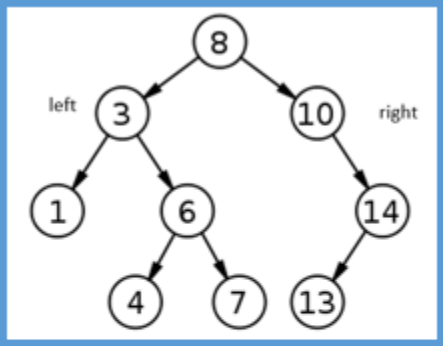
Diseño TAD

- **Binary Search Tree (BST)**

TAD Árbol BST		
		
{inv: node.left.getValue < node.right.getValue ∀ node ≠ nil}		
Operaciones:		
CrearArbol:		→Arbol
Insertar:	→Objeto	
Buscar:	→Elemento	→Objeto
Eliminar:	→Elemento	→Boolean
EstaVacio:	→Arbol	→Boolean

CrearArbol()
Crea un arbol vacio (root ≠ nil)
{pre: none}
{post: Crea un arbol vacio con todo los nodos = nil }

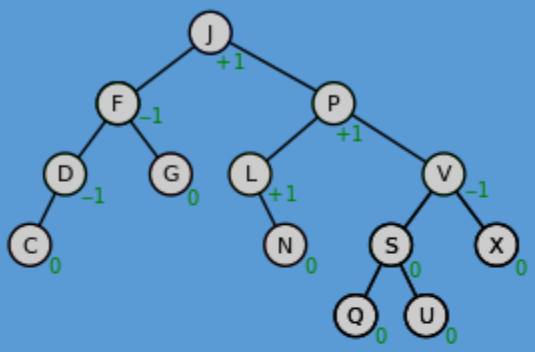
Insertar()
*Inserta un nuevo nodo en el arbol *
{pre: node ≠ nil}
{post: Se inserta el nodo en el arbol con los valores asignados al nodo }

Buscar()
*Busca un nodo en el arbol *
{pre: root ≠ nil}
{post: retorna el elemento que se busco en el arbol (Si se encuentra) }

Eliminar()
Elimina un nodo del arbol
{pre: root ≠ nil}
{post: Elimina el nodo del arbol y verifica que no se realicen cambios en las invariantes }

EstaVacia()
Retorna true si el arbol esta vacio y false een caso contrario
{pre: root ≠ nil}
{post: Retorna true si el arbol esta vacio y false si no lo esta }

- **AVL Tree**

TAD Árbol AVL		
		
{inv: -1<=node.fB>=1 ∀ node ≠ nil }		
Operaciones:		
CrearArbol:		→Arbol
Insertar:	→Objeto	
Buscar:		→Objeto
Eliminar:	→Objeto	
Minimo:		→Objeto
Maximo:		→Objeto
Predecesor:		→Objeto
Sucesor:		→Objeto

CrearArbol()
Crea un arbol vacio (root ≠ nil)
{pre: none}
{post: }

Insertar()
*Inserta un nuevo nodo en el arbol *
{pre: node.fb = 0}
{post: Inserta el nuevo nodo en el arbol con factor de balanceo 0 }

Buscar()
*Busca un nodo en el arbol *
{pre: root ≠ nil}
{post: retorna el elemento que se busco en el arbol (Si se encuentra) }

Eliminar()
Elimina un nodo del arbol
{pre: root ≠ nil}
{post: Elimina el nodo del arbol y verifica que no se realicen cambios en las invariantes }

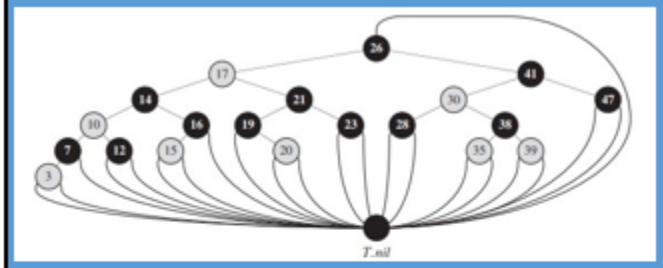
Minimo()
Busca el nodo de menor valor en el arbol
{pre: root ≠ nil}
{post: Retorna el elemento que se encuentra más a la izquierda del arbol }

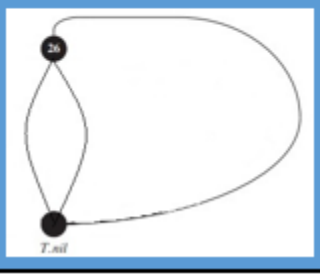
Maximo()
Busca el nodo de mayor valor en el arbol
{pre: root ≠ nil}
{post: Retorna el elemento que se encuentra más a la derecha del arbol }

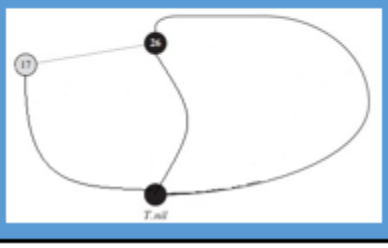
Predecesor()
Busca el padre del nodo que se ingresa por parametro
{pre: root ≠ nil}
{post: Retorna el padre del nodo que se ingresó }

Sucesor()
Busca el/los hijo del nodo que se ingresa por parametro
{pre: root ≠ nil}
{post: Retorna el/los nodos hijos del que se ingresó }

- **Red-Black Tree**

TAD Árbol Rojo-Negro		
		
{inv: root.color ≠ red }		
{inv: root.parent = nil }		
{inv: node.color = red v node.color = black ∀ node ≠ nil}		
{inv: node = nil → node.color = black}		
{inv: node.color = red → node.left = black ∧ node.rigth = black}		
{inv: node.blackHeight = node.sibling.blackHeight ∀ node ≠ root}		
Operaciones:		
CrearArbol:		→Arbol
Insertar:	→Objeto	
Buscar:		→Objeto
Eliminar:	→Objeto	
Minimo:		→Objeto
Maximo:		→Objeto
Predecesor:		→Objeto
Sucesor:		→Objeto

CrearArbol()
Crea un arbol vacio (root ≠ nil)
{pre: none}

{post: }

Insertar()
*Inserta un nuevo nodo en el arbol *
{pre: node.color = red}

{post: }

Eliminar()
Elimina un nodo del arbol
{pre: root ≠ nil}
{post: Elimina el nodo del arbol y verifica que no se realicen cambios en las invariantes }

Minimo()
Busca el nodo de menor valor en el arbol
{pre: root ≠ nil}
{post: Retorna el elemento que se encuentra más a la izquierda del arbol }

Maximo()
Busca el nodo de mayor valor en el arbol
{pre: root ≠ nil}
{post: Retorna el elemento que se encuentra más a la derecha del arbol }

Predecesor()
Busca el padre del nodo que se ingresa por parametro
{pre: root ≠ nil}
{post: Retorna el padre del nodo que se ingresó }

Buscar()
*Busca un nodo en el arbol *
{pre: root ≠ nil}
{post: retorna el elemento que se busco en el arbol (Si se encuentra) }

Sucesor()
Busca el/los hijo del nodo que se ingresa por parametro
{pre: root ≠ nil}
{post: Retorna el/los nodos hijos del que se ingresó }