Computación y Estructuras Discretas I
Seguimiento 11 - Árboles binarios

Estudiante:
Cristian Felipe Perafan Chilito - A00378035

TAD Node
Node<T> = newNode<T>(T value)
{inv : ⟨T != int,char,double…)⟩(Los valores de los nodos no deben almacenar datos primitivos)}
Operaciones: • Node() T value → Node<T> • getValue() N/A → T value • setValue() T value → N/A • getFather() N/A → Node<T> • setFather() Node<T> → N/A • getLeftSon() N/A → Node<T> • setLeftSon() Node<T> → N/A • getRightSon() N/A → Node<T> • setRightSon() Node<T> → N/A

Descripción formal de las operaciones:

Node()
“Crea un nuevo nodo T con un valor de tipo genérico T”
{pre: ⟨El tipo de dato T no debe ser primitivo⟩ }
{pos: ⟨El Node<T> es creado en el sistema ⟩ }

getValue()
“ Consulta el valor <T> que tiene un Node<T> y retorna dicho valor <T>”
{pre: ⟨El valor T del Node<T> debe ser diferente de null⟩ }
{pos: ⟨El valor <T> del Node<T> es retornado en el sistema⟩ }

setValue()
“Modifica el valor <T> que tiene un Node<T>”
{pre: ⟨El valor T del Node<T> debe ser diferente de null⟩ }
{pos: ⟨El valor T del Node<T> ha sido modificado⟩ }

getFather()
“ Consulta el padre que tiene un Node<T> y retorna un NodePadre <T>”
{pre: ⟨El valor Node<T> para el cual se quiere hacer la consulta no deber ser la raíz del árbol⟩ }
{pre: ⟨El valor de NodePadre<T> es retornado en el sistema⟩ }

setFather()
“ Modifica el padre que tiene un Node<T> ”
{pre: ⟨El valor Node<T> para el cual se quiere hacer la modificación no deber ser la raíz del árbol⟩ }
{pos: ⟨El NodePadre<T> de Node<T> es modificado en el sistema⟩ }

getLeftSon()
“ Consulta el hijo izquierdo que tiene un Node<T> y retorna un NodeHijoIzquierdo <T>”
{pre: ⟨El hijo izquierdo de Node<T> para el cual se quiere hacer la consulta debe ser diferente de null⟩ }
{pos: ⟨El hijo izquierdo de Node<T> es retornado en el sistema⟩ }

setLeftSon()
“ Modifica el hijo izquierdo que tiene un Node<T> ”
{pre: ⟨El nuevo hijo izquierdo del Node<T> no debe ser de tipo primitivo⟩ }
{pos: ⟨El hijo izquierdo del Node<T> es modificado en el sistema⟩ }

getRightSon()
“ Consulta el hijo derecho que tiene un Node<T> y retorna un NodeHijoDerecho <T>”
{pre: ⟨El hijo derecho de Node<T> para el cual se quiere hacer la consulta debe ser diferente de null⟩ }
{pos: ⟨El hijo derecho de Node<T> es retornado en el sistema⟩ }

setLeftSon()
“ Modifica el hijo derecho que tiene un Node<T> ”
{pre: ⟨El nuevo hijo derecho del Node<T> no debe ser de tipo primitivo⟩ }
{pos: ⟨El hijo derecho del Node<T> es modificado en el sistema⟩ }

TAD BinaryTree
BinaryTree = new BinaryTree<T>()
{inv : ⟨T != int,char,double…)⟩(Los valor almacenados en los nodos de este árbol binario no deben ser primitivos)}
Operaciones: • BinaryTree() N/A → BinaryTree<T> • addAnElement() Node<T> originRoot, T element → N/A • isEmptyTheTree() N/A → boolean • toDeleteAnElement() Node<T> originRoot .Node<T> → boolean • toDeleteALeaf() Node<T> father, Node<T> aux, boolean isLeftSon → N/A • toTeDeleteANodeWithOnlyOneSon() Node<T> father, Node<T> aux, boolean isLeftSon → N/A • printForLevels () Node<T> root → void • heightOfTree() Node<T> root → int • preOrder Node<T> root → void