Árboles

EEDD - GRADO EN ING. INFORMATICA - UCO

Árboles binarios de búsqueda equilibrados

Contenidos

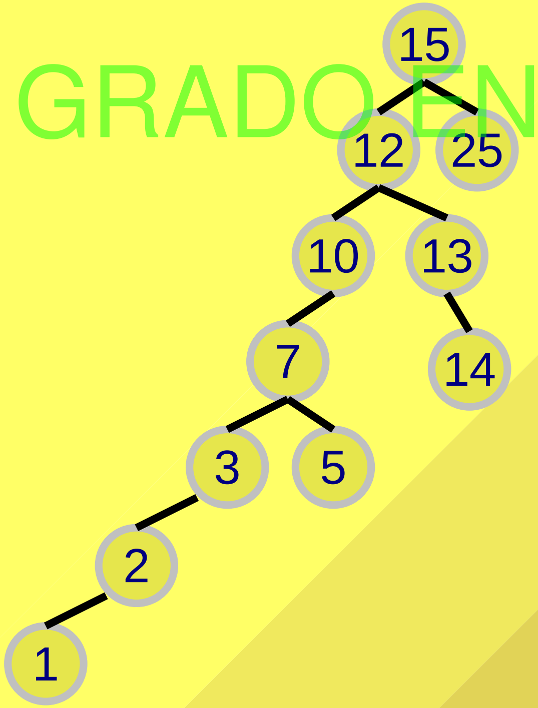
- Concepto de árbol perfectamente equilibrado.
- Concepto de árbol equilibrado (AVL).
- Especificación del TAD AVLTree
- Representación enlazada.
- Operaciones de equilibrado del árbol.
- Operaciones de inserción y borrado.

EEDD - GRADO EN ING. INFORMATICA - UCO

Motivación

- La búsqueda es la operación estrella O(H).

Dada la secuencia: {15,12,25,10,13,7,14,3,5,2,1} ¿cuál será el BSTree?



EEDD - GRADO EN ING. INFORMATICA - UCO

¿Cuál es la altura del árbol?

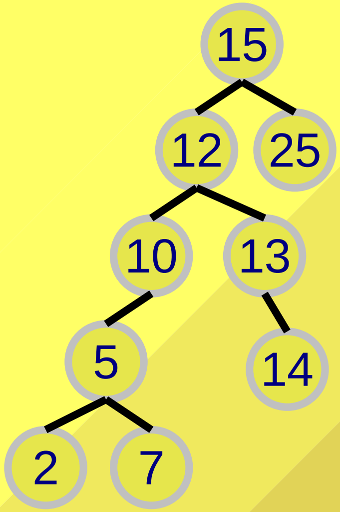
¿Cuál sería la altura mínima posible para este árbol?

¿Se puede hacer mejor?

Árbol Perfectamente Equilibrados

- Concepto.
  - Un árbol no vacío T está perfectamente equilibrado si para todo subárbol T:
    - $|Tamaño(T_{der})-Tamaño(T_{izq})| <= 1$

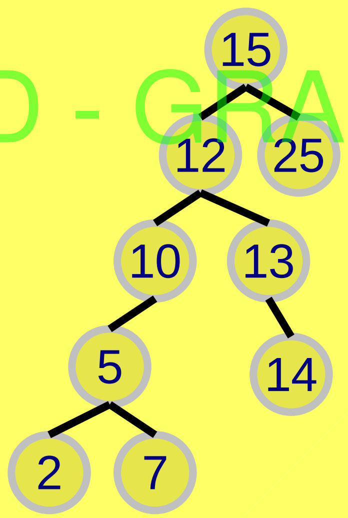
EEDD - GRADO EN ING. INFORMATICA - UCO



¿Está perfectamente equilibrado?

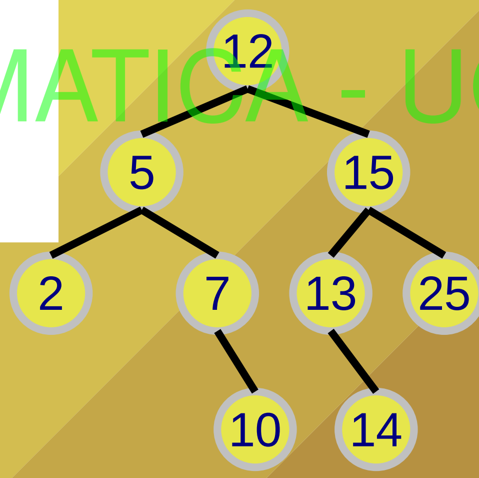
Árbol Perfectamente Equilibrados

```
Algorithm generateTree(d:Array[G], Var
  t:BSTree[G])
Prec: d.size()==0 OR “d is in order”
Begin
  If d.size>0 Then
    t.insert(d[d.size()//2])
    generateTree(d[0:d.size()//2], t)
    generateTree(d[d.size()//2:
                  d.size()], t)
  End-If
End.
```



No equilibrado

¿Cuál es **una** versión perfectamente equilibrada?



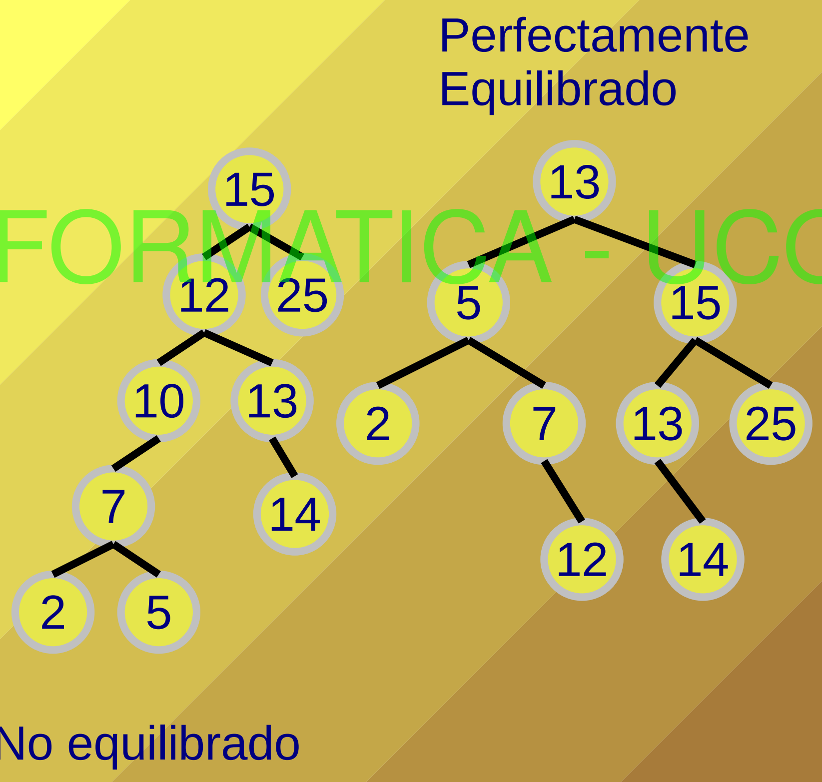
Perfectamente Equilibrado

EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol Perfectamente Equilibrados

- Ventajas:
  - $H=[Log_2 N]$
  - Búsqueda $O(H) = O(Log_2 N)$.
- Inconvenientes:
  - Alto coste en Inserción o Borrado para mantener el equilibrio.

EEDD - GRADO EN ING. INFORMATICA - UCO



Árbol AVL

- Concepto.
  - Adelson-Velskii y Landis (AVL)
  - Un árbol no vacío T es AVL si para todo subárbol:
    - $|h(T_{der})-h(T_{izq})|<=1$
  - Factor de Equilibrio (FE): **$h(T_{der})-h(T_{izq})$**
  - Ventajas árboles AVL:
    - Propiedad: $H \approx Log_2(N)$
    - Búsqueda $O(H) \approx O(Log_2 N)$.

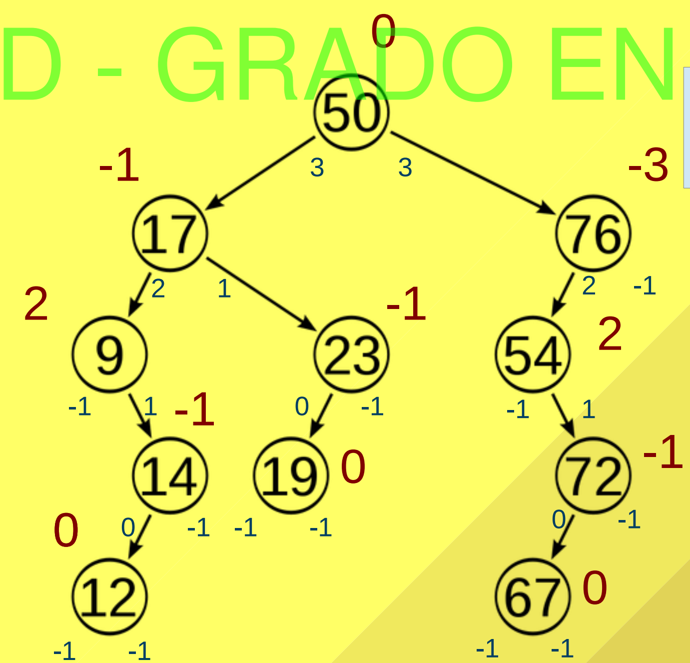
EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Ejemplo de cálculo de los factores de equilibrio.
  - Factor de Equilibrio (FE) **h(der)-h(izq)**

¿Puedes calcular los FE?

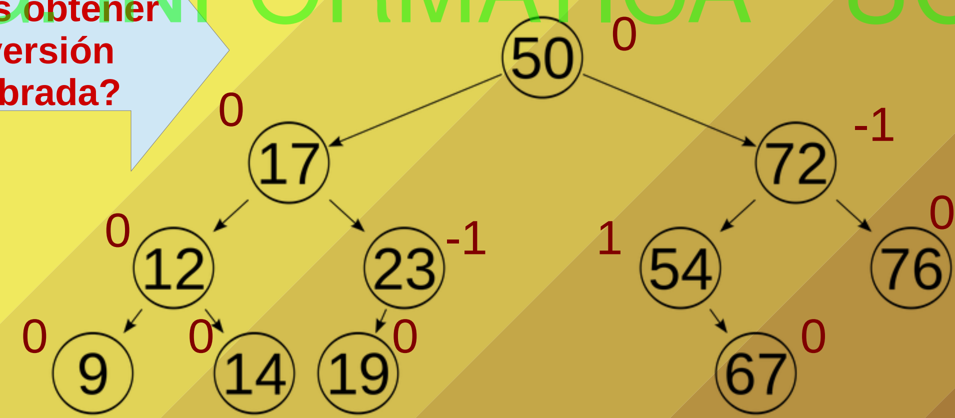
EEDD - GRADO EN ING. INFORMATICA - UCO



No equilibrado

¿Puedes obtener Una versión Equilibrada?

¿Puedes calcular los FE?



Equilibrado pero ¿está perfectamente equilibrado?

Árbol AVL

- Mantenimiento del equilibrio.
  - El desequilibrio se producirá al insertar/borrar y se corrige con “rotaciones” O(1) en la rama donde se produjo el desequilibrio.
  - Para cada subárbol en la rama, si |FE|>1, estará desequilibrado.
  - En promedio:
    - Hay que rotar en el 50% de las inserciones (sólo una o dos veces)
    - Hay que rotar en el 20% de los borrados (puede ser necesaria más de dos veces).

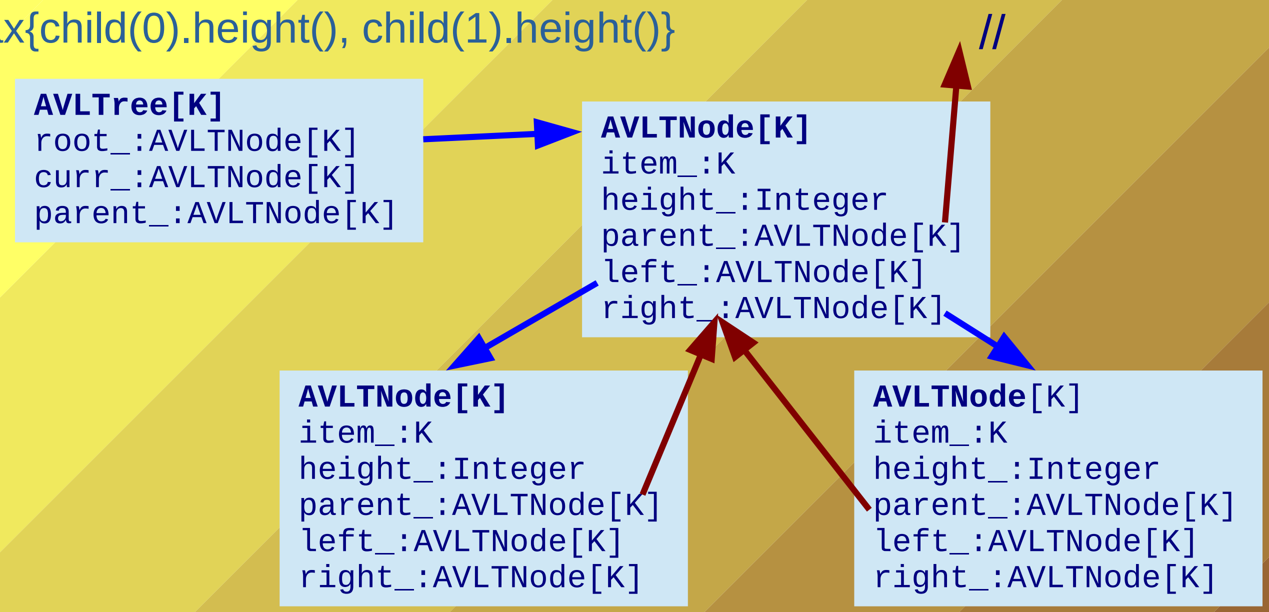
EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Representación con nodos enlazados.
  - Nuevo TAD AVLTNode como una extensión del TAD BTNode con operaciones para observar/modificar:
    - **parent(), setParent(AVLTNode[T])** //El padre del nodo.
    - **child(Int), setChild(Int, AVLTNode[T])** //child(0) hijo izquierdo, child(1) hijo derecho.
    - **height(), updateHeight()** // La altura del nodo en el árbol (O(1)).
    - **balanceFactor()** // Factor de balance del node (O(1))
    - **Invariante:** height()=1+max{child(0).height(), child(1).height()}

EEDD - GRADO EN ING. INFORMATICA - UCO

```
Algorithm AVLTNode[T]::setChild(
     c:INT; //Child to set {0,1}
     n:AVLTNode[T]
); //node
BEGIN
  IF (c = 0) THEN
     left_ <- n
  ELSE
     right_ <- n
  IF (n <> Void) THEN
     n.setParent(this)
  updateHeight()
End.
```



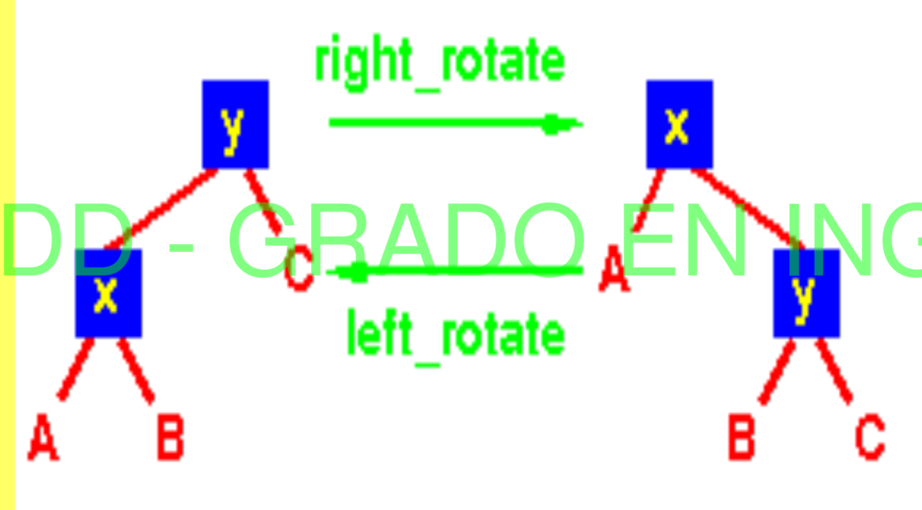
Árbol AVL

- Operaciones de inserción y borrado.
  - El primer paso de ambas operaciones se ejecuta usando la operación correspondiente del BSTree.
  - Segundo paso: makeBanlanced():
    - Siguiendo la cadena de nodos desde la posición del cursor hasta el nodo raíz, hacer:
      - Actualizar la altura del nodo O(1).
      - Calcular su factor de equilibrio FE O(1).
      - Si |FE|>1, el subárbol está desequilibrado, equilibrar con rotaciones O(1).

EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Mantenimiento del equilibrio: rotación.



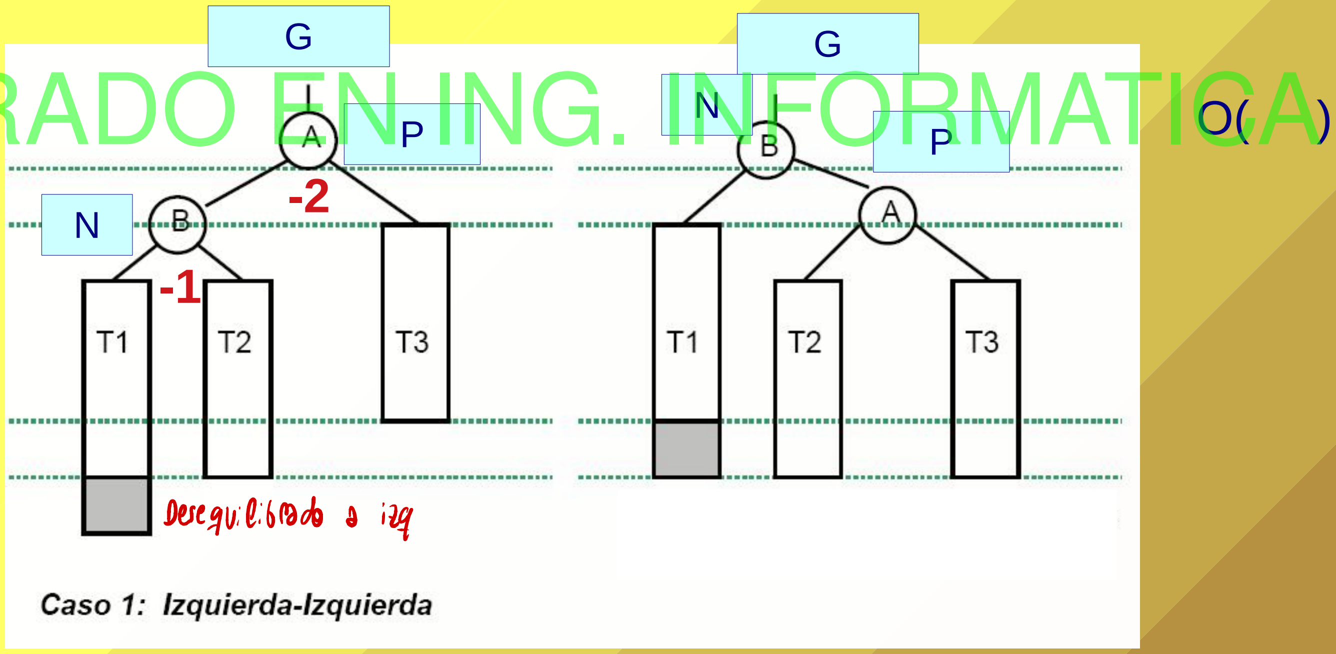
En ambas configuraciones, el recorrido en orden es el mismo:
A x B y C

```
Algorithm AVLTree[T]::rotate(
     P:AVLTNode[T]; //Rotate node
     dir:Int       //rotate dir.
):AVLTNode[T]  //new root node.
VAR
  G:AVLTNode[T] //Grandad.
  N:AVLTNode[T] //The child to promote.
  CN:AVLTNode[T] //Close nephew
  gpDir:Int //Direction G->P
BEGIN
  G <- P.parent()
  gpDir <- G.child(0)==P ? 0 : 1
  N <- P.child(1-dir)
  CN <- N.child(dir)
  P.setChild(1-dir, CN)
  N.setChild(dir, P)
  IF G<>Void THEN
     G.setChild(gpDir, N)
  ELSE
     N.setParent(Void)
     setRoot(N)
  END-IF
  RETURN N // new root of the rotated subtree
End.
```

EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 1: izq-izq -> rotación a derecha.



EEDD - GRADO EN ING. INFORMATICA - UCO

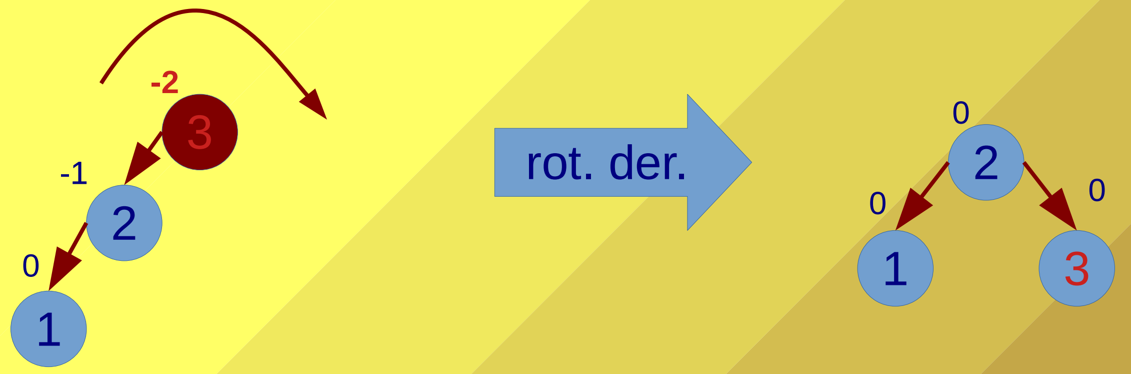
Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 1: ejemplo.

{3,2,**1**}

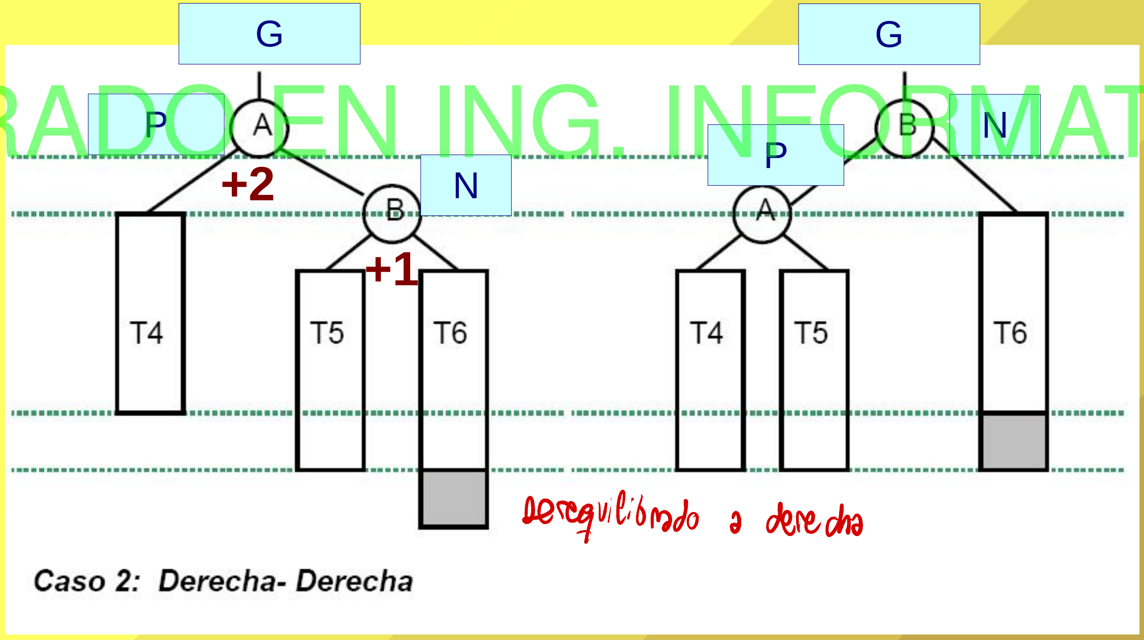
EEDD - GRADO EN ING. INFORMATICA - UCO

Desequilibrio izq-izq → rotación a der. de P.



Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 2: der-der -> rotación izquierda.



O( )

EEDD - GRADO EN ING. INFORMATICA - UCO

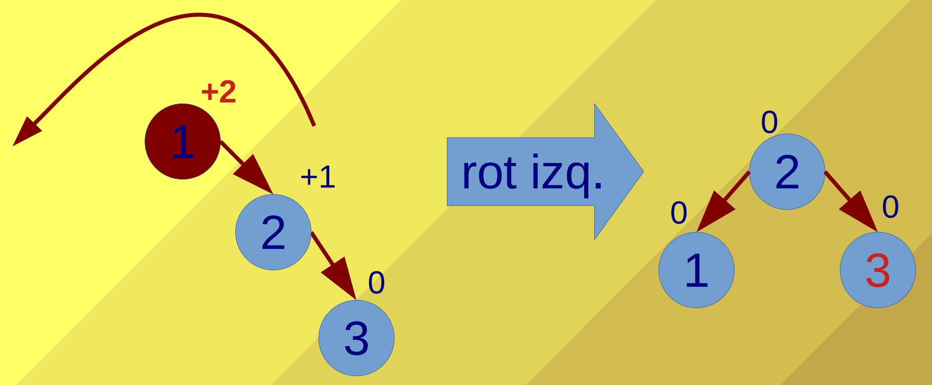
Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 2: ejemplo.

{1,2,**3**}

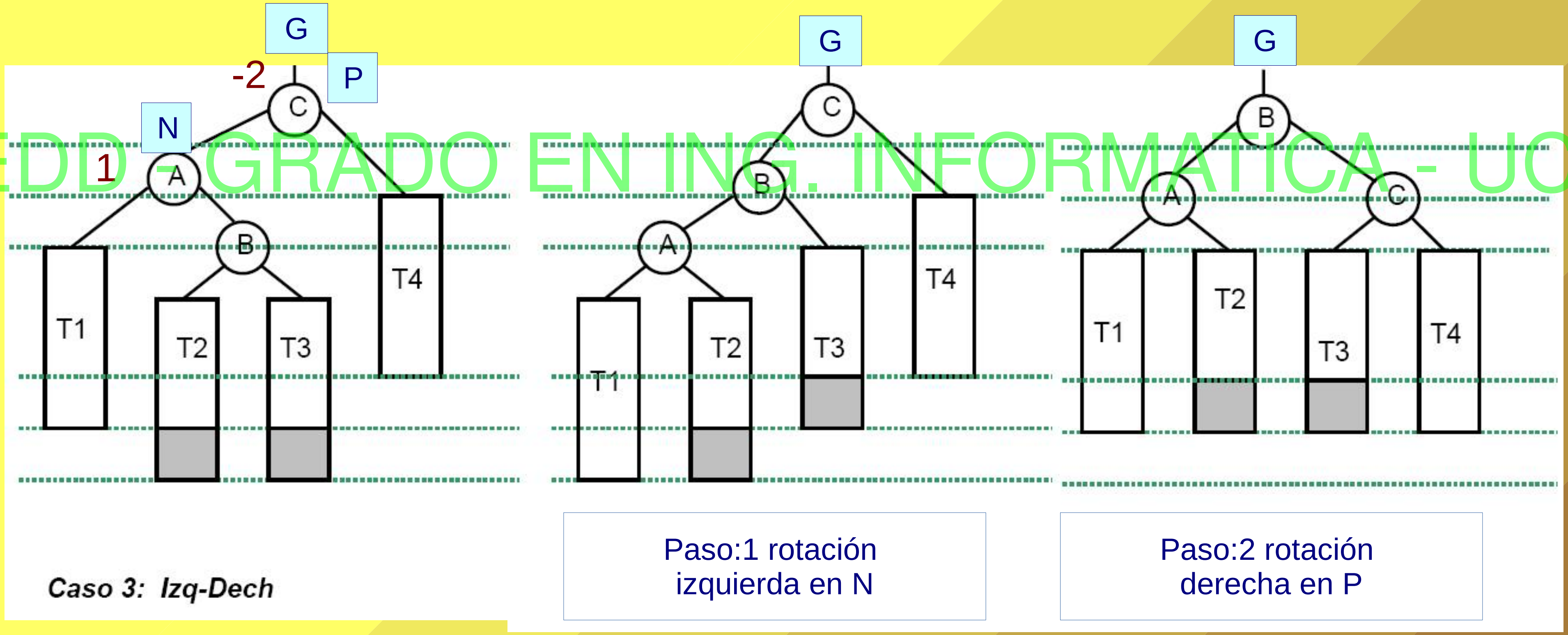
EEDD - GRADO EN ING. INFORMATICA - UCO

Desequilibrio der-der → rotación a izq. de P.



Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 3: izq-der -> rotación doble izq-der.



EEDD - GRADO EN ING. INFORMATICA - UCO

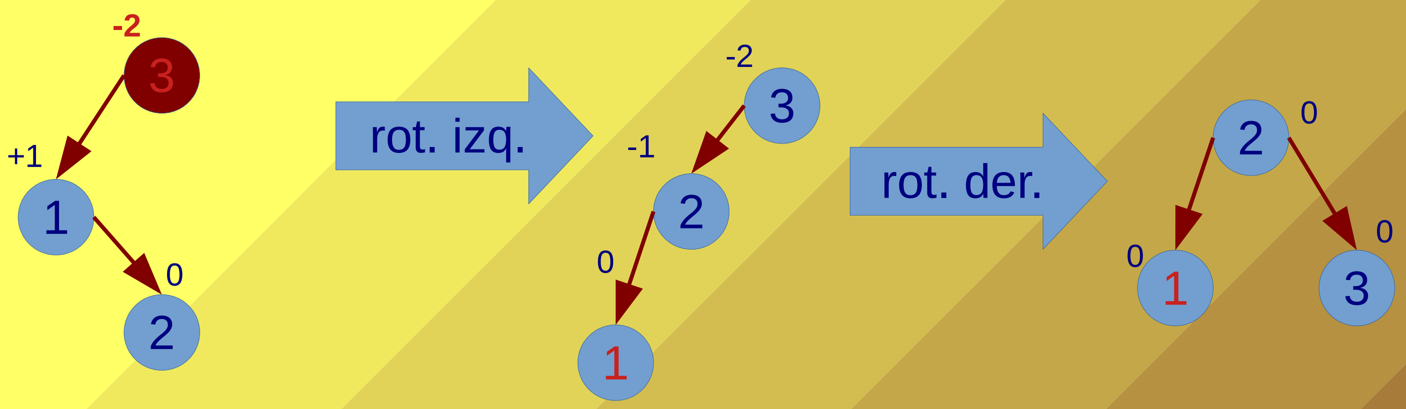
Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 3: ejemplo.

{3,1,**2**}

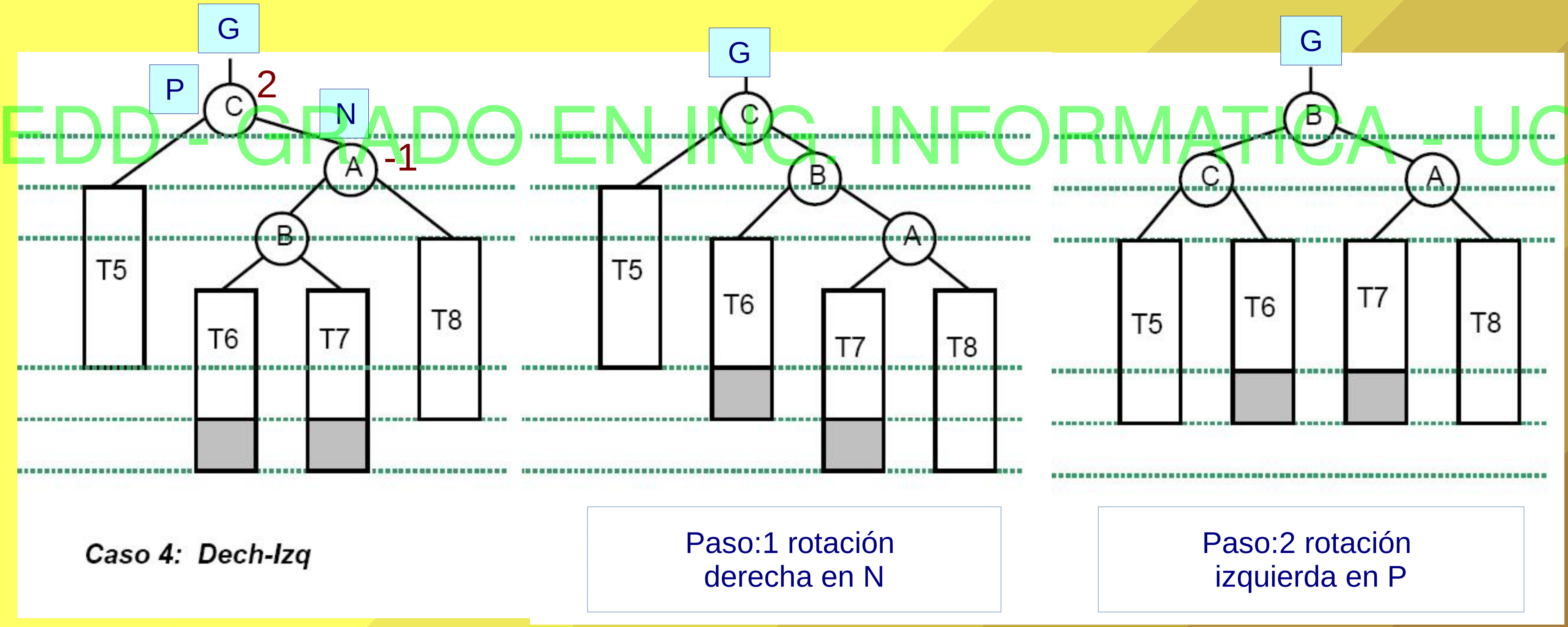
EEDD - GRADO EN ING. INFORMATICA - UCO

Desequilibrio izq-dercha → rotación a izq. de N + rot a der. de P.



Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 4: der-izq -> rotación doble der-izq.



EEDD - GRADO EN ING. INFORMATICA - UCO

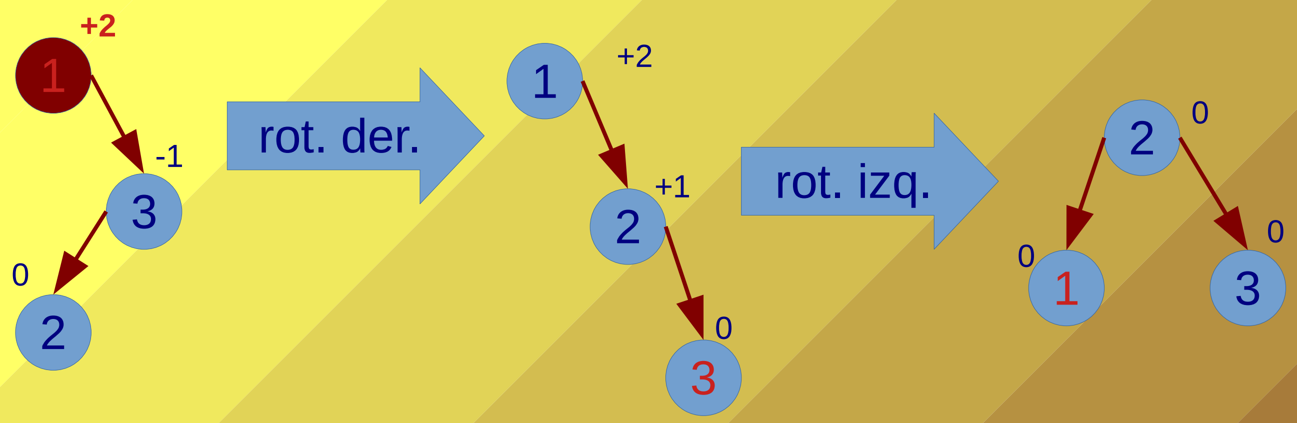
Árbol AVL

- Mantenimiento del equilibrio: Inserción.
  - Caso 4: ejemplo.

{1,3,**2**}

EEDD - GRADO EN ING. INFORMATICA - UCO

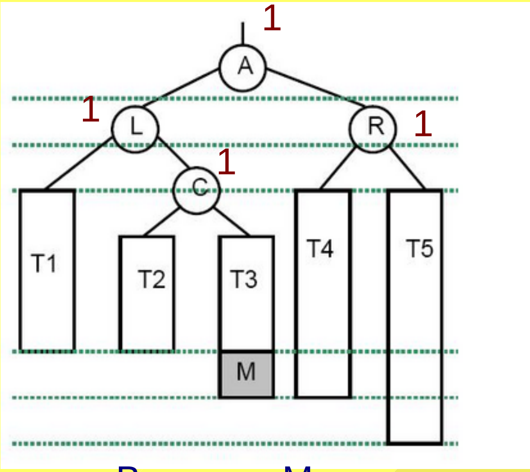
Desequilibrio der.-izq. → rotación a der. de N + rotación a izq. de P



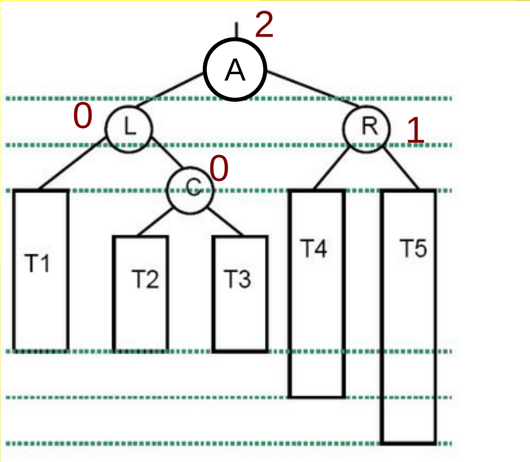
Árbol AVL

- Mantenimiento del equilibrio: Borrado.
  - Mismos casos que en la inserción.
  - Puede ser necesario aplicar más de una rotación.
  - En el peor de los casos H*O(1) -> O(H).

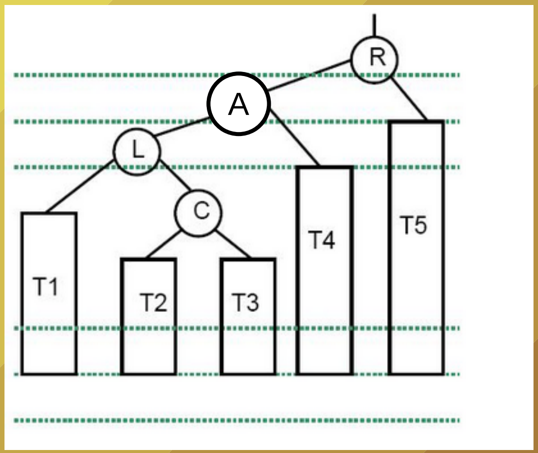
EEDD - GRADO EN ING. INFORMATICA - UCO



Borrar en M.



Desequilibrio (Caso 2)



Rotación simple izquierda.

Árbol AVL

- Algoritmo “makeBalanced”.

```
Algorithm AVLTree[T]::makeBalanced()
Var
  P:AVLTNode[T] #The root of the subtree
  N:AVLTNode[T] #The child to promote
  bfP: Integer #Balanced factor of parent
  bfN: Integer #Balanced factor of child.
  dir: Integer #rotate direction
Begin
  P <- parent_ #From cursor position to root.
  While P<>Void Do
    P.updateHeight()
    bfP <- P.balance_factor()
    If abs(bfP)>1 Then #subtree is unbalanced
      dir ← bfP<0 ? 0 : 1
      N ← P.child(dir)
      bfN ← N.balanceFactor()
      If bfP*bfN >= 0 Then #cases 1 or 2
         P <- rotate(P, 1-dir)
      Else #cases 3 or 4
         rotate(N, dir)
         P <- rotate(P, 1-dir)
      End-If
   End-If
   P <- P.parent()
  End-While
End.
```

O(   )

EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Recorrido en orden: iterador.

TAD: AVLTreeIterator[T]

Observers:
- isValid():Bool //Is pointing to a valid position in the tree?
- get():T //Get the value pointed by the iterator.
  - Pre-c: isValid()
- operator =(other:BSTreeIterator[T]):Bool //is other equal to this?

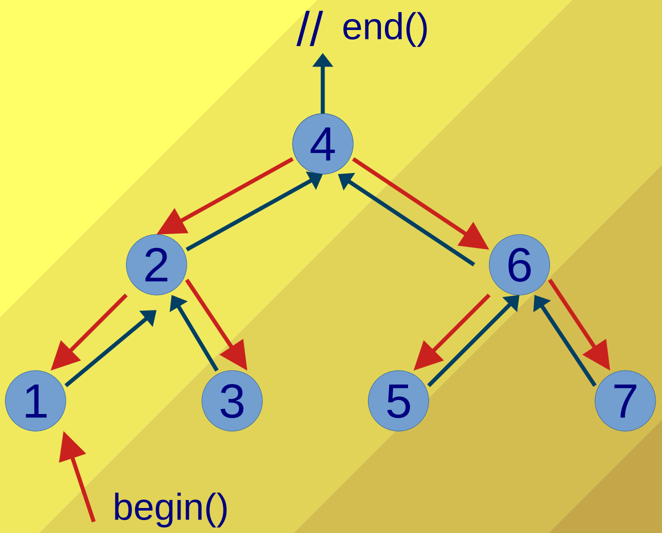
Modifiers:
- next() // goto inorder sucessor.
  - Pre-c: isValid()
- prev() // goto inorder predecessor.
  - Pre-c: isValid()

¿por qué no hay set()?

```
AVLTreeIterator[T]
tree_:AVLTree[T]
node_:AVLTNode[T]
```

EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL



- Iterador: creación.

TAD: AVLTree[T]

Observers:
- begin():AVLTreeIterator[T]
  - //Get an iterator at the begin position.
- end():AVLTreeIterator[T]
  - //Get an iterator at the end position.
  - Pos-c: not isEmpty() or begin()=end()

EEDD - GRADO EN ING. INFORMATICA - UCO

```
Algorithm AVLTreeIterator[T]::operator=(
          o:AVLTreeIterator[T]): Bool
Begin
   Return tree_ = o.tree_ AND
          node_ = o.node_
End.
```

```
Algorithm AVLTree[T]::end():AVLTreeIterator[T]
Var iter:AVLTreeIterator[T]
Begin
   iter.tree_ := This
   iter.node_ := Void
   Return iter
End.
```

```
Algorithm AVLTree[T]::begin():AVLTreeIterator[T]
Var iter:AVLTreeIterator[T]
Begin
   iter.tree_ := This
   iter.node_ := root_
   While iter.node_ <> Void Do
      iter.node_ := iter.node_.left()
   Return iter
End.
```

O(h)

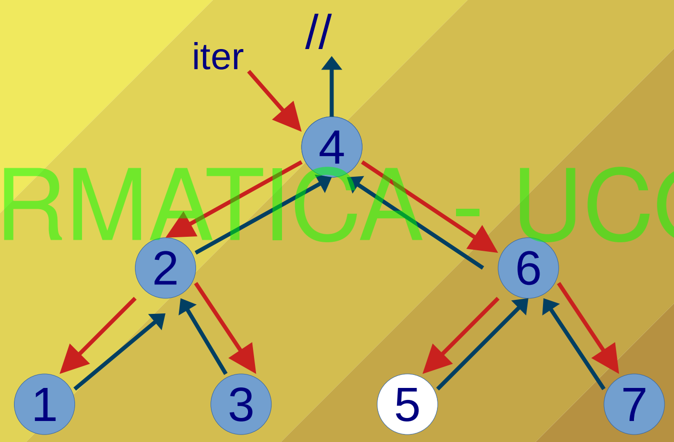
Árbol AVL

- Iterador: algoritmo next() (caso 0)
  - Hay sub-árbol derecho.

O(h)

EEDD - GRADO EN ING. INFORMATICA - UCO

```
If node_.right()<>Void Then
   node_ := node_.right()
   While node_.left()<>Void Do
      node_ := node_.left()
```



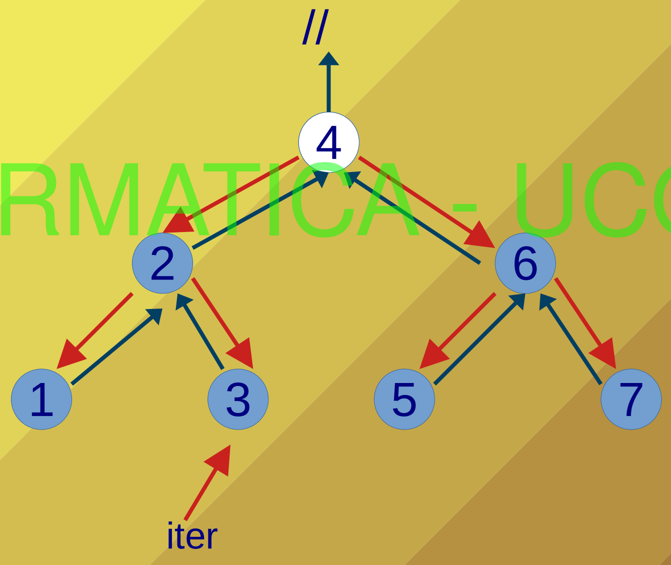
Árbol AVL

- Iterador: algoritmo next() (caso 1)
  - No hay sub-árbol derecho.

EEDD - GRADO EN ING. INFORMATICA - UCO

O(h)

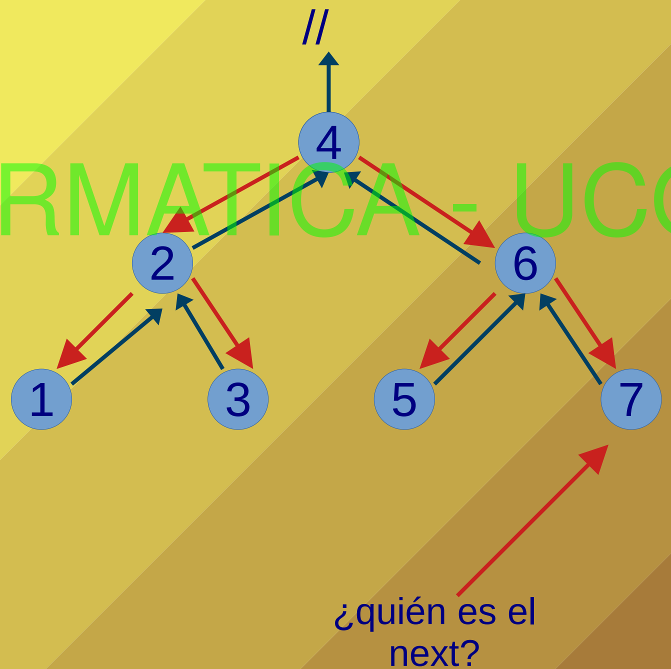
```
// subir niveles hasta que el nodo actual sea
// un hijo izquierdo y por lo tanto el padre
// es el siguiente en orden o sea Void.

      parent := node_->parent()
      While parent <> Void And
            node_ = parent.right() Do
         node_ := parent
         parent := node_.parent()
      End-While
      node_ := parent
```



Árbol AVL

- Iterador: algoritmo next().

```
Algorithm AVLTreeIterator[T]::next()          O(  )
Var parent:BSTNode[T]
Begin
   If node_.right()<>Void Then
      // Caso 0.
      node_ := node_.right()
      While node_.left()<>Void Do
         node_ := node_.left()
   Else
      // Caso 1.
      parent := node_->parent()
      While parent <> Void And
            node_ = parent.right() Do
         node_ := parent
         parent := node_.parent()
      End-While
      node_ := parent
   End-If
End.
```



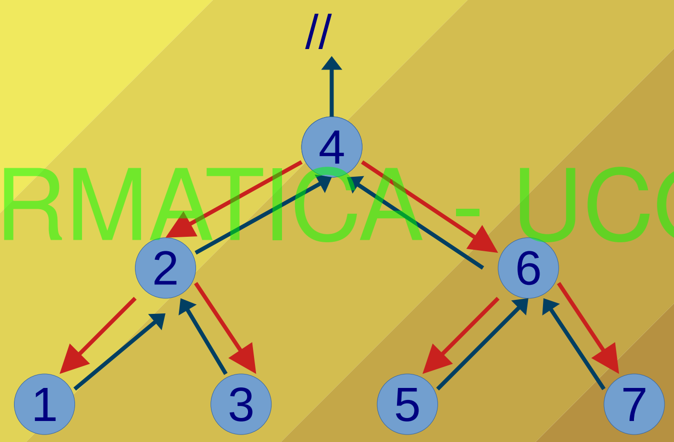
EEDD - GRADO EN ING. INFORMATICA - UCO

Árbol AVL

- Iterador: algoritmo prev().

```
Algorithm AVLTreeIterator[T]::prev()
Var parent:BSTNode[T]
Begin
   If node_.left()<>Void Then
      node_ := node_.left()
      While node_.right()<>Void Do
         node_ := node_.right()
   Else
      parent := node_->parent()
      While parent <> Void And
            node_ = parent.left() Do
         node_ := parent
         parent := node_.parent()
      End-While
      node_ := parent
   End-If
End.
```

O (?)



EEDD - GRADO EN ING. INFORMATICA - UCO

Resumen

- El BST puede degenerar con H >> $Log_2(N)$
- El árbol perfectamente equilibrados tiene $H=|Log_2(N)|$ pero mantener el equilibrio perfecto es complicado.
- Los árboles AVL son una solución de compromiso con $H \approx |Log_2(N)|$
- Un nodo estará equilibrado si su $|FE|<=1$
- Los desequilibrios se corrigen de abajo-arriba con operaciones de rotación O(1).

EEDD - GRADO EN ING. INFORMATICA - UCO

Referencias

- Lecturas recomendadas:
  - Apuntes de la asignatura (moodle).
  - Caps. 10, 11 y 12 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.
  - Caps 9 y 13.5 de *“Data structures and software develpment in an object oriented domain”*, Tremblay J.P. y Cheston, G.A. Prentice-Hall, 2001.
  - Wikipedia:
    - en.wikipedia.org/wiki/AVL_tree

EEDD - GRADO EN ING. INFORMATICA - UCO