**ExtraExDS\_4Trees – Exercise 1**

**Specification**

class binary\_tree[node]

genres b\_tree, node, label, int

operations

parent: node b\_tree 🡪 node

left\_child: node b\_tree 🡪 node

right\_child: node b\_tree 🡪 node

label: node b\_tree 🡪 label

create: b\_tree b\_tree label -> b\_tree

root: b\_tree 🡪 node

makenull: b\_tree 🡪 b\_tree

num\_nodes: b\_tree 🡪 int

num\_leafs: b\_tree 🡪 int

num\_2children: b\_tree 🡪 int

count\_repeat: b\_tree b\_tree 🡪 int

endspec

**Implementation**

node = record

element : label

leftchild: ^node

rightchild: ^node

parent: ^node

endrecord

label: elementtype

class B\_Tree

private root: ^node

public ^node parent(n: node)

public ^node left\_child(n: node)

public ^node right\_child(n: node)

public label label(n: node)

public void create(leftTree, rightTree: ^node, l: label)

public ^node root()

public void makenull()

public int num\_nodes(n: ^node)

public int num\_leafs(n: ^node)

public int num\_2children(n: ^node)

public int count\_repeat(n1, n2: ^node)

endclass

public int B\_Tree:: num\_nodes(n: ^node)

counter: int

counter:= 0

if n == null

return counter

else

counter += 1

counter += num\_nodes(n^.leftchild)

counter += num\_nodes(n^.rightchild)

return counter

endif

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the tree to count the number of nodes from the tree.

public int B\_Tree:: num\_leafs(n: ^node)

counter: int

counter:= 0

if(n == null) { Only if the root is null}

return counter

else

if(n^.leftchild == null && n^.rightchild == null) { If there are no children, is a leaf}

counter += 1

endif

if(n^.leftchild == null)

counter += num\_leafs(n^.rightchild)

endif

if(n^.rightchild == null)

counter += num\_leafs(n^.leftchild)

endif

return counter

endif

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the tree to count the number of leaves from the tree.

public int B\_Tree:: num\_2childrens(n: ^node)

counter: int

counter:= 0

if(n == null)

return counter

endif

if(n^.leftchild == null && n^.rightchild == null) { If there are no children, is a leaf}

return counter

else if(n^.leftchild == null)

counter += num\_2childrens(n^.rightchild)

return counter

else if(n^.rightchild == null)

counter += num\_2childrens(n^.leftchild)

return counter

else

counter += 1

counter += num\_2childrens(n^.leftchild)

counter += num\_2childrens(n^.rightchild)

return counter

endif

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the tree to count the number of nodes with 2 children from the tree.

public int B\_Tree:: count\_repeat(n1, n2: ^node)

counter: int

counter:= 0

if(n1 == null || n2== null)

return counter

else

if(n1^.element == n2^.element)

counter+= 1

endif

if(n1^.rightchild != null && n2^.rightchild != null)

counter += count\_repeat(r1^.rightchild, r2^.rightchild)

endif

if(n1^.leftchild != null && n2^.leftchild != null)

counter += count\_repeat(r1^.leftchild, r2^.leftchild)

endif

return counter

endif

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the first tree in the worst case even though they are not in the second tree.