|  |
| --- |
| **TAD BST (ABB)** |
| Image result for ABB tree  Right-node  Left-node  root |
| { inv : Node.leftNode <= Node && Node.rightNode > Node} |
| Primitive operations:   * insert: Node -> ABB * delete: Node x Id -> ABB * find: Node x Id -> Node * getSucessor: Node -> Node * getPredecessor: Node -> Node * isEmpty: -> Boolean * getSize: -> Integer |

|  |
| --- |
| **Insert(node)**  “Adds a node to the tree”  {pre: TRUE}  {post: stack = { BST.getSize = ABB.getSize + 1 } } |

|  |
| --- |
| **Delete(node, id)**  “Deletes a node of the tree”  {pre: node ∈ BST}  {post: stack = { BST.getSize = BST.getSize - 1 } } |

|  |
| --- |
| **Find(node)**  “Searches a node on the tree and return it if exists”  {pre: TRUE}  {post: stack = {node } } |

|  |
| --- |
| **GetSucessor(node)**  “Returns node successor”  {pre: node ∈ BST}  {post: stack = { node,successor } } |

|  |
| --- |
| **GetPredecessor (node)**  “Returns node predecessor”  {pre: node ∈ BST}  {post: stack = { node,predecessor } } |

|  |
| --- |
| **IsEmpty()**  “Tells if root is NIL”  {pre: TRUE}  {post: False if BTS.root != NIL, True if contrary}} |

|  |
| --- |
| **GetSize (node)**  “Returns the total number of nodes in BTS”  {pre: TRUE}  {post: stack = {BTS, size} } |

|  |
| --- |
| **TAD RBT (ROJINEGRO)** |
| Resultado de imagen para red black tree  Left-node  Right-node  root |
| { inv :  -Node.leftNode <= Node && Node.rightNode > Node -Node.color = black OR Node.color = red -From root to any leaf, there is equal number of black nodes -If Node.color = red, then Node.leftNode.color = black AND Node.rightNode.color = black -Root.color = black} |
| Primitive operations:   * insert: Node -> RBT * insertFixup: Node -> RBT * delete: Node x Id -> RBT * deleteFixup Node -> RBT * find: Node x Id -> Node * getSucessor: Node -> Node * getPredecessor: Node -> Node * isEmpty: -> Boolean * getSize: -> Integer * rightRotate: Node -> RBT * rightRotateFixUp: Node -> RBT * leftRotate: Node -> RBT * leftRotateFixUp: Node -> RBT |

|  |
| --- |
| **Insert(node)**  “Adds a node to the tree”  {pre: TRUE}  {post: stack = { BST.getSize = ABB.getSize + 1 } } |

|  |
| --- |
| **Delete(node, id)**  “Deletes a node of the tree”  {pre: node ∈ BST}  {post: stack = { BST.getSize = BST.getSize - 1 } } |

|  |
| --- |
| **Find(node)**  “Searches a node on the tree and return it if exists”  {pre: TRUE}  {post: stack = {node } } |

|  |
| --- |
| **GetSucessor(node)**  “Returns node successor”  {pre: node ∈ BST}  {post: stack = { node,successor } } |

|  |
| --- |
| **GetPredecessor (node)**  “Returns node predecessor”  {pre: node ∈ BST}  {post: stack = { node,predecessor } } |

|  |
| --- |
| **IsEmpty()**  “Tells if root is NIL”  {pre: TRUE}  {post: False if BTS.root != NIL, True if contrary}} |

|  |
| --- |
| **GetSize (node)**  “Returns the total number of nodes in BTS”  {pre: TRUE}  {post: stack = {BTS, size} } |

|  |
| --- |
| **InsertFixUp (node)**  “Recolor and rotates the nodes necessaries to keep invariants  {pre: TRUE}  {post: stack = {RBT with all properties} } |

|  |
| --- |
| **rotateRight (node)**  “Rotates the node to right”  {pre: TRUE}  {post: stack = {RBT with roted node} } |

|  |
| --- |
| **rotateRightFixUp (node)**  “Corrects the changes that violate invariants at rotate to keep invariants”  {pre: TRUE}  {post: stack = {RBT with invariants checked} } |

|  |
| --- |
| **rotateLeft (node)**  “Rotates the node to left”  {pre: TRUE}  {post: stack = {RBT with rotated node} } |

|  |
| --- |
| **rotateLeftFixUp (node)**  “Corrects the changes that violate invariants at rotate to keep invariants”  {pre: TRUE}  {post: stack = {RBT with invariants checked} } |

|  |
| --- |
| **deleteFixUp (node)**  Corrects the changes that violate invariants at delete a node to keep invariants“  {pre: TRUE}  {post: stack = {RBT with invariants checked } } |

|  |
| --- |
| **TAD AVL** |
| Resultado de imagen para avl tree  root  Left-node  Right-node |
| { inv :  -Node.leftNode <= Node && Node.rightNode > Node -Node.balanceFactor >=-1 && Node.balanceFactor <=1 } |
| Primitive operations:   * insert: Node -> AVL * delete: Node x Id -> AVL * find: Node x Id -> Node * getSucessor: Node -> Node * getPredecessor: Node -> Node * isEmpty: -> Boolean * getSize: -> Integer * rightRotate: Node -> AVL * leftRotate: Node -> AVL * rebalance: Node ->AVL |

|  |
| --- |
| **Insert(node)**  “Adds a node to the tree”  {pre: TRUE}  {post: stack = { AVL.getSize = AVL.getSize + 1 } } |

|  |
| --- |
| **GetSucessor(node)**  “Returns node successor”  {pre: node ∈ AVL}  {post: stack = { node,successor } } |

|  |
| --- |
| **IsEmpty()**  “Tells if root is NIL”  {pre: TRUE}  {post: False if AVL.root != NIL, True if contrary}} |

|  |
| --- |
| **Find(node)**  “Searches a node on the tree and return it if exists”  {pre: TRUE}  {post: stack = {node } } |

|  |
| --- |
| **GetSize (node)**  “Returns the total number of nodes in AVL”  {pre: TRUE}  {post: stack = {AVL, size} } |

|  |
| --- |
| **Delete(node, id)**  “Deletes a node of the tree”  {pre: node ∈ AVL}  {post: stack = { AVL.getSize = AVL.getSize - 1 } } |

|  |
| --- |
| **rebalance(node)**  “Rotates nodes necessaries to keep invariants”  {pre: TRUE}  {post: stack = {node } } |

|  |
| --- |
| **GetPredecessor (node)**  “Returns node predecessor”  {pre: node ∈ AVL}  {post: stack = { node,predecessor } } |