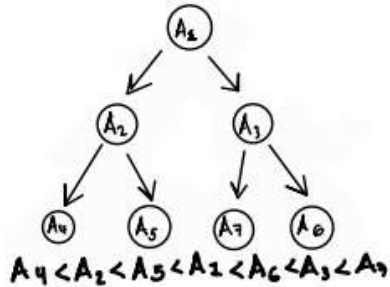


TAD Arbol AVL	
Arbol AVL = {a1, a2, a3 ... aN}	
<p>- A1 es el principal elemnto, A2 y A3 son sub arboles de A1, cualquier elemnto menor a A1 va hacia la izquierda, y si es mayor a A1 va a la derecha</p> <p>-El elemento arbol AVL tiene un factor auto-balanceable calculado con respecto a la profundidad de sus sub arboles</p>	
 <p>A4 < A2 < A5 < A1 < A6 < A3 < A7</p>	
<p>Factor auto-balanceable : $\maxDepthRightSide - \maxDepthLeftSide$</p> <p>Inv : {A1>A2, A1<A3} &&</p> <p> FactorAutoBalanceable = 1 o 0 para cualquier arbol AVL o sub arbol, hacia la izquierda del elemento. La altura de la rama izquierda no puede ser mas de una unidad que la rama derecha o viceversa.</p>	
<p>Operaciones primitivas:</p> <ul style="list-style-type: none"> -createAVL: ->AVL -Insert: Element x AVL ->AVL -delete: Element x AVL ->AVL -search: AVL ->Elemento -searchElement: AVL ->Elemento -rotateRight: Element x AVL -> AVL -rotateLeft: Element x AVL -> AVL -rebalance: Element x AVL -> AVL -RecalculateFactorBalances:Elemento x AVL -> AVL -maxDepth: Element x AVL -> AVL 	

Insert(K key,E newItem) : Modifier
"Insertar una nueva llave dentro del arbol, si la llave ya existe, se inserta una nueva posicion"
{ pre: Árbol binario AVL inicializado } { post: Incrementa la profundidad de la rama con +1 en este subárbol específico }

Delete(K key): Modifier
"Eliminar un elemento o clave especificos del árbol binario"
{ pre: Árbol binario AVL inicializado } { post: Disminuye la profundidad de la rama con -1 en este subárbol específico }

Search(K key): Analyzer
"Busca un valor de clave específico dentro del árbol binario y lo devuelve"
{ pre: Árbol binario AVL inicializado } { post: Devuelve la ArrayList de elementos o devuelve un "False" si la clave no existe }

SearchElement(K key): Analyzer
"Busca un elemento específico con un valor clave único y lo devuelve"
{ pre: Árbol binario AVL inicializado } { post: Element: El elemento con el valor clave específico, si el elemento no existe, devuelve False }

MaxDepth(Element) : Modifier
"Calcula la rama más profunda"
{ pre: Arbol AVL debe permanecer inicializado } { post: Entero de la profundidad máxima de la rama }

RotateRight(Element) : Modifier
"Saque de la cola el último elemento de la cola y elimínelo"
{ pre: Árbol binario AVL debe inicializarse y los objetivos de rotación deben existir != null } { post: Estructura binaria de tres modificada con una rotación a la derecha }

RotateLeft(Element) : Modifier
"Devuelve la longitud total de la cola en una variable entera"
{ pre: AVL Binary Tree debe inicializarse y los objetivos de rotación deben existir != null } { post: Estructura binaria de tres modificada con una rotación a la izquierda }

Rebalance() : Modifier
"Reequilibrar el árbol binario para asegurar el factor de eficiencia"
{ pre: AVL Binary Tree debe inicializarse y desbalancear } { post: Determina el caso de rotación con un Switch y llama a las rotaciones respectivas }

CreateAVL() : Constructor
"Crear (inicializar) un nuevo árbol binario AVL vacío para agregar nuevos elementos"
{ pre: TRUE } { post: NewTree: El nuevo árbol binario AVL creado listo para agregar nuevos elementos }

RecalculateFactorBalances(Element) : Modifier
"Vuelva a calcular los nuevos saldos de factores de todos los nodos padre hasta el elemento insertado"
{ pre: AVL Binary Tree debe permanecer inicializado } { post: AVL Binary Tree con los saldos de factores recalculados }