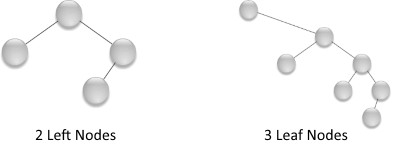
**Important**: This tutorial has an online part, which you should complete on LMS (tutorial section). The deadline for online task is Monday 21 November at 8:00 A.M

# Problem 1

Write the method **LeafNodes** part of the Binary Tree ADT. It should return the number of leaf nodes in the tree.

***Method: public int LeafNodes()***

**Examples**:



public int LeafNodes(){

if(root==null)

return 0;

BTNode<T> p=root;

//Stack is sed to store the right child of nodes

Stack<BTNode<t>> stack= new Stack<BTNode<T>>();

int count=0;

while(p!=null){

if(p.right==null&&p.left==null)

count++;

else if(p.right!=null)

stack.push(p.right);

if(p.left!=null)

p=p.left;

else{

if(!stack.empty())

p=stack.pop();

else {

p=null;

}

}

return count;

}

public int LeafNodes(){

return LeafNodes\_rec(root);

}

private int LeafNodes\_rec(BTNode<T> p){

int count=0;

if(p==null)

return 0;

if(p.right==null&&p.left==null)

counter++;

return counter+ LeafNodes\_rec(p.left)+ LeafNodes\_rec(p.right);

}

**Problem 2**

Write the method **TreeHeight** part of the Binary Tree ADT. It should return the height of the tree. The height of the tree is the longest path from the root to a leaf node.

***Method: public int TreeHeight()***

**Example**:

public TreeHeight(){

return TreeHeight\_rec(root);

}

private int TreeHeight\_rec(BTNode<T> p){

if(p==null)

return 0;

int LeftHeight= TreeHeight\_rec(p.left);

int RightHeight= TreeHeight\_rec(p.right);

return LeftHeight>RightHeight? LeftHeight+1:RightHeight+1;

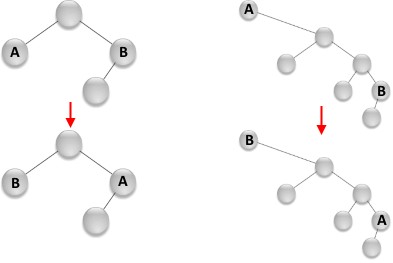
}

**Problem 3**

Write the static method **SwapMost** that takes a Binary tree **bt** and swaps the data of the left most node with the right most node.

***Method: public static <T> void SwapMost(BinaryTree<T> bt)***

**Example:**



Public static <T> void SwapMost(BT<t>bt){

//start from root

bt.find(Relatives.Root);

//find most left child

while(bt.find(Relatives.LeftChild));

//save the data

T MostLeft= bt.retreieve();

//go back to root

bt.find(Relatives.Root);

while(bt.find(Relatives.rightChild));

T Mostright= bt.retreieve();

bt.find(Relatives.Root);

while(bt.find(Relatives.LeftChild));

bt.update(Mostright);

bt.find(Relatives.Root);

while(bt.find(Relatives.rightChild));

bt.update(MostLeft);

}

}

