TAD BST

BST = {Root = <root>, Root.RightChildren = <rightChildren>, Root.LeftChildren = <leftChildren>, Value = <value>, Repetitions = <repetitions> }

Invariant: root != nill, RightChildren != vacio, LeftChildren != ≡ ∧ Root.LeftChildren.value ≤ Root.value ≤ Raiz.RightChildren.value

Construction operations:
*Create : → BST
Modifier operations:
*InsertElement: BSTxvalue → BST
*Remove: BSTxvalue → BST
*ModifyRight: BSTxvalue → BST
*ModifyLeft: BSTxvalue → BST
*ModifyValue: BSTxvalue → BST
Analyzer operations:
*isEmpty: BST → boolean
*PreOrder: BST → sequence of values
*InOrder: BST → sequence of values
*PosOrder: BST → sequence of values
*Height: BST → Integer
*search: BST → BST
*weight: BST → entero

Create (value)

"Creates an element of the BST with the elements LeftChildren and RightChildren empty, but with a defined value"

{ pre: TRUE }

{ post: elementBST = {RightChildren = <nill>, LeftChildren = <nill> , Value=<value>}

InsertElement(value)

"Inserts an element in the BST having into account its value."

{ pre: TRUE}

{post: root = {RigthChildren = <nill>, LeftChildren = < element>, Value = <value> } }

Remove(value)

"Removes an element passed by parameter of the BST's children."

{ pre: element to be removed is in the BST}

{ post: False if the element wasn't removed,
True otherwise }

ModifyRight(BST, value)

"Modifies the right children of the BST passed by parameter."

{ pre: BST != vacio ∧value != nill }
{ post: BST = {RightChildren = <value>, ...}

ModifyRight(BST, value)

"Modifies the left children of the BST passed by parameter."

{ pre: BST != vacio ∧value != nill }
{ post: BST = {leftChildren = <value>, ...}

ModifyValue(element, v)

"Modifies the value of the element which is in the BST."

{ pre: element= {..., Value: <value>,...}
{ post: BST.Value= v }

isEmpty(BST):
"Informs if the BST is empty."
{ pre: TRUE }
{ pre: arbolBB={Root:<root>},...}

{post: False if the BST.raiz != ≡,
True otherwise}

Height(arbol):

"Returns an integer which represents the height of the BST."

{pre: TRUE }

{ post: n | n ∈ Z+ }

PreOrden():

"Traverses all elements of the binary search tree so that the root is traversed before the traversal of the left and right subtrees"
{pre: TRUE }

{ post: S = (a1, a2, a3,... an}, (an ∈ Z) ∧ (n ∈ Z+) }

PosOrden():

"It goes through all the elements of the binary search tree following the order left, right, root"
{pre: TRUE }

{ post: S = (a1, a2, a3,... an}, (an ∈ Z) ∧ (n ∈ Z+) }

inOrden():

"It goes through all the elements of the binary search tree following the order left, root, right"
{pre: TRUE }

{ post: S = (a1, a2, a3,... an}, (an ∈ Z) ∧ (n ∈ Z+) }