

Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje Java.

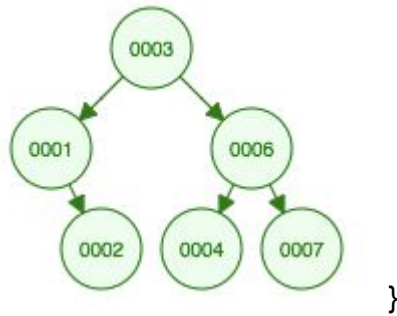
crearArbolABB para el método BinarySearchTree()
 agregar para el método add()
 contieneValor para el método contains()
 buscar para el método search()
 obtenerMayor para el método getGreatest()
 obtenerMenor para el método getLeast()
 eliminar para el método remove()
 limpiarArbol para el método clear()
 darTamaño para el método size()
 validarArbol para el método validate()
 ValidarNodo para el método validateNode()

crearArbolABB()

“crea un nuevo árbol de búsqueda binaria”

{pre: True}

{post: ArbolABB=

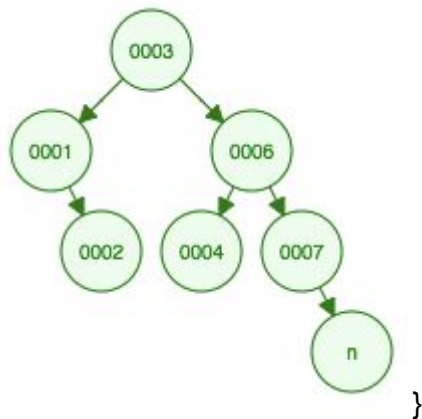


agregar(ArbolABB, n)

“agrega un nodo al árbol ABB”

{pre: True, n es de tipo nodo}

{post: ArbolABB=



contieneValor(n)

“verifica si el valor entrante por parámetro está en el árbol”

{pre: Árbol Creado}

{post: true si n se encuentra, false de lo contrario}

Buscar(n)

“buscar el valor que entra por parámetro en el árbol”

{pre: Árbol Creado, n es de tipo Nodo}

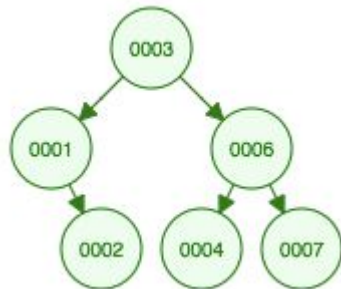
{post: n, si el elemento es encontrado, null de lo contrario}

eliminar(n)

“elimina el nodo que se pasa por parámetro”

{pre: n es de tipo nodo}

{post: ArbolABB=



}

limpiarArbol()

“elimina los valores de los nodos que existen”

{pre: Árbol creado}

{post: ArbolABB=



}

darTamaño()

“Da el tamaño de nodos del árbol”

{pre: Arbol creado}

{post: <tamaño> , el cual es de tipo int}

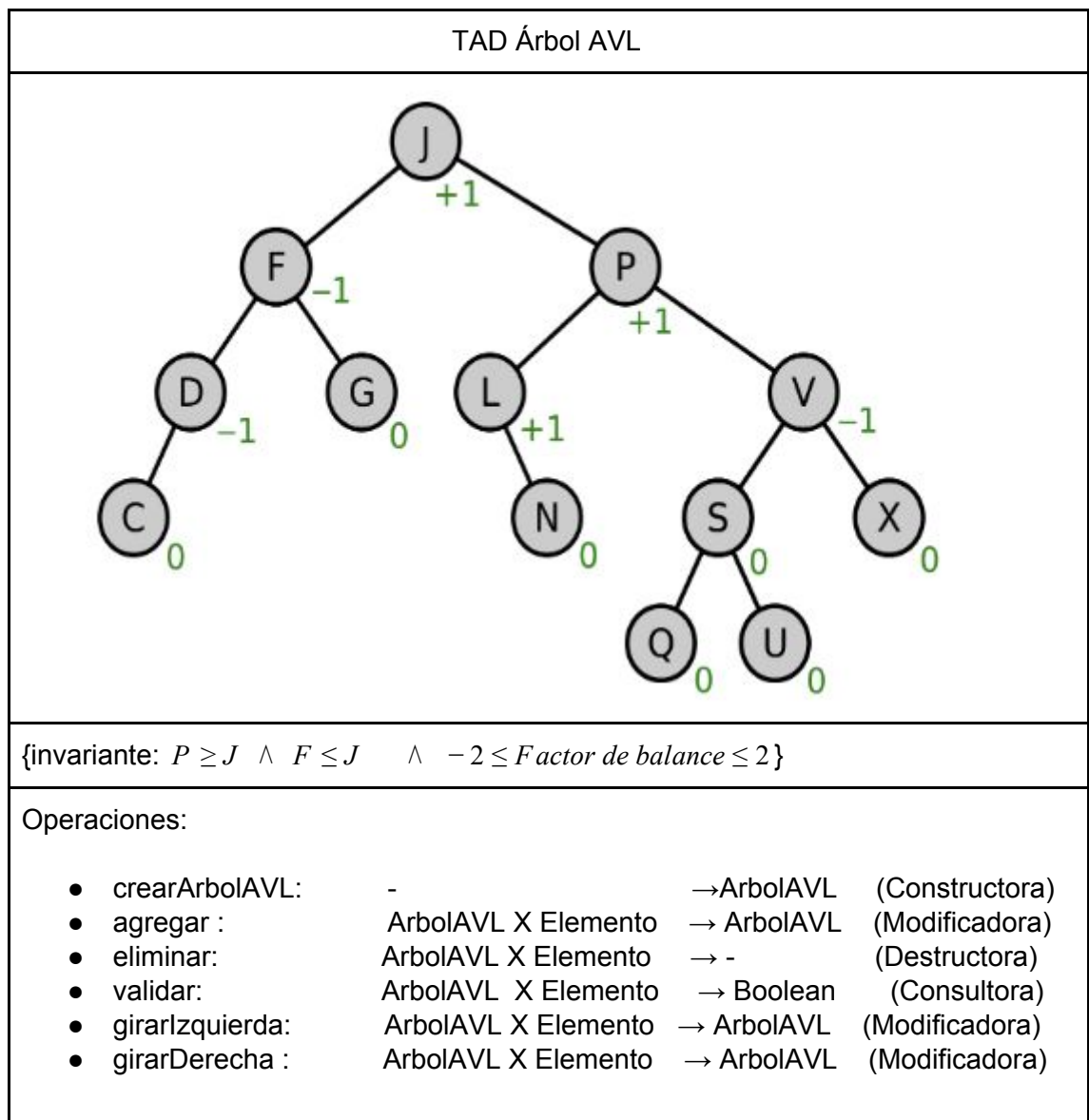
validarArbol()

“Valida si el árbol está vacía ”

{pre: Árbol creado}

{post: true si la raíz es null, false de lo contrario

}



Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje java.

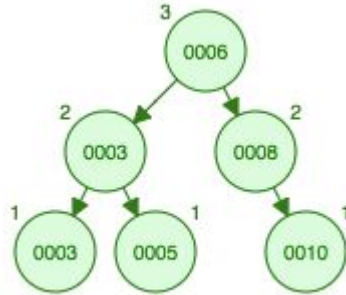
crearArbolAVL para el método AVLTree()
 agregar para el método addValue()
 eliminar para el método removeValue()
 validarNodo para el método validateNode()
 girarlzquierda para el método rotateLeft()
 girarDerecha para el método rotateRight()

crearArbolAVL()

“crea un nuevo árbol de tipo AVL”

{pre: True}

{post: ArbolAVL=



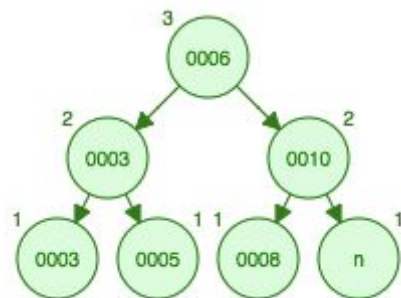
}

agregar(ArbolAVL, n)

“agrega un nodo al árbol AVL”

{pre: True, n es de tipo nodo}

{post: ArbolAVL=



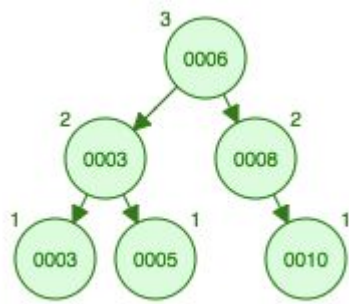
}

eliminar(n)

“elimina el nodo que se pasa por parámetro”

{pre: n es de tipo nodo}

{post: ArbolAVL=



}

validar()

“Valida si el árbol se encuentra vacío ”

{pre: Árbol creado}

{post: true si la raíz es null, false de lo contrario
}

girarIzquierda(n)

“el árbol se gira a la izquierda sobre el nodo pasado por parámetro ”

{pre: Árbol creado}

{post: el árbol es girado a la izquierda sobre el nodo}

girarDerecha(n)

“gira el árbol a la derecha sobre el nodo que pasa por parámetro ”

{pre: Árbol creado}

{post: el árbol es girado a la derecha sobre el nodo pasado por parámetro. }

| TAD Árbol Roji-negros | |
|--|--|
| <pre> graph TD F((F)) --- B((B)) F --- G((G)) B --- A((A)) B --- D((D)) D --- C((C)) D --- E((E)) G --- H((H)) style F fill:#000,color:#fff style B fill:#f00,color:#fff style G fill:#000,color:#fff style A fill:#000,color:#fff style D fill:#000,color:#fff style C fill:#f00,color:#fff style E fill:#f00,color:#fff style H fill:#f00,color:#fff </pre> | |
| <p>{invariante: $G \geq F \wedge B \leq F$ - Cada nodo es rojo o negro. - Toda hoja (NIL) es negra. - Si un nodo es rojo, sus dos hijos son negros. - Todo camino desde un nodo a cualquier hoja descendente contiene el mismo número de nodos negros.}</p> | |
| <p>Operaciones:</p> <ul style="list-style-type: none"> • crearArbolRJ: - → ArbolRoji-Negros (Constructora) • agregar : ArbolRoji-Negros X Elemento → ArbolRoji-Negros (Modificadora) • eliminar: ArbolRoji-Negros X Elemento → - (Destructor) • girarlzquierda: ArbolRoji-Negros X Elemento → ArbolRoji-Negros (Modificadora) • girarDerecha : ArbolRoji-Negros X Elemento → ArbolRoji-Negros (Modificadora) • validar: ArbolRoji-Negros X Elemento → Boolean (Consultora) | |

Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje java.

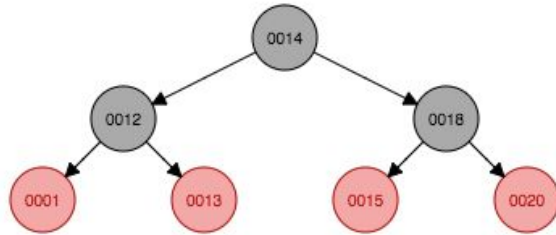
crearArbolRJ para el método RedBlackTree()
 agregar para el método addValue()
 eliminar para el método removeNode()
 validar para el método validate()
 girarlzquierda para el método rotateLeft()
 girarDerecha para el método rotateRight()

crearArbolRoji-Negro()

“crea un nuevo árbol de tipo Roji-Negros”

{pre: True}

{post: Árbol Roji-Negros=

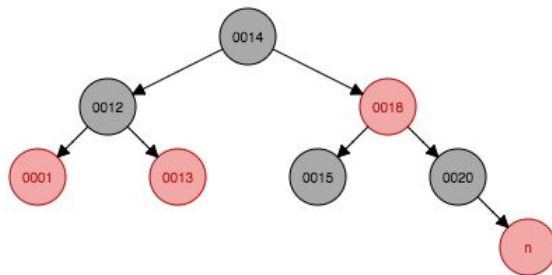


}

agregar(ArbolRoji-Negros, n)

“agrega un nodo al árbol Roji-Negros”

{pre: True, n es de tipo nodo}



{post: ArbolRoji-Negros=

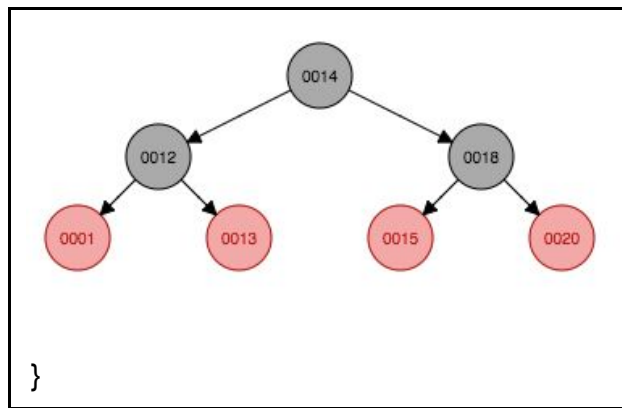
}

eliminar(n)

“elimina el nodo que se pasa por parámetro”

{pre: n es de tipo nodo}

{post: ArbolRoji-Negros=



validar()

“Valida si el árbol se encuentra vacío ”

{pre: Árbol creado}

{post: True si la raiz es igual a null, false de lo contrario }

girarIzquierda(n)

“el árbol es girado hacia la izquierda sobre el nodo n ”

{pre: Árbol creado}


{post: el árbol es girado a la izquierda }

girarDerecha()

“el árbol es girado hacia la derecha sobre el nodo n ”

{pre: Árbol creado}

{post: el árbol es girado hacia la derecha }

| TAD Nodo de Árbol ABB | |
|---|--|
|  | |
| {invariante: $n - i \leq n \leq n + i$ } | |
| Operaciones: <ul style="list-style-type: none"> • crearNodo: - →Nodo (Constructora) • darID : Nodo →Nodo (Consultora) | |


Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje java.

crearNodo para el método Node()

darId para el método getID()


| |
|--|
| crearNodo() “crea un nodo de tipo Árbol ABB ” {pre: nodo creado} {post: NodoABB= <div data-bbox="563 1200 647 1290" data-label="Diagram"> </div> } |
|--|

| |
|---|
| darID() “retorna el tipo de dato que contiene el nodo ” {pre: nodo creado} {post: valores entre 1 y 12 según sea el criterio.} |
|---|

| TAD Nodo de Árbol AVL | | | |
|--|--|--|--|
|  | | | |
| {invariante: $n - i \leq n \leq n + i \wedge -2 \leq \text{Factor de balance} \leq 2$ } | | | |
| <p>Operaciones:</p> <ul style="list-style-type: none"> • crearNodo: - →Nodo (Constructora) • esHoja : Nodo → Boolean (Consultora) • actualizarFC : Nodo → Nodo (Consultora) • darFC Nodo → Nodo (Consultora) • darID : Nodo →Nodo (Consultora) | | | |

Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje java.

crearNodo para el método Node()
darId para el método getID()
esHoja para el método isLeaf()
actualizarFC para el método updateHeight()
darFC para el método getBalanceFactor()

| |
|--|
| <p>crearNodo()</p> <p>“crea un nodo de tipo Árbol AVL ”</p> <p>{pre: nodo creado}</p> <p>{post: NodoAVL=</p> <div style="text-align: center;">  </div> <p>}</p> |
|--|

darID()

“retorna el tipo de dato que contiene el nodo”

{pre: nodo creado}

{post: valores entre 1 y 12 según sea el criterio.}

esHoja()

“verifica si el nodo es hoja”

{pre: nodo creado}

{post: true si es hoja, false de lo contrario}

actualizarFC()

“se actualiza el factor de balanceo”

{pre: nodo creado}


{post: el valor del factor de balanceo es actualizado}

darFC()

“entrega el factor de balanceo”


{pre: nodo creado}

{post: se entrega el factor, siendo este de tipo int }

| TAD Nodo de Árbol Roji-Negros | | | |
|---|--|--|--|
|  | | | |
| {invariante: $n - i \leq n \leq n + i \wedge n \text{ puede ser Negro o Rojo}$ } | | | |
| <p>Operaciones:</p> <ul style="list-style-type: none"> • crearNodo: - →Nodo (Constructora) • darID : Nodo →Nodo (Consultora) • esHoja : Nodo → Boolean (Consultora) • darTio : Nodo → Nodo (Consultora) • darHermano Nodo → Nodo (Consultora) • darAbuelo Nodo --> | | | |

Para este TAD se usaron las siguientes referencias para los métodos reales en nuestro código de lenguaje java.

crearNodo para el método Node()
 darId para el método getID()
 esHoja para el método isLeaf()
 darTio para el método getUncle()
 darHermano para el método getSibling()
 darAbuelo para el método getGrandParent()

| |
|---|
| crearNodo() “crea un nodo de tipo Árbol Roji-Negro ” {pre: nodo creado} {post: NodoRoji-Negro= <div>  </div> } |
|---|

| |
|--|
| darID() “retorna el tipo de dato que contiene el nodo |
|--|

”

{pre: nodo creado}

{post: valores entre 1 y 12 según sea el criterio.}

esHoja()

“verifica si el nodo es hoja”

{pre: nodo creado}

{post: true si es hoja, false de lo contrario}

darTio()

“entrega el tío del nodo actual”

{pre: nodo creado}

{post: Se entrega el tío del nodo actual, si no tiene se entrega null}

darHermano()

“entrega el hermano del nodo actual”

{pre: nodo creado}

{post: se entrega al hermano del nodo actual, sino tiene, es null}