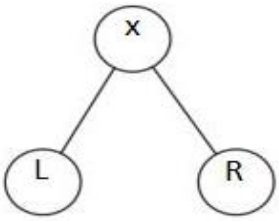
TAD - TIPO ABSTRACTO DE DATOS

TAD ÁRBOL BINARIO DE BÚSQUEDA

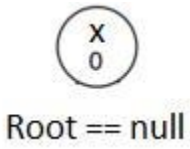
Representación:
Sea X, L, R nodos, donde L y R son sub árboles de X.

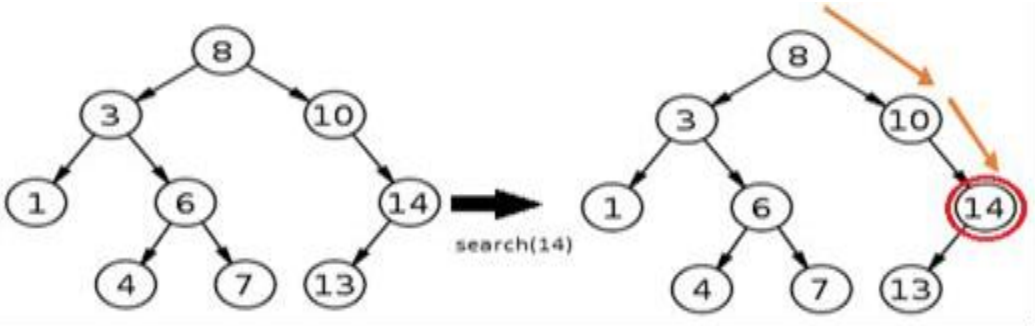


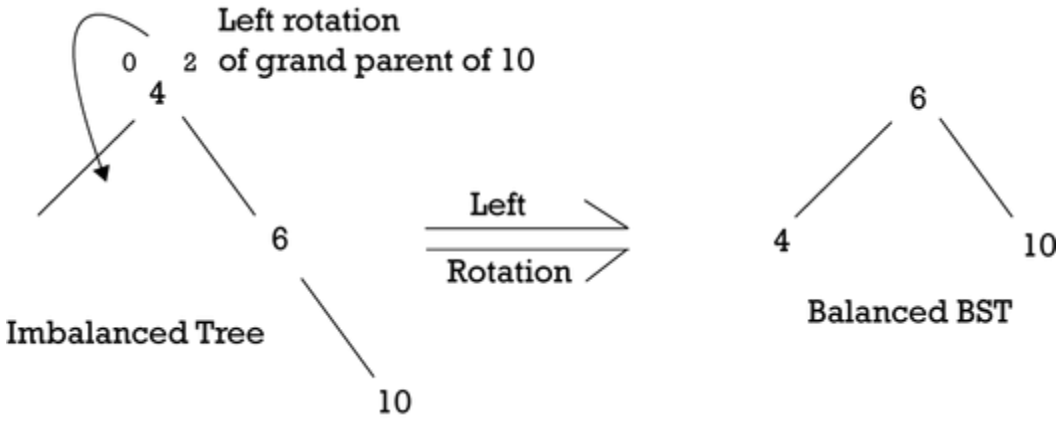
Invariante:
El sub árbol izquierdo de X contiene valores menos que X y el sub árbol derecho contiene los valores mayores que X.

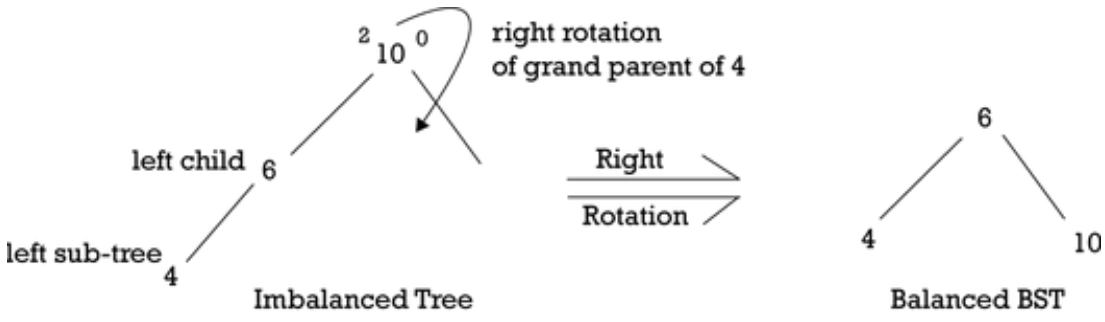
Operaciones:

CreateBSTree		→ BSTree
SearchNode	**Key**	→ BSTNode
RotationToLeft	**BSTNode**	→ void
RotationToRight	**BSTNode**	→ void
AddNode	**Key, Value**	→ BSTree
Remove	**Key**	→ Boolean

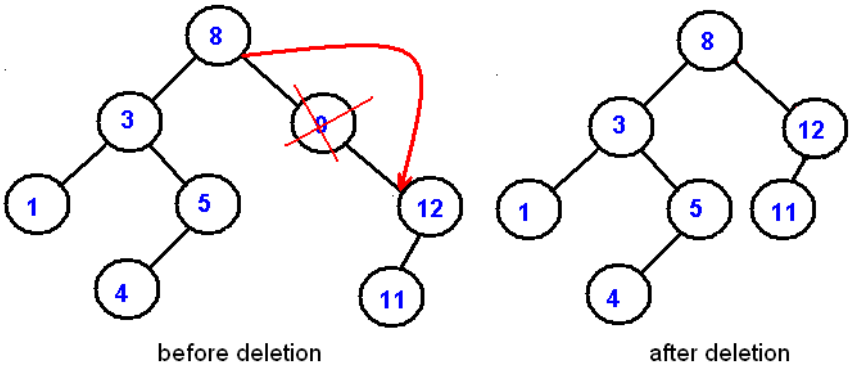
BSTree()
Pre: true
Post: El árbol es creado 

SearchNode(K key)
Pre: true
Post: Si la llave es encontrada, retorna el nodo con todos los valores asociados a ella. 

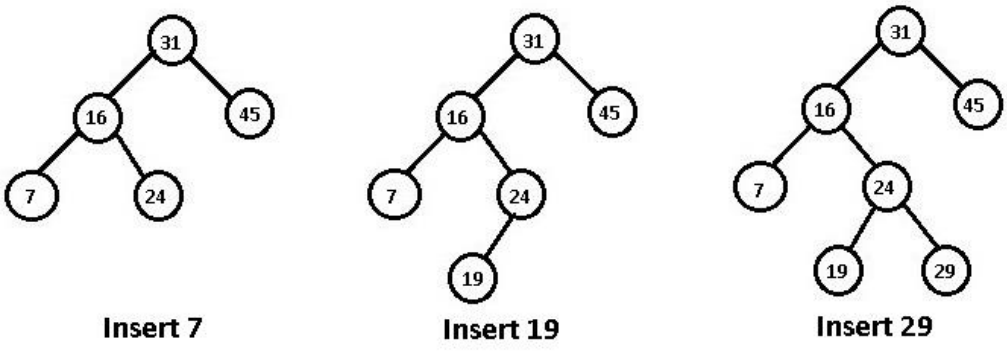
RotationToLeft(BSTNode<K, V> x)
Pre: x es diferente de null
Post: Si la llave existe, entonces la llave rota a la izquierda. 

RotationToRight(BSTNode<K, V> x)
Pre: x es diferente de null
Post: Si la llave existe, entonces la llave rota a la derecha. 

DeleteNode(K key)
Pre: true
Post: Si la llave es encontrada, se elimina el nodo con todos los valores asociados a ella.

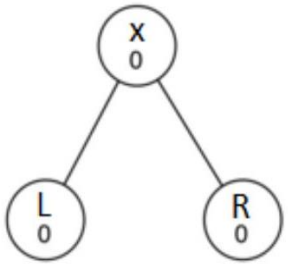


AddNode(AVLNode x)
Pre: x diferente de null
Post: Se agrega el nodo al árbol y se balancea completamente.



TAD ÁRBOL AVL

Representación:
Sea X, L, R nodos, donde L y R son sub árboles de X, y la altura de R menos la altura de L no difiere por mas de 1.



Invariante:
-1 <= | Factor de balanceo | <= 1

Operaciones:

CreateAVLTree		→ AVLTree
AddNode	AVLNode	→ void
DeleteNode	Key	→ Boolean

AVLTree()

Pre: true

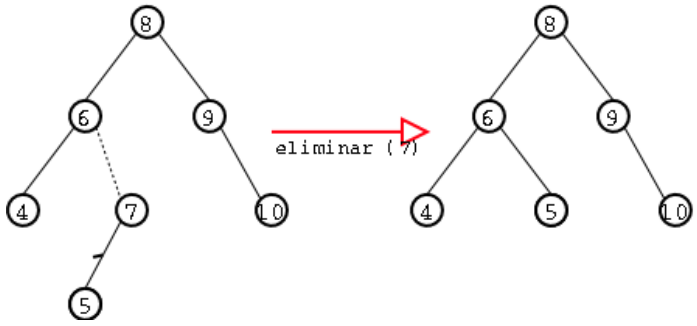
Post: El árbol es creado

Root == null

DeleteNode(K key)

Pre: true

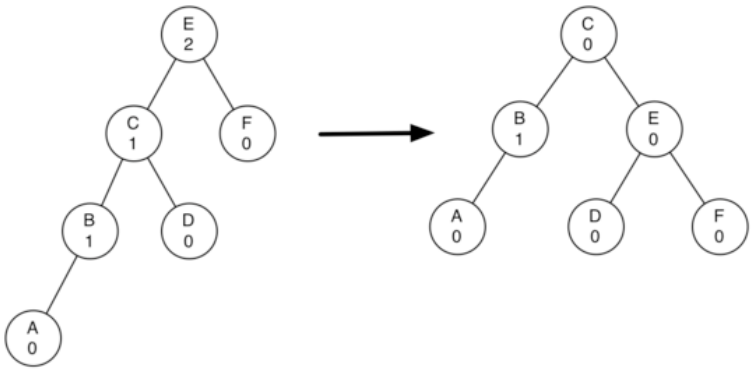
Post: Si la llave es encontrada, se elimina el nodo con todos los valores asociados a ella, y luego de esto, se balancea automáticamente.
Retorna true si el nodo fue eliminado.



AddNode(AVLNode x)

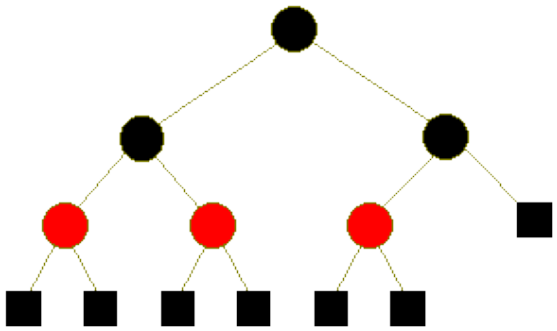
Pre: x diferente de null

Post: Se agrega el nodo al árbol y se balancea completamente.



TAD ÁRBOL ROJINEGRO

Representación:
Sea X, L, R nodos, donde L y R son sub árboles de X.



Invariante:

1. Cada nodo es rojo o negro.
2. La raíz es negra.
3. Todas las hojas (null) son negras.
4. Los hijos de un nodo rojo deben ser negros.
5. La altura negra debe ser la misma.

Operaciones:

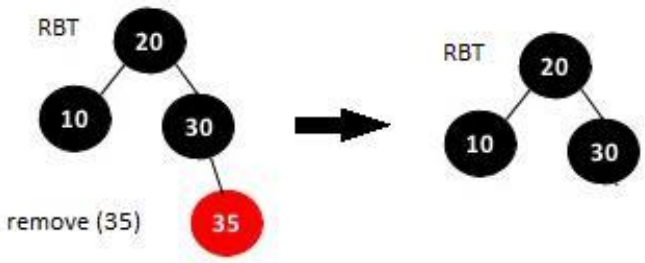
CreateRedBlackTree		→ RedBlackTree
AddNode	RBNode	→ void
DeleteNode	Key	→ Boolean

RedBlackTree()

Pre: true

Post: El árbol es creado

Root == null

DeleteNode(K key)
Pre: true
Post: Si la llave es encontrada, se elimina el nodo con todos los valores asociados a ella, y luego de esto, se balancea automáticamente. Retorna true si el nodo fue eliminado. 

AddNode(RBTNode x)
Pre: x diferente de null
Post: Se agrega el nodo al árbol y se balancea completamente.