**TADs**

|  |  |  |  |
| --- | --- | --- | --- |
| **AVL TAD** | | | |
| AVL = { root } | | | |
| { inv: forall ParentNode>LeftChildNode ^ ParentNode < RightChildNode} | | | |
| Primitive operations: | | | |
| * AVL |  | * Avl | Constructor |
| * addNode | root X Element | * Avl | Modifier |
| * printInOrder | root | * Avl | Analyzer |
| * printPreOrder | root | * Avl | Analyzer |
| * printPostOrder | root | * Avl | Analyzer |
| * deleteNode | root X Element | * Node | Modifier |
| * search | root X Element | * Node | Analyzer |
| * getHeight | root | * Integer | Analyzer |
| * getMinNode | root | * Node | Analyzer |
| * getMaxNode | root | * Node | Analyzer |
| * evaluateBalance | root | * Avl | Analyzer |
| * simpleRightRotation | node | * Avl | Modifier |
| * simpleLeftRotation | node | * Avl | Modifier |
| * doubleRightRotation | node | * Avl | Modifier |
| * doubleLeftRotation | node | * Avl | Modifier |

|  |
| --- |
| **Avl ()** |
| Binary Search Tree constructor  { pre: TRUE}  { post: Bst = { root = NIL , size = 0 } |

|  |
| --- |
| **addNode(Element)** |
| Create a new node with the element and place it in the tree.  { pre: root }  { post: root!=null, size+1} |

|  |
| --- |
| **printInOrder (root)** |
| Print the tree in the following way: left subtree, root, right subtree."  { pre: root }  { post: left.node,root,right.node } |

|  |
| --- |
| **printPreOrder(root)** |
| Print the tree in the following order: root, left subtree, right subtree.  { pre: root }  { post: root,left.node,right.node } |

|  |
| --- |
| **printPostOrder(root)** |
| Print the tree in the following order: left subtree right subtree, root.  { pre: root }  { post: left.node , right.node, root } |

|  |
| --- |
| **deleteNode (root*, element*)** |
| To delete a specific node from the tree.  { pre: root, <element> }  { post: node.element=null} |

|  |
| --- |
| **search (root*, element*)** |
| Search for an element in the tree's nodes  { pre: root, <element> }  { post: if true node.element | else null} |

|  |
| --- |
| **getHeight (root)** |
| Calculate the height of the tree.  { pre: root}  { post: Integer } |

|  |
| --- |
| **getMinNode(root)** |
| Find the minimum node in the tree.  { pre: root }  { post: node.left (while !=null) } |

|  |
| --- |
| **getMaxNode(root)** |
| Find the maximum node in the tree.  { pre: root }  { post: node.right (while !=null) } |

|  |
| --- |
| **evaluateBalance (root)** |
| Calcula el balance del arbol  { pre: root } |