|  |
| --- |
| **TAD BST** |
| Árbol binario de búsqueda - Wikipedia, la enciclopedia libre |
| {inv: All the values in left tree’s son are less or equal to the value that the root contain. All values in its right are greater than the value which contains the root. The left and right are BST too. |
| **BST: <BST> Constructor**  **insertE: < ind,value > Modifier**  **insertE: <node> Modifier**  **insertE: <node1,node2> Modifier**  **searchEquals: <key> <ArrayList<V>v>Analyzer**  **searchEquals: <node, key> <node> Analyzer** |

TAD’S

|  |
| --- |
| BST() |
| Create an empty BST |
| {pre: } |
| {post: BST b = ∅ } |

|  |
| --- |
| InsertE(ind,value) |

|  |
| --- |
| Create a node then insert to the tree |
| {pre: ind and value different from null} |
| {post: Node was created} |

|  |
| --- |
| InsertE(Node<K,V>) |
| Add a new node to the tree |
| {pre: node !=null} |
| {post: The new node was added} |

|  |
| --- |
| InsertE(Node<K,V> current, Node<K,V> newNode) |
| Insert a new node to the tree |
| {pre: current and newNode non-null} |
| {post: The new node is added to the tree} |

|  |
| --- |
| searchEquals(Node<K,V> current, K key) |
| Search for a specific node according to the key |
| {pre: current!=null} |
| {post: node if node.getKey=key or null if the opposite happens} |

|  |
| --- |
| searchEquals (K key) |
| Search for a list using key as an index |
| {pre: } |
| {post: ArrayList<V> & if v≠∅ or null } |

|  |
| --- |
| **TAD AVL** |
|  |
| INV: BF (node) = Height(rightTree(node)) - Height(leftTree(node)) | BF (node) € {- 1,0,1}} |
| <AVL> **Constructor**  insert: < key,value >**Modifier**  insert: <node>**Modifier**  insert: <node1,node2>**Modifier**  searchElement: <key> <ArrayList<V>v>**Analyzer**  searchElement: <node, key> **Analyzer**  searchByRange: <key> <ArrayList<V>v>**Analyzer**  search: <node, key> **Analyzer**  inOrder: <node>**Analyzer**  preOrder: <node>**Analyzer**  rightRotate:<node> > < NodeAVL<K,V>> **Modifier**  getBalance:<node> < NodeAVL<K,V>> **Modifier** |

|  |
| --- |
| AVL() |
| Create a new AVL empty |
| {pre: right child minus left child heights} |
| {post: AVL that it is either empty, or both children  they are also AVL and the difference between their heights is less than or equal to 1} |

|  |
| --- |
| InsertE(key,value) |

|  |
| --- |
| Create a node and add it |
| {pre: key and value different from null} |
| {post: Node created} |

|  |
| --- |
| Insert(Node<K,V>) |
| Insert a new node to the tree |
| {pre: node different from null} |
| {post: The new node is added to the tree} |

|  |
| --- |
| Insert(Node<K,V> current, Node<K,V> newNode) |
| Insert a new node to the tree |
| {pre: current and newNode non-null} |
| {post: The new node is added to the tree} |

|  |
| --- |
| searchElement(Node<K,V> current, K key) |
| Search for a specific node according to the key |
| {pre: current!=null} |
| {post: node if node.getKey=key or null if the opposite happens} |

|  |
| --- |
| search (K key) |
| Returns a list with the indices of the found key |
| {pre:tree!=null } |
| {post: ArrayList<V> v if v≠∅ or null if the opposite happens } |

|  |
| --- |
| inOrder (Node<K,V> node, K key) |
| Find all nodes with less than the value of the key |
| {pre: node!=null} |

|  |
| --- |
| {post: } |

|  |
| --- |
| preOrder (Node<K,V> node, K key) |
| Find all nodes with more than the value of the key |
| {pre: node!=null } |
| {post: } |

|  |
| --- |
| leftRotate(NodeAVL<K,V> node) |
| Rotate the tree to left for balance the tree |
| {pre: AVL!=null, (is not\_Empty(AVL)) and (is not\_Empty(Hijo\_Left) } |
| {post: The three rotate to the left} |

|  |
| --- |
| rightRotate(NodeAVL<K,V> node) |
| Rotate the tree to right for balance the tree |
| {pre: AVL!=null, (is not\_Empty(AVL)) and (is not\_Empty(Hijo\_Right)} |
| {post: The three rotate to the right} |

|  |
| --- |
| getBalance(NodeAVL<K,V> node) |
| Balance the tree when the child has a height h + 2 |
| {pre: the right or left child has an h + 2} |
| {post:the h of tree are = h } |

Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente





