un elemento es $O(\log n)$
 - Hay que encontrar la hoja donde insertarlo, $O(\log n)$
 - La reestructuración podría llegar hasta la raíz, $O(\log n)$
- **Eliminar** un elemento es $O(\log n)$
 - Hay que encontrar la hoja donde intercambiar, $O(\log n)$
 - La reestructuración podría llegar hasta la raíz, $O(\log n)$

- Estructuras de datos con elementos ordenados por algún criterio:
 - Obtención de rangos de DNIs
 - Obtención de valores (números, fechas, textos) entre dos límites: listado de apellidos, listado de operaciones bancarias...
- Búsquedas del más próximo basadas en algún criterio.
 - Obtener el número almacenado más cercano a X
 - Obtener la fecha más cercana a F en la que se hizo una operación
 - Obtener el patrón más parecido a P