Unidad 5: Árboles

Algoritmos y Estructuras de Datos

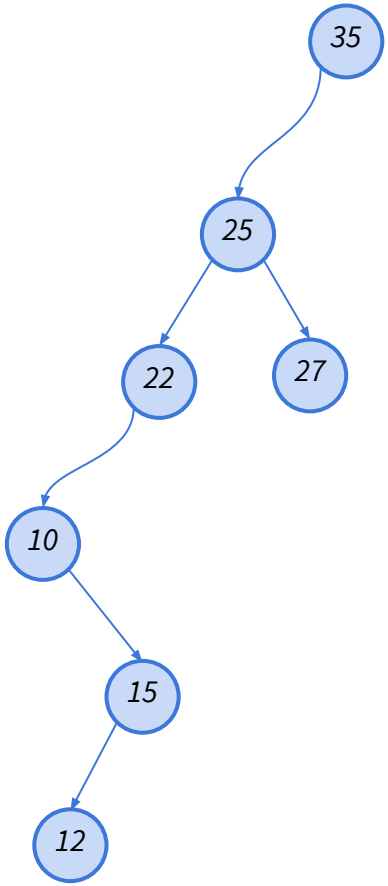
Árboles Binarios AVL

Ing. Juan Ignacio Iturriaga

Árboles Binarios Búsqueda (ABB)

Concepto de equilibrio

¿Sigue siendo un ABB?

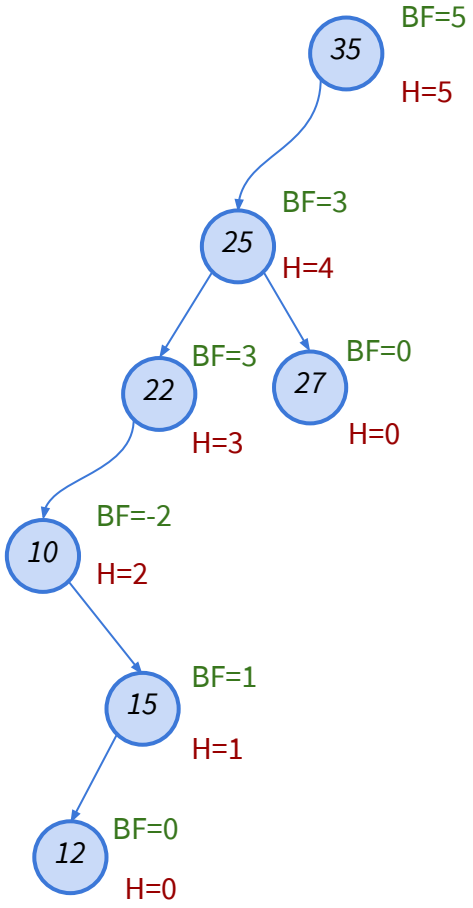


- ✓ Es de grado 2
- ✓ Para cada nodo todos sus descendientes de la rama izquierda tienen un valor menor.
- ✓ Para cada nodo todos sus descendientes de la rama derecha tiene un valor mayor.

¿Está equilibrado? ¿Por qué?

Árboles Binarios Búsqueda (ABB)

Definición de Altura y Factor de balanceo



Sea ***T*** un ABB y sean T_i y T_d sus subárboles, su **altura** H(***T***) es:

- -1, si el árbol ***T*** es nulo.
- 1 + max(H(T_i), H(T_d)), si contiene más nodos.

```
int height(btn *node){
    int result = -1;
    if (node != NULL) {
        result = max(height(node->left), height(node->right)) +1;
    }
    return result;
}
```

El **factor de equilibrio** (*Balance Factor*) de ***T*** es:

- BF(***T***) = H(T_i) - H(T_d)

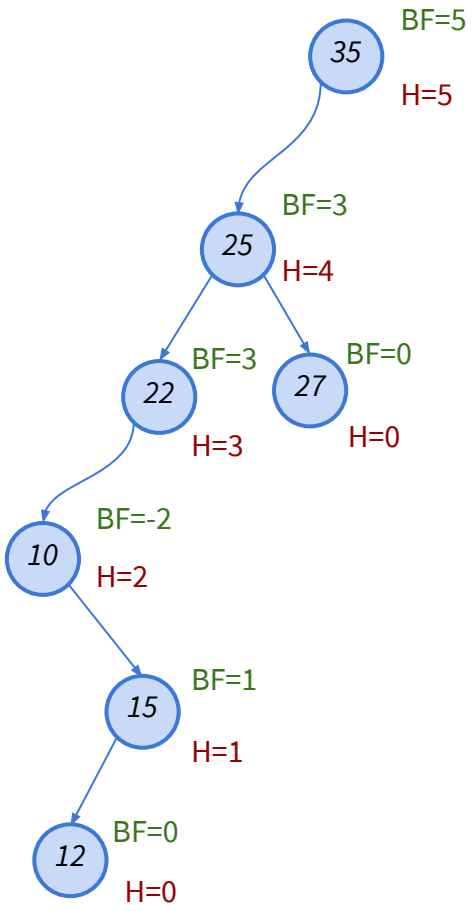
```
int balanceFactor(btn *node){
    int result = 0;
    if (node != NULL) {
        result = height(node->left) - height(node->right);
    }
    return result;
}
```

Árboles AVL

Concepto

T es un Árbol AVL si:

- ***T*** es nulo
- ***T*** no es nulo, T_i y T_d sus subárboles y:
  - T_i es AVL
  - T_d es AVL
  - | BF(T) | < 2



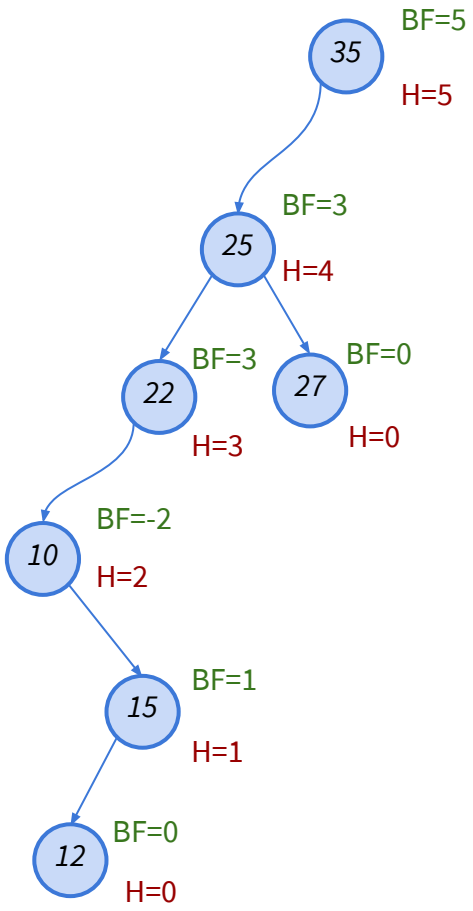

```
int isAVL(btn *node){

}
```

Árboles AVL

Concepto

T es un Árbol AVL si:

- ***T*** es nulo
- ***T*** no es nulo, T_i y T_d sus subárboles y:
  - T_i es AVL
  - T_d es AVL
  - | BF(T) | < 2



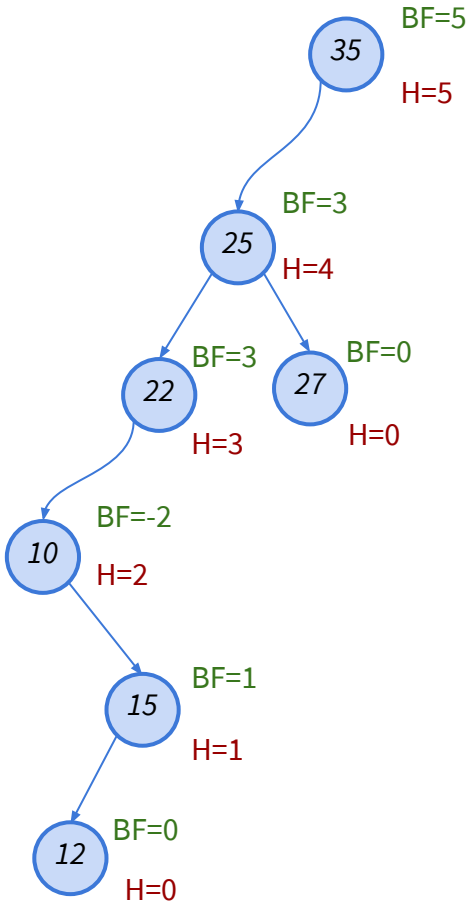

```
int isAVL(btn *node){

    if (abs(balanceFactor(node))<2) {

    } else {
        return 0;
    }
}
```

Árboles AVL

Concepto

T es un Árbol AVL si:

- ***T*** es nulo
- ***T*** no es nulo, T_i y T_d sus subárboles y:
  - T_i es AVL
  - T_d es AVL
  - | BF(T) | < 2



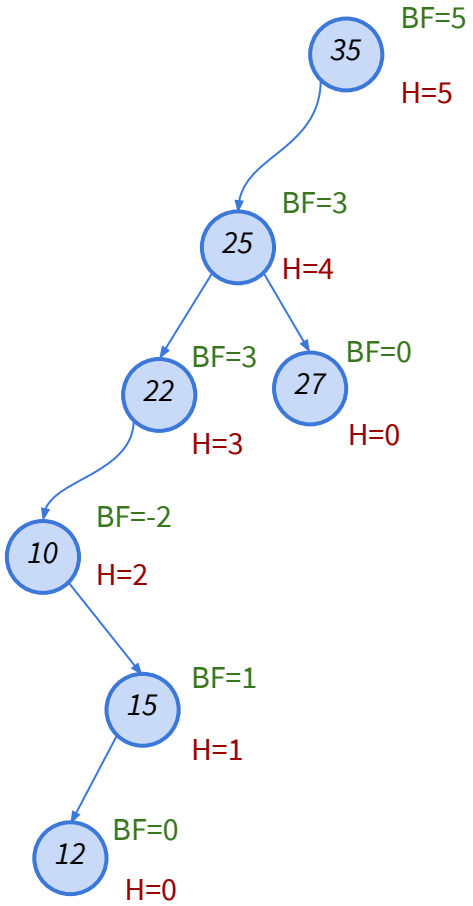

```
int isAVL(btn *node){


    if (abs(balanceFactor(node))<2) {
        if (isAVL(node->left)){

        } else {
            return 0;
        }
    } else {
        return 0;
    }
}
```

Árboles AVL

Concepto

T es un Árbol AVL si:

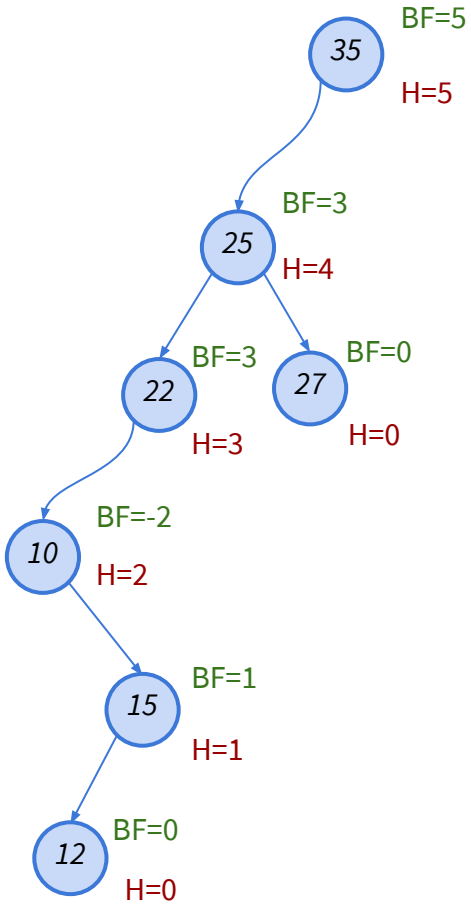
- ***T*** es nulo
- ***T*** no es nulo, T_i y T_d sus subárboles y:
  - T_i es AVL
  - T_d es AVL
  - | BF(T) | < 2




```
int isAVL(btn *node){


    if (abs(balanceFactor(node))<2) {
        if (isAVL(node->left)){
            return isAVL(node->right);
        } else {
            return 0;
        }
    } else {
        return 0;
    }
}
```

Árboles AVL

Concepto



T es un Árbol AVL si:

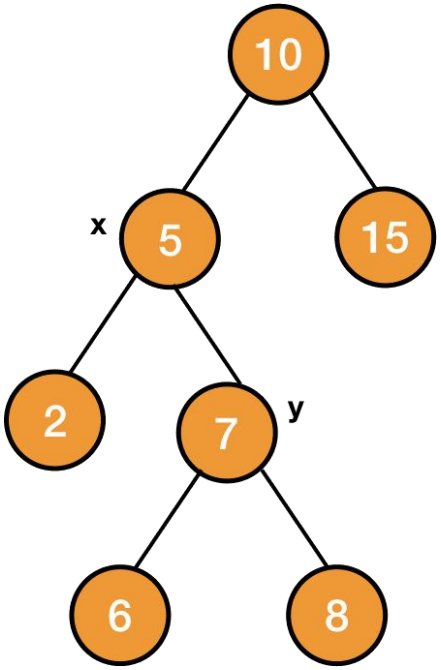
- ***T*** es nulo
- ***T*** no es nulo, T_i y T_d sus subárboles y:
  - T_i es AVL
  - T_d es AVL
  - | BF(T) | < 2

```
int isAVL(btn *node){
    if (node == NULL) return 1;

    if (abs(balanceFactor(node))<2) {
        if (isAVL(node->left)){
            return isAVL(node->right);
        } else {
            return 0;
        }
    } else {
        return 0;
    }
}
```

Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

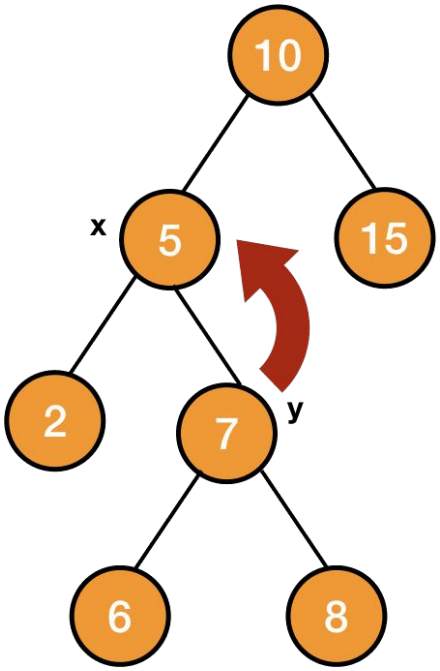


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***

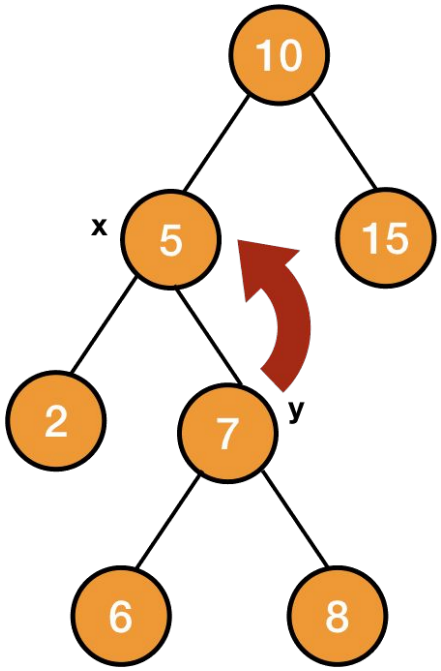


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol

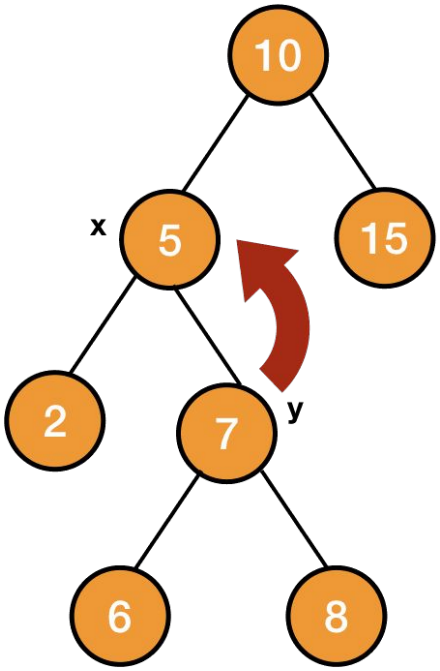


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***

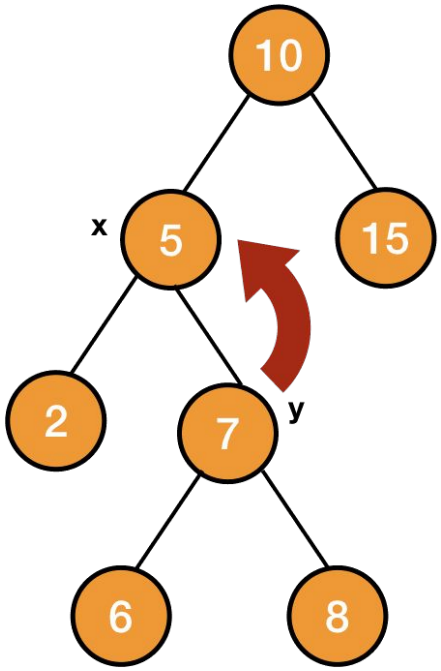


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***

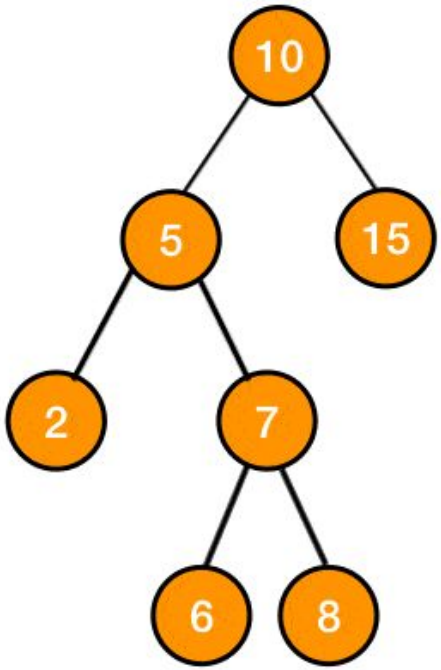


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***

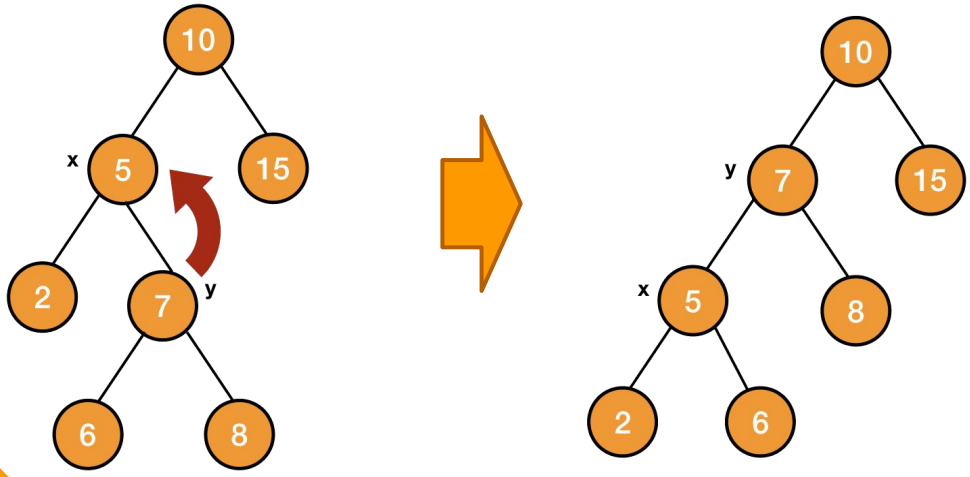


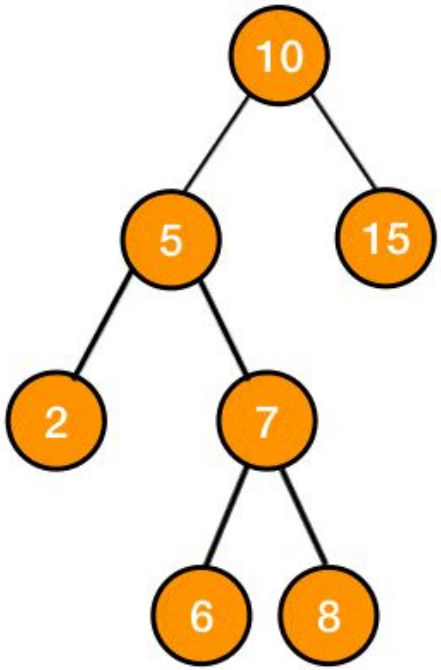
Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***



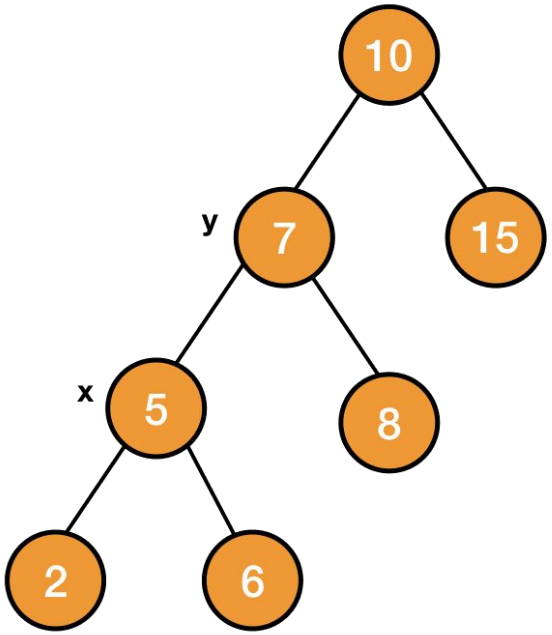


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***

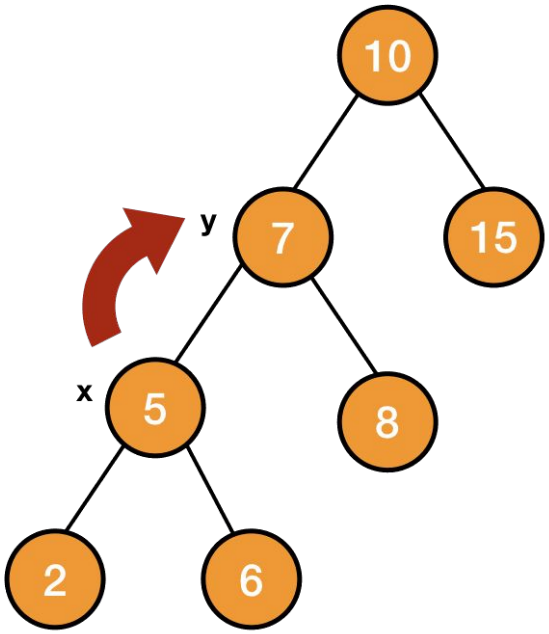


Árboles AVL

Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***



Rotar a derecha sobre el nodo ***y***

- ***y*** tiene un hijo izquierdo ***x***
- ***x*** se convertirá en la nueva raíz del subárbol
- el hijo derecho de ***x*** será ***y***
- el hijo derecho anterior de ***x*** será el nuevo hijo izquierdo de ***y***

Árboles AVL

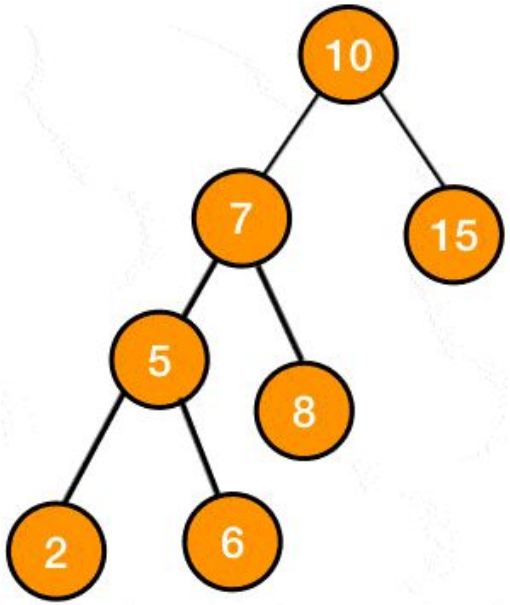
Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***

Rotar a derecha sobre el nodo ***y***

- ***y*** tiene un hijo izquierdo ***x***
- ***x*** se convertirá en la nueva raíz del subárbol
- el hijo derecho de ***x*** será ***y***
- el hijo derecho anterior de ***x*** será el nuevo hijo izquierdo de ***y***



Árboles AVL

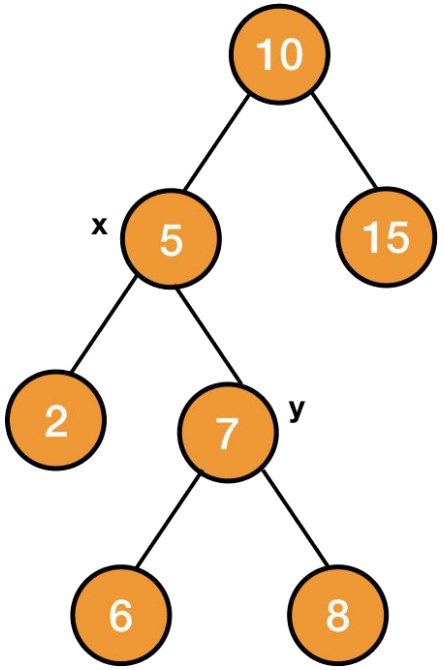
Rotaciones

Rotar a izquierda sobre el nodo ***x***

- ***x*** tiene un hijo derecho ***y***
- ***y*** se convertirá en la nueva raíz del subárbol
- el hijo izquierdo de ***y*** será ***x***
- el hijo izquierdo anterior de ***y*** será el nuevo hijo derecho de ***x***

Rotar a derecha sobre el nodo ***y***

- ***y*** tiene un hijo izquierdo ***x***
- ***x*** se convertirá en la nueva raíz del subárbol
- el hijo derecho de ***x*** será ***y***
- el hijo derecho anterior de ***x*** será el nuevo hijo izquierdo de ***y***

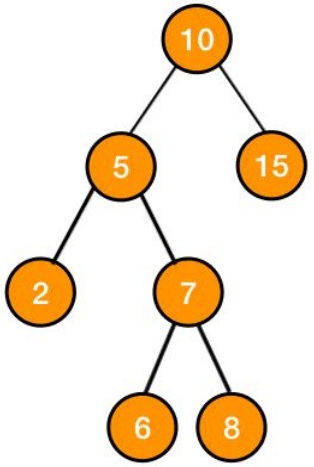


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){




    return 1;
}
```

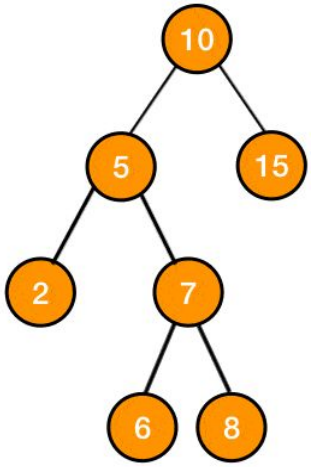


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){


    btn *aux = (*node)->right;


    return 1;
}
```

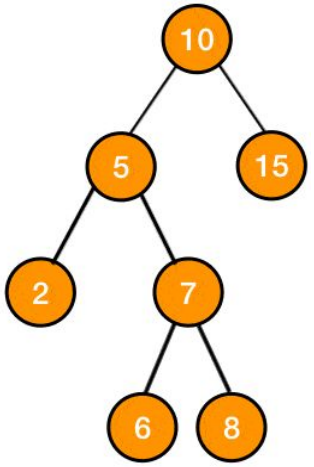


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){


    btn *aux = (*node)->right;
    (*node)->right = aux->left;


    return 1;
}
```

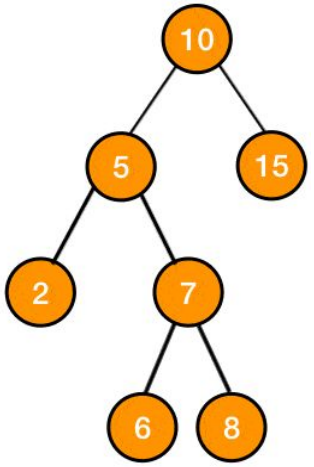


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){



    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;


    return 1;
}
```

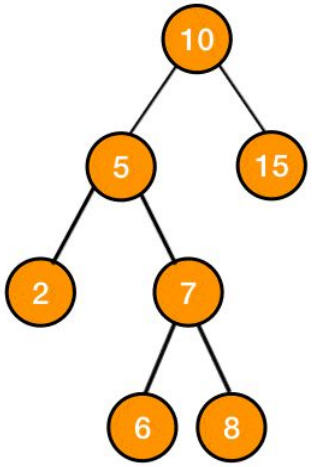


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){



    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;


    return 1;
}
```

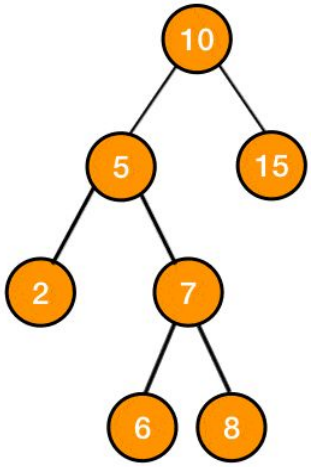


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){



    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;
    *node = aux;

    return 1;
}
```

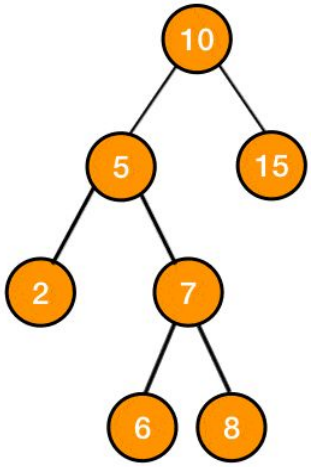


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){


    if ((*node)->right == NULL) return 0;

    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;
    *node = aux;

    return 1;
}
```

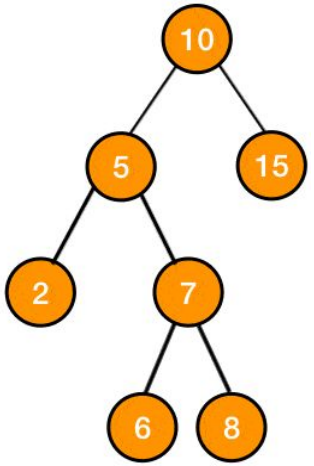


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;
    if ((*node)->right == NULL) return 0;

    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;
    *node = aux;

    return 1;
}
```

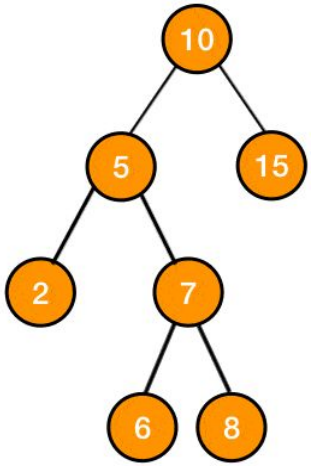


Árboles AVL

Rotaciones - algoritmos

Rotar a izquierda

```
int leftRotation (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;
    if ((*node)->right == NULL) return 0;

    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;
    *node = aux;

    return 1;
}
```

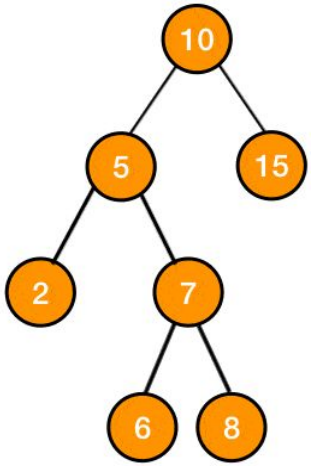


Árboles AVL

Rotaciones - algoritmos

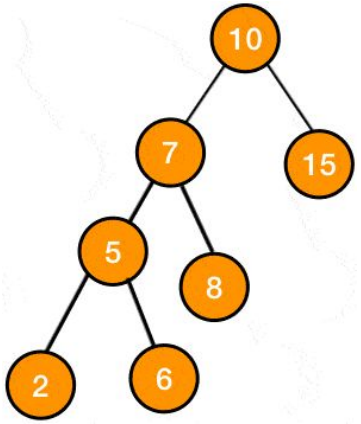
Rotar a izquierda

```
int leftRotation (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;
    if ((*node)->right == NULL) return 0;

    btn *aux = (*node)->right;
    (*node)->right = aux->left;
    aux->left = *node;
    *node = aux;

    return 1;
}
```



Rotar a derecha

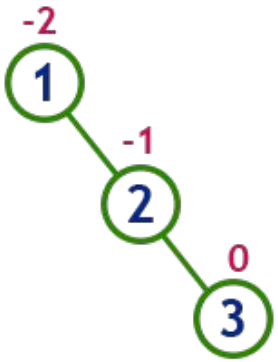
```
int rightRotation (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;
    if ((*node)->left == NULL) return 0;

    btn *aux = (*node)->left;
    (*node)->left = aux->right;
    aux->right = *node;
    *node = aux;

    return 1;
}
```



Árboles AVL

Situaciones de Rotación

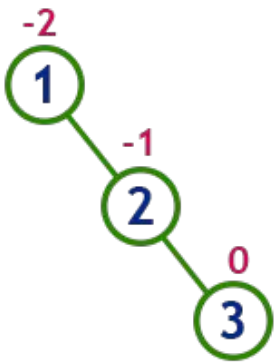
Insertamos: 1, 2, 3



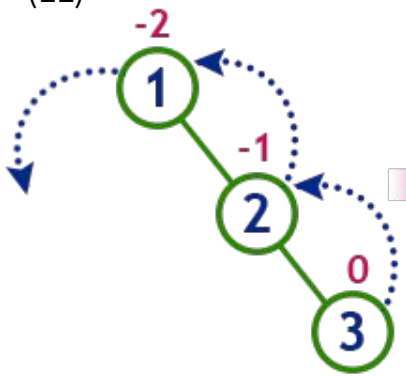
Árboles AVL

Situaciones de Rotación

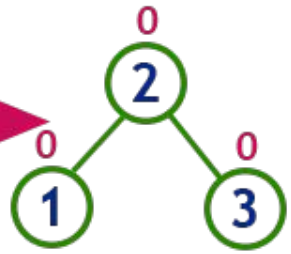
Insertamos: 1, 2, 3



Rotación a Izq. en 1 (LL)



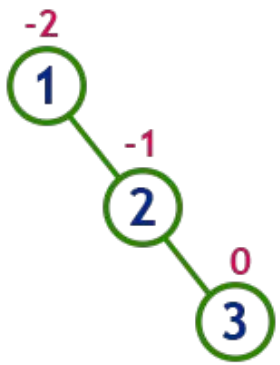
Balanceado



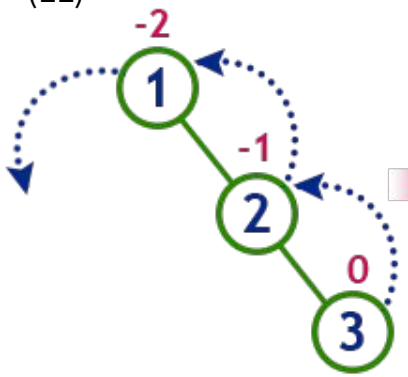
Árboles AVL

Situaciones de Rotación

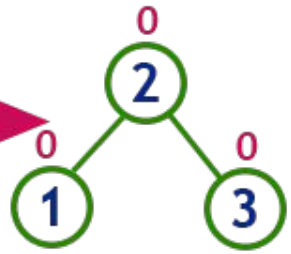
Insertamos: 1, 2, 3



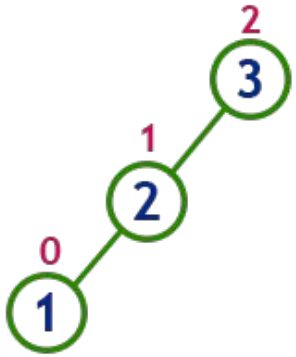
Rotación a Izq. en 1 (LL)



Balanceado



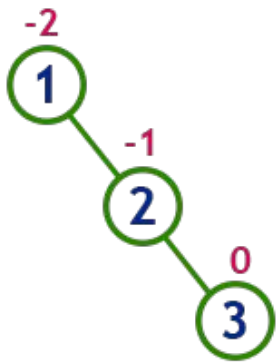
Insertamos: 3, 2, 1



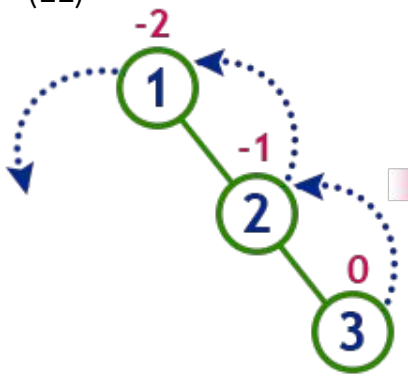
Árboles AVL

Situaciones de Rotación

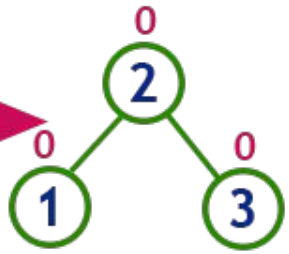
Insertamos: 1, 2, 3



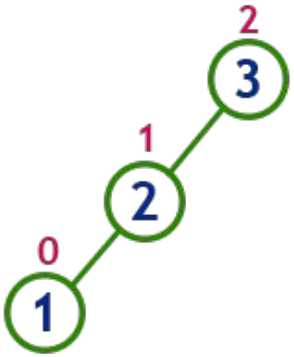
Rotación a Izq. en 1 (LL)



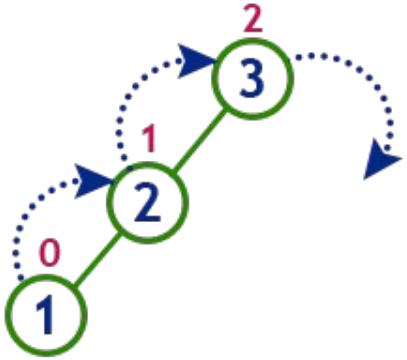
Balanceado



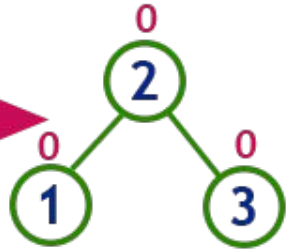
Insertamos: 3, 2, 1



Rotación a Der. en 3 (RR)



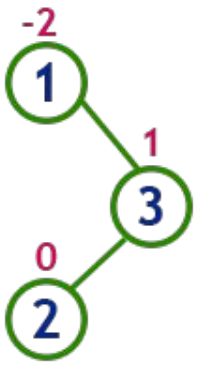
Balanceado



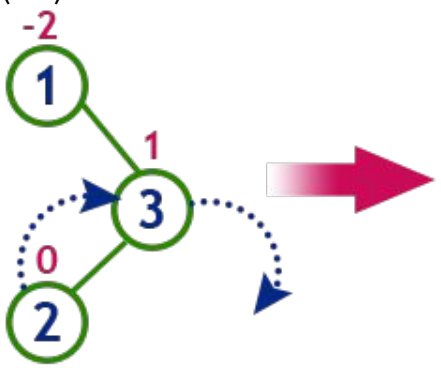
Árboles AVL

Situaciones de Rotación

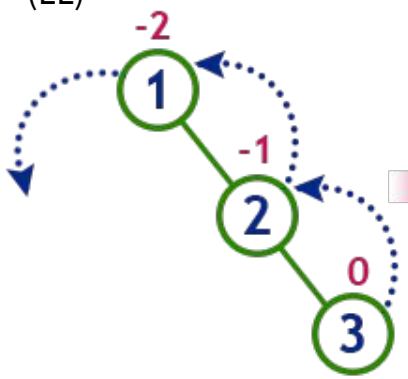
Insertamos: 1, 3, 2



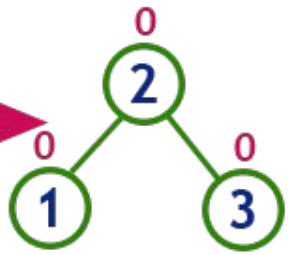
Rotación a Der. en 3 (RR)



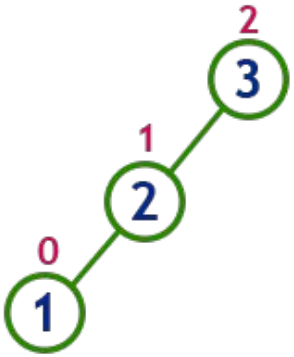
Rotación a Izq. en 1 (LL)



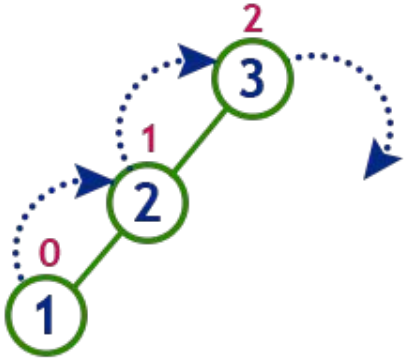
Balanceado



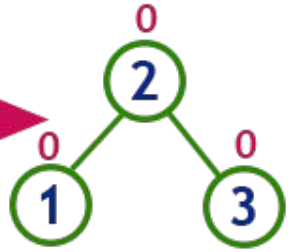
Insertamos: 3, 2, 1



Rotación a Der. en 3 (RR)



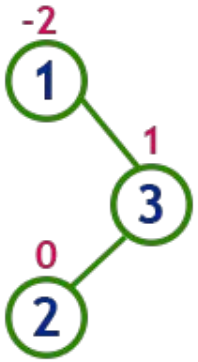
Balanceado



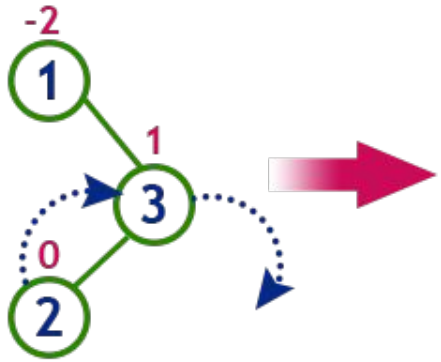
Árboles AVL

Situaciones de Rotación

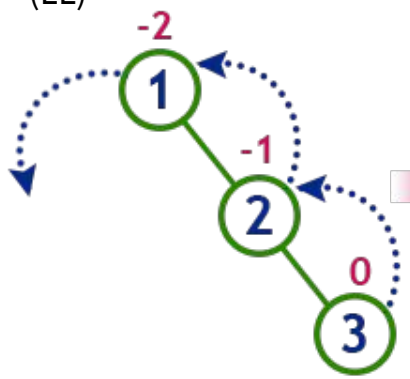
Insertamos: 1, 3, 2



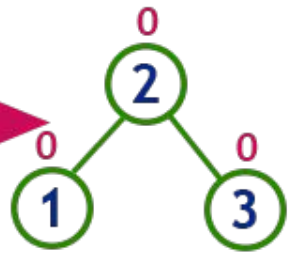
Rotación a Der. en 3 (RR)



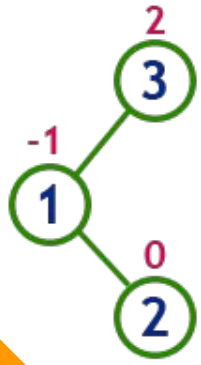
Rotación a Izq. en 1 (LL)



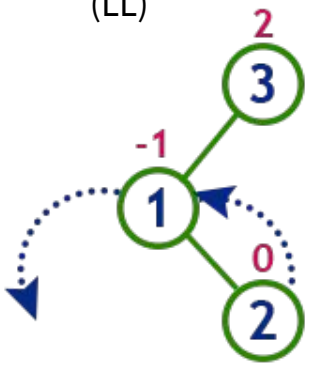
Balanceado



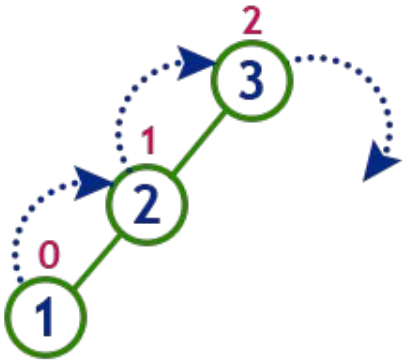
Insertamos: 3, 1, 2



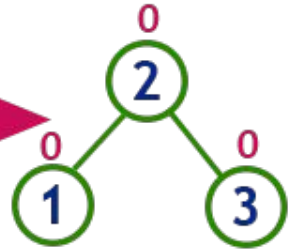
Rotación a Izq. en 1 (LL)



Rotación a Der. en 3 (RR)



Balanceado



Árboles AVL

Situaciones de Rotación

Si BF(**n**) = -2

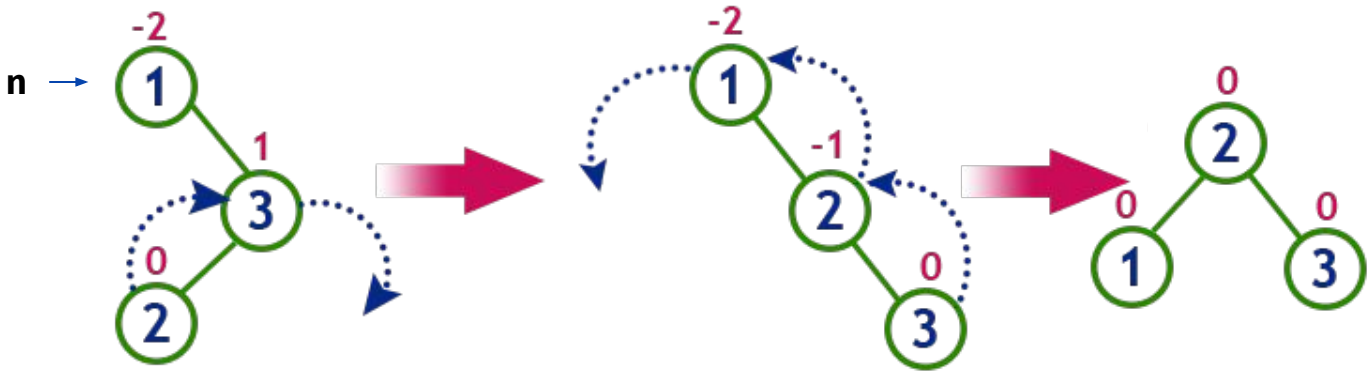
Si BF(**n.right**) > 0

Rotación a Der. en **n.right**

luego:

Rotación a Izq. en **n**

Balanceado



Si BF(**n**) = 2

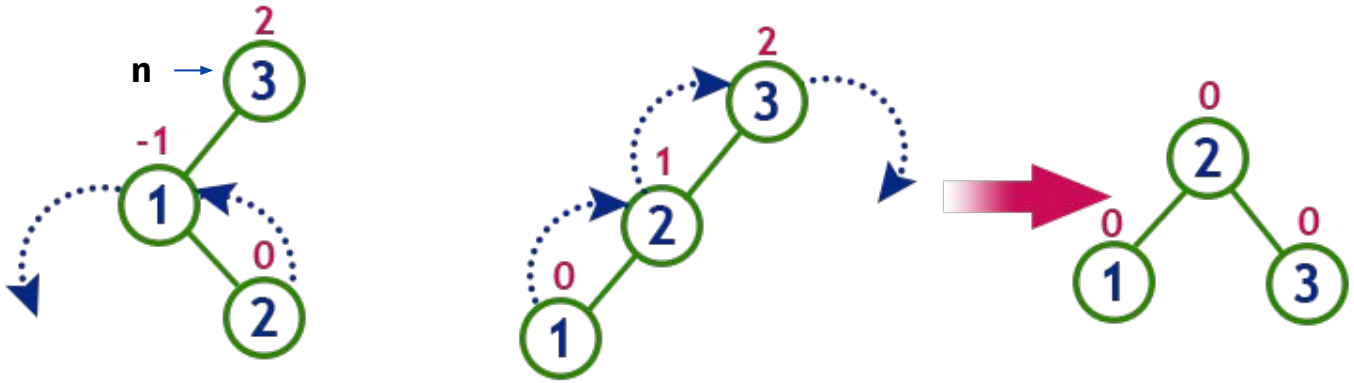
Si BF(**n.left**) < 0

Rotación a Izq. en **n.left**

luego:

Rotación a Der. en **n**

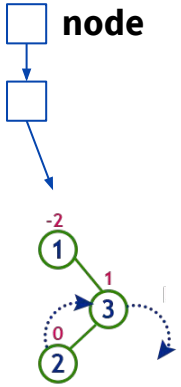
Balanceado



Árboles AVL

Algoritmo para balancear un nodo

```
int balance (btn **node){
```

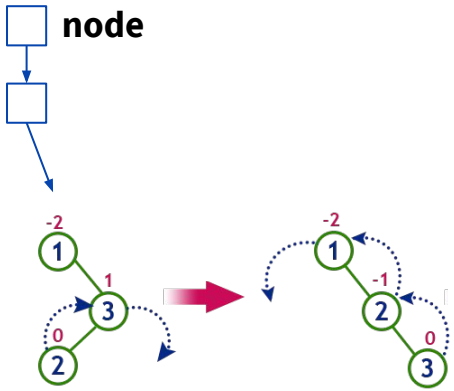



```
    return 1;
}
```

Árboles AVL

Algoritmo para balancear un nodo

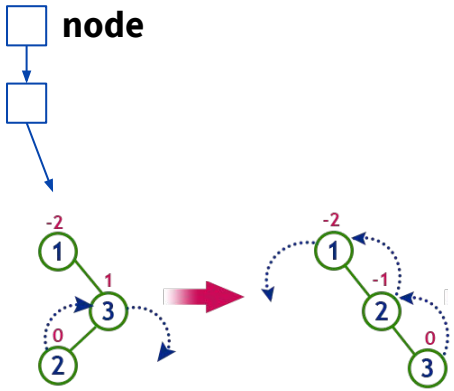
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }









    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

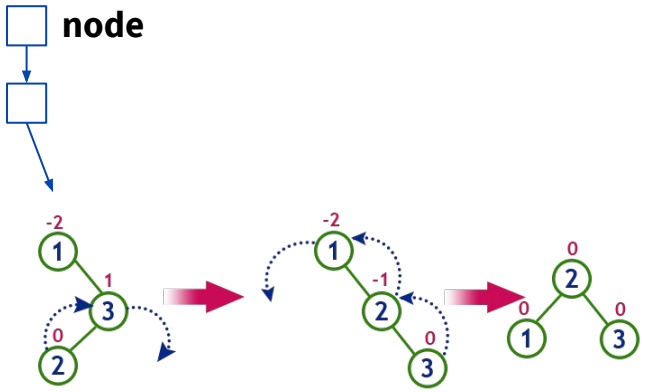
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }









    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

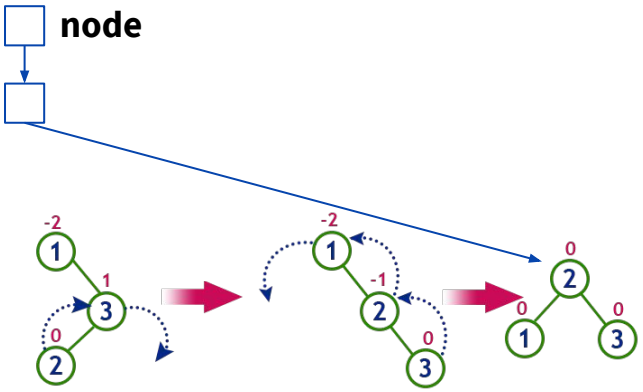
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));









    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

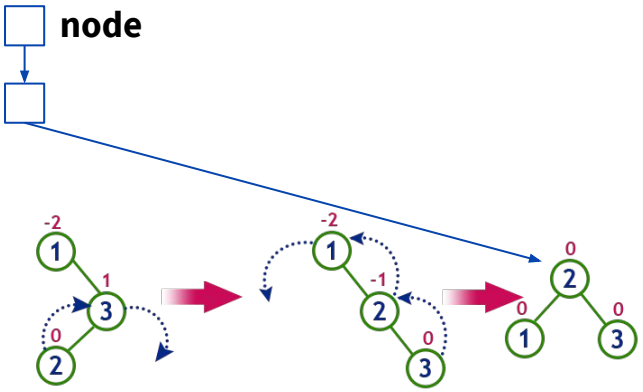
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));









    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

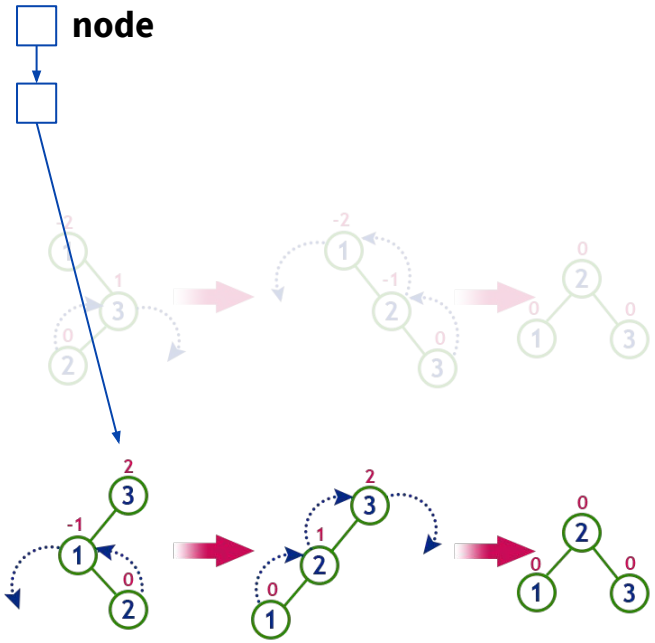
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));









    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

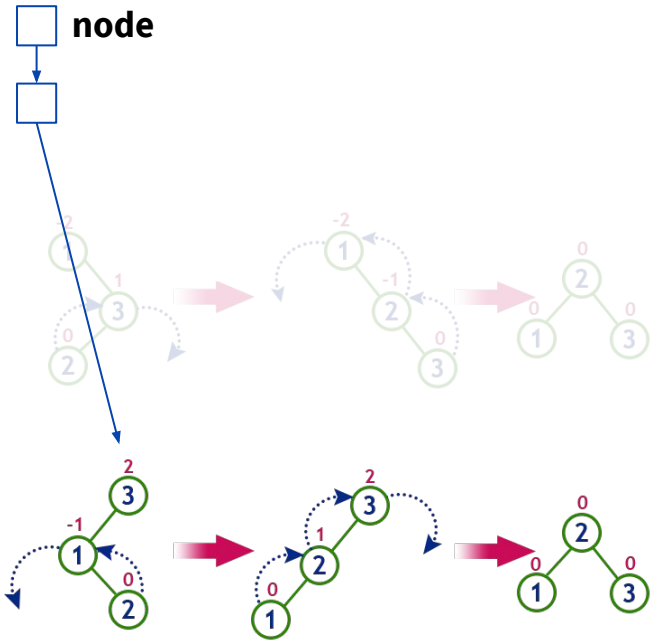
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));


    } else if (bf >= 2) {




    }



    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

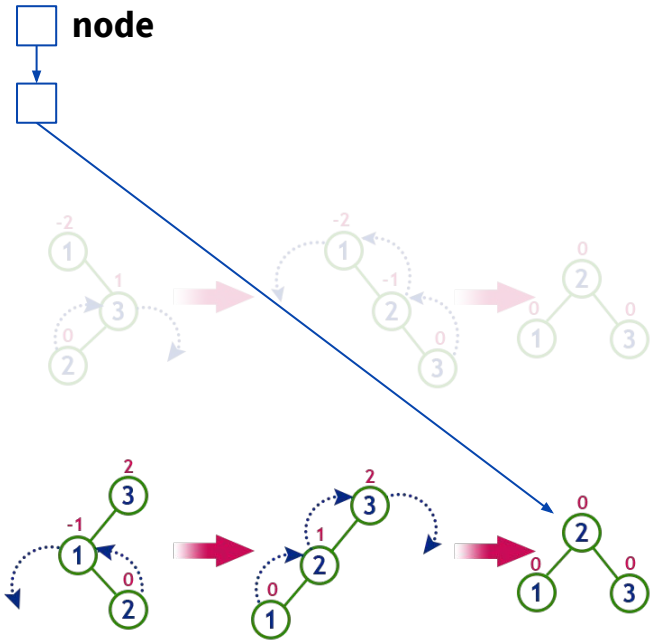
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));


    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }

    }




    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

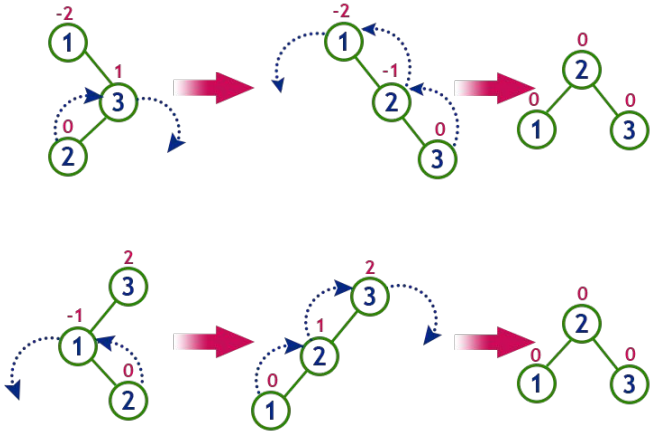
```
int balance (btn **node){



    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));


    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(&(*node));
    }




    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

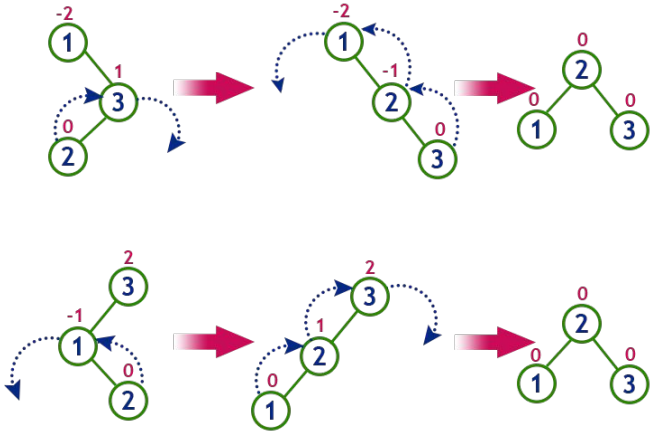
```
int balance (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));


    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(&(*node));
    }




    return 1;
}
```



Árboles AVL

Algoritmo para balancear un nodo

```
int balance (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));


    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(&(*node));
    }

    if (abs(balanceFactor(*node)) > 1){
        balance (&(*(node)));
    }

    return 1;
}
```



Árboles AVL

Algoritmo para balancear un árbol

```
int balance (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));

    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(&(*node));
    }

    if (abs(balanceFactor(*node)) > 1){
        balance (&(*(node)));
    }

    return 1;
}
```

```
int balanceTree (btn **node){

    balanceTree (&((*node)->left));
    balanceTree (&((*node)->right));
    balance (&(*node));

    return 1;
}
```

Árboles AVL

Algoritmo para balancear un árbol

```
int balance (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(&(*node));

    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(&(*node));
    }

    if (abs(balanceFactor(*node)) > 1){
        balance (&(*(node)));
    }

    return 1;
}
```

```
int balanceTree (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    balanceTree (&((*node)->left));
    balanceTree (&((*node)->right));
    balance (&(*node));

    return 1;
}
```

Árboles AVL

Algoritmo para balancear un árbol

```
int balance (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    int bf = balanceFactor(*node);

    if (bf <= -2) {
        if (balanceFactor((*node)->right) >0) {
            rightRotation(&((*node)->right));
        }
        leftRotation(node);

    } else if (bf >= 2) {
        if (balanceFactor((*node)->left) <0) {
            leftRotation(&((*node)->left));
        }
        rightRotation(node);
    }

    if (abs(balanceFactor(*node)) > 1){
        balance (node);
    }

    return 1;
}
```

```
int balanceTree (btn **node){
    if (node == NULL) return 0;
    if (*node == NULL) return 0;

    balanceTree (&((*node)->left));
    balanceTree (&((*node)->right));
    balance (node);

    if (!isAVL(*node)){
        balanceTree(node);
    }

    return 1;
}
```

Árboles AVL

Algoritmos para insertar y eliminar (versión simplificada)

```
int insertAVL (btn **node, int value) {

        btn *new = create(value);
        if (!insertNode(node, new)) {
            free(new); //Duplicado
        } else {
            balanceTree (node);
        };

}
```

```
    }
}
```

Árboles AVL

Algoritmos para insertar y eliminar (versión simplificada)

```
int insertAVL (btn **node, int value) {

        btn *new = create(value);
        if (!insertNode(node, new)) {
            free(new); //Duplicado
        } else {
            balanceTree (node);
        };

}


int insertNode(btn **node, btn *new) {
    if (node == NULL) return 0;
    if (new == NULL) return 1;

    if ((*node) == NULL) {
        *node = new;
        return 1;
    } else {
        if ((*node)->value > new->value) {
            return insertNode(&((*node)->left), new);

        } else if ((*node)->value < new->value) {
            return insertNode(&((*node)->right), new);

        } else {
            return 0;
        }
    }
}
```

Árboles AVL

Algoritmos para insertar y eliminar (versión simplificada)

```
int insertAVL (btn **node, int value) {

        btn *new = create(value);
        if (!insertNode(node, new)) {
            free(new); //Duplicado
        } else {
            balanceTree (node);
        };

}
```

```
int removeAVL (btn **node, int value){

      btn *deleted = NULL;
      btn **toDel = NULL;
      toDel = findPtr(value, node);
      deleted = removeNode(toDel);
      if (deleted != NULL) {
            balanceTree (node);
        };
    }
```

```
int insertNode(btn **node, btn *new) {
    if (node == NULL) return 0;
    if (new == NULL) return 1;

    if ((*node) == NULL) {
        *node = new;
        return 1;
    } else {
        if ((*node)->value > new->value) {
            return insertNode(&((*node)->left), new);

        } else if ((*node)->value < new->value) {
            return insertNode(&((*node)->right), new);

        } else {
            return 0;
        }
    }
}
```

Árboles AVL

Algoritmos para insertar y eliminar (versión simplificada)

```
int insertAVL (btn **node, int value) {

        btn *new = create(value);
        if (!insertNode(node, new)) {
            free(new); //Duplicado
        } else {
            balanceTree (node);
        };

}
```

```
int insertNode(btn **node, btn *new) {
    if (node == NULL) return 0;
    if (new == NULL) return 1;

    if ((*node) == NULL) {
        *node = new;
        return 1;
    } else {
        if ((*node)->value > new->value) {
            return insertNode(&((*node)->left), new);

        } else if ((*node)->value < new->value) {
            return insertNode(&((*node)->right), new);

        } else {
            return 0;
        }
    }
}
```

```
int removeAVL (btn **node, int value){

      btn *deleted = NULL;
      btn **toDel = NULL;
      toDel = findPtr(value, node);
      deleted = removeNode(toDel);
      if (deleted != NULL) {
            balanceTree (node);
        };
    }
```

```
btn *removeNode(btn **node) {
    if (!node) return NULL;
    if (!(*node)) return NULL;

    btn *aux = *node;

    insertNode(&(aux->right), (aux->left));
    *node = aux->right;
    aux->left = NULL;
    aux->right = NULL;

    return aux;
}
```

Unidad 5: Árboles

Algoritmos y Estructuras de Datos

Árboles Binarios AVL