**Tree functions in Java**

public static boolean **isLeaf**( BinNode<Integer> node )

// returns true if the node is a leaf

{

return node.getLeft() == null && node.getRight() == null;

}

public static int **countNodes**( BinNode<Integer> tr)

// returns the number of nodes in tree tr

{

if (tr == null) return 0;

return countNodes(tr.getLeft()) + countNodes(tr.getRight()) + 1;

}

public static int **countLeaves**( BinNode<Integer> tr)

// returns the number of leaves in tree tr

{

if (tr == null) return 0;

if (isLeaf(tr) ) return 1;

return countLeaves(tr.getLeft()) + countLeaves(tr.getRight());

}

public static int **sumNodes**( BinNode<Integer> tr)

// returns the sum of the values of all the nodes in tree tr

{

if (tr == null) return 0;

return sumNodes(tr.getLeft()) + sumNodes(tr.getRight()) + tr.getInfo();

}

public static int **sumLeaves**( BinNode<Integer> tr)

// returns the sum of the values of all the leaves in tree tr

{

if (tr == null) return 0;

if (isLeaf(tr) ) return tr.getInfo();

return sumNodes(tr.getLeft()) + sumNodes(tr.getRight()) + tr.getInfo();

}

public static boolean **hasOneSon**( BinNode<Integer> node )

// returns true is the node has only one son

{

return (node.getLeft() != null && node.getRight() == null)

|| (node.getLeft() == null && node.getRight() != null);

}

public static boolean **hasTwoSons**( BinNode<Integer> node )

// returns true if the node has two sons

{

return (node.getLeft() != null && node.getRight() != null);

}

public static int **countOneSon**( BinNode<Integer> tr)

// returns the number of nodes in the tree with only one son

{

if (tr == null) return 0;

if (hasOneSon(tr) ) return 1;

return countOneSon(tr.getLeft()) + countOneSon(tr.getRight());

}

public static int **countRightSons**( BinNode<Integer> tr)

// returns the number of right sons in the tree

{

if (tr == null) return 0;

if (tr.getRight()!= null)

return countRightSons(tr.getLeft()) + countRightSons(tr.getRight()) + 1;

return countRightSons(tr.getLeft()) + countRightSons(tr.getRight());

}

public static int **countLeftSons**( BinNode<Integer> tr)

// returns the number of leftsons in the tree

{

if (tr == null) return 0;

if (tr.getLeft()!= null)

return countLeftSons(tr.getLeft()) + countLeftSons(tr.getRight()) + 1;

return countLeftSons(tr.getLeft()) + countLeftSons(tr.getRight());

}

public static boolean **exsists**( BinNode<Integer> tr, int x )

// return true if the value x is in the tree tr

{

if (tr == null) return false;

if (tr.getInfo() == x) return true;

return exsists(tr.getLeft(),x) || exsists( tr.getRight(), x);

}

public static boolean **exsists**( BinNode<Integer> tr, BinNode<Integer> node )

// return true if node is in the tree tr

{

if (tr == null) return false;

if (tr == node) return true;

return exsists(tr.getLeft(), node) || exsists( tr.getRight(), node);

}

public static int **height**( BinNode<Integer> tr)

// returns the height ot the tree

{

if (tr == null) return -1;

return Math.max(height(tr.getLeft()) , height(tr.getRight()) ) + 1;

}

public static int **countNodesAtLevel**( BinNode<Integer> tr, int x)

// returns number of nodes at level x

{

if (tr == null) return 0;

if ( x == 0 ) return 1;

return countNodesAtLevel(tr.getLeft(), x-1) + countNodesAtLevel(tr.getRight(), x-1);

}

public static int **sumNodesAtLevel**( BinNode<Integer> tr, int x)

// returns the sum of all nodes at level x

{

if (tr == null) return 0;

if ( x == 0 ) return tr.getInfo();

return sumNodesAtLevel(tr.getLeft(), x-1) + sumNodesAtLevel(tr.getRight(), x-1);

}

public static boolean **isDescendant** (BinNode<Integer> tr, int x, int y)

// Retruns true if node with value y is a descendant of node with value x

{

if (!exsists(tr,x)) return false;

return (exsists(tr.getLeft(),y)|| (exsists(tr.getRight(),y)) );

}

public static boolean **isDescendant** (BinNode<Integer> tr, BinNode<Integer> node1, BinNode<Integer> node2)

// Retruns true if node2 is a descendant of node1

{

if (!exsists(tr,node1)) return false;

return (exsists(node1,node2) && node1 != node2 );

}

public static int **treeMax**( BinNode<Integer> tr)

// retruns the largest value in the tree tr

{

int big = tr.getInfo();

if ( tr.getLeft() != null ) big = Math.max(big, treeMax(tr.getLeft()));

if ( tr.getRight() != null ) big = Math.max(big, treeMax(tr.getRight()));

return big;

}

public static boolean **isComplete**( BinNode<Integer> tr)

// retruns true is the the tr is complete

{

if ( tr.getLeft() == null && tr.getLeft() == null) return true;

if ( tr.getLeft() == null || tr.getLeft() == null) return false;

if ( height(tr.getLeft()) != height(tr.getRight()) ) return false;

return isComplete(tr.getLeft()) && isComplete(tr.getRight());

}

public static BinNode<Integer> **parent** ( BinNode<Integer> tr, BinNode<Integer> son)

// returns the parent of son

{

if ( tr == null || tr.getLeft() == son || tr.getRight() == son) return tr;

BinNode<Integer> tmp = parent(tr.getLeft(), son);

if (tmp != null) return tmp;

return parent(tr.getRight(), son);

}

public static boolean **areBrothers** (BinNode<Integer> tr, BinNode<Integer> son1, BinNode<Integer> son2)

// returns true is son1 and son2 are brothers

{

return son1 != son2 && parent(tr,son1) == parent(tr,son2);

}

public static BinNode<Integer> **ancestor** (BinNode<Integer> tr, BinNode<Integer> node1, BinNode<Integer> node2)

// returns the closest common ancestor of node1 and node2 in tree tr

{

BinNode<Integer> parentNode1 = parent (tr,node1);

if (parentNode1 == null ) return null;

if (isDescendant(tr,parentNode1,node2)) return parentNode1;

return ancestor (tr, parentNode1, node2);

}

public static int **nodeLevel** (BinNode<Integer> tr, BinNode<Integer> node)

// returns the level of node is tree tr

{

if ( tr == node) return 0;

return nodeLevel(tr,parent(tr,node) + 1;

}

public static void **preOrder** ( BinNode<Integer> tr)

// preorder transversal of the tree

// the method visit is a virtual function that should be over ridden

{

if (tr != null)

{ visit(tr) // the process to do on each node for example

//System.out.println(tr.getInfo())

preOrder(tr.getLeft());

preOrder(tr.getRight());

}

}

public static void **inOrder** ( BinNode<Integer> tr)

// inorder transversal of the tree

// the method visit is a virtual function that should be over ridden

{

if (tr != null)

{ inOrder(tr.getLeft());

visit(tr) // the process to do on each node for example

//System.out.println(tr.getInfo())

inOrder(tr.getRight());

}

}

public static void **postOrder** ( BinNode<Integer> tr)

// postorder transversal of the tree

// the method visit is a virtual function that should be over ridden

{

if (tr != null)

{ postOrder(tr.getLeft());

postOrder(tr.getRight());

visit(tr) // the process to do on each node for example

// System.out.println(tr.getInfo())

}

}