By:
Juan Diego Lora
Santiago Prado
Jeison Lasprilla

TAD

TAD AVLTree

AVLTree = {Root = <root>, comparator = <comparator>}

{ invariant: The types of each node on avlTree have the same type.
The id of each node on avlTree is different}

- Operaciones Primitivas:
-
- CreateAVLTree: Comparator -> AVLTree
- InsertElement: Patient -> Patient
- InsertElement: Patient x NodeTree -> Patient
- getHeight : NodeTree -> Number
- rotateWithLeftChild: NodeTree -> NodeTree
- rotateWithRightChild: NodeTree -> NodeTree
- doubleWithLeftChild: NodeTree -> NodeTree
- doubleWithRightChild: NodeTree -> NodeTree
- findPatient: Texto -> NodeTree
- findPatient: Texto x NodeTree -> NodeTree
- inorder: -> Text
- inorder: NodeTree -> Text
- getBalance: NodeTree -> Number
- findMinimum: NodeTree -> NodeTree
- getRoot: -> NodeTree
- setRoot: NodeTree -> NodeTree

CreateAVLTree (comparator)
"Create a new avl tree with a comparator"
{ pre: TRUE }
{ post: avlTree={root = ⊙, comparator: <comparator> } ⋀ comparator ∈ Comparator}

InsertElement (patient)
"Redirect to insertElement(patient, nodeTree)"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: redirect to insertElement(patient, root)"

InsertElement (patient, nodeTree)
"Insert a new patient to the tree"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: avlTree (node = nodeTree) / (nodeTree.value = patient ⋀ (node ∈ root.children ∨ root = node))}

getHeight(nodeTree)
"Return the height of that node"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: -1 if height is null
else height }

rotateWithLeftChild(nodeTree)
"rotate left. And return it"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTree> }

rotateWithRightChild(nodeTree)
"rotate right. And return it"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTree> }

doubleWithLeftChild(nodeTree)
"rotate left, then right. And return it"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTree> }

doubleWithRightChild(nodeTree)
"rotate right, then left. And return it"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTree> }

findPatient(id)
"redirect to findPatient(id, nodeTree)"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: redirect to findPatient(id, root) }

findPatient(id, nodeTree)
"Search the patient by id"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTree> / nodeTree.patient.id = id}

inorder()
"Redirect to inorder(nodeTree)"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: redirect to inorder(root)}

inorder(nodeTree)
"Print the patient in every node "
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: Nodes<nodeTree1, nodeTree2 ... nodeTreeX > / nodeTree1 < nodeTree2 < ... nodeTreeX}

getBalance(nodeTree)
"get the balance. substract the height of each child of the nodeTree"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: nodeTree.leftChild - nodeTree.rightChild }

findMinimum(nodeTree)
"find the minimun node of the right child of nodeTree"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <nodeTreeX> }

getRoot()
"Return the root"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: <root> }

setRoot(nodeTree)
"set the root"
{ pre: avlTree = {root = ⊙, comparator: <comparator>} ⋀ comparator ∈ Comparator}
{ post: root = nodeTree }