

OBTENER MENOR CLAVE del ABB

Función menorClave(Nodo): int

Si nodo.izq == null

retornar nodo.clave

retornar menorClave(nodo.izq)

Se parado en Arbol y Nodos

ArbolBB

Función menorClave(): enteros

Si raiz == null

retornar -1

retornar raiz.menorClave()

Nodo ABB

Función menorClave(): enteros

Si izq == null

retornar clave

retornar izq.menorClave()

OBTENER MAYOR CLAVE

Funcion mayorClave(nodo) : int

Si nodo.der == null

retornar nodo.clave

retornar mayorClave(nodo.der)

Arbol ABB

Funcion mayorClave

si raiz == null

retornar -1

retornar raiz.mayorClave()

Nodo ABB

Funcion mayorClave

si der == null

retornar clave

retornar der.mayorClave()

GENERAR PILA con NODOS de una RAMA

Funcion GenerarPilaDesdeRaiz(clave, nodo, pila)

Si nodo es null
retornar false

pila.push(nodo)

Si nodo.clave == clave
retornar true

Si generalPilaDesdeRaiz(clave, nodo.izq, pila) o generalPilaDesdeRaiz(clave, nodo.der, pila)
retornar true

pila.pop()

retornar false

INSERTAR en AVL

Función insertarAVL(nodo, clave)

Si nodo es null → crear nodo

Si clave < nodo.clave

nodo.izq = insertarAVL(nodo.izq,
clave)

Si clave > nodo.clave

nodo.der = insertarAVL(nodo.der, clave)

actualizarAltura(nodo)

balance := altura(izq) - altura(der)

aplicar rotaciones de ser necesario

retornar nodo

Árbol AVL

Función insertar (clave)

Si raíz == null

raíz := nuevo Nodo AVL (clave)

Sino

raíz := raíz.insertar (clave)

Nodo AVL

Si clave < this.clave

izq := izq.insertar (clave)

Sino

der := der.insertar (clave)

actualizarAltura ()

balance := altura(izq) - altura(der)

Si balance > 1 y clave < izq.clave →
rotar de rechazar)

Si balance < -1 y clave > der.clave →
rotar Derechar)

Si balance > 1 y clave > izq.clave \rightarrow

izq = izq.rotarIzquierda(); rotarDerecha()

Si balance < -1 y clave < der.clave \rightarrow

der = der.rotarDerecha(); rotarIzquierda

return this

Método Altura()

Función Altura(nodo: NodoAVL): Entero

Si nodo == null

 retornar -1

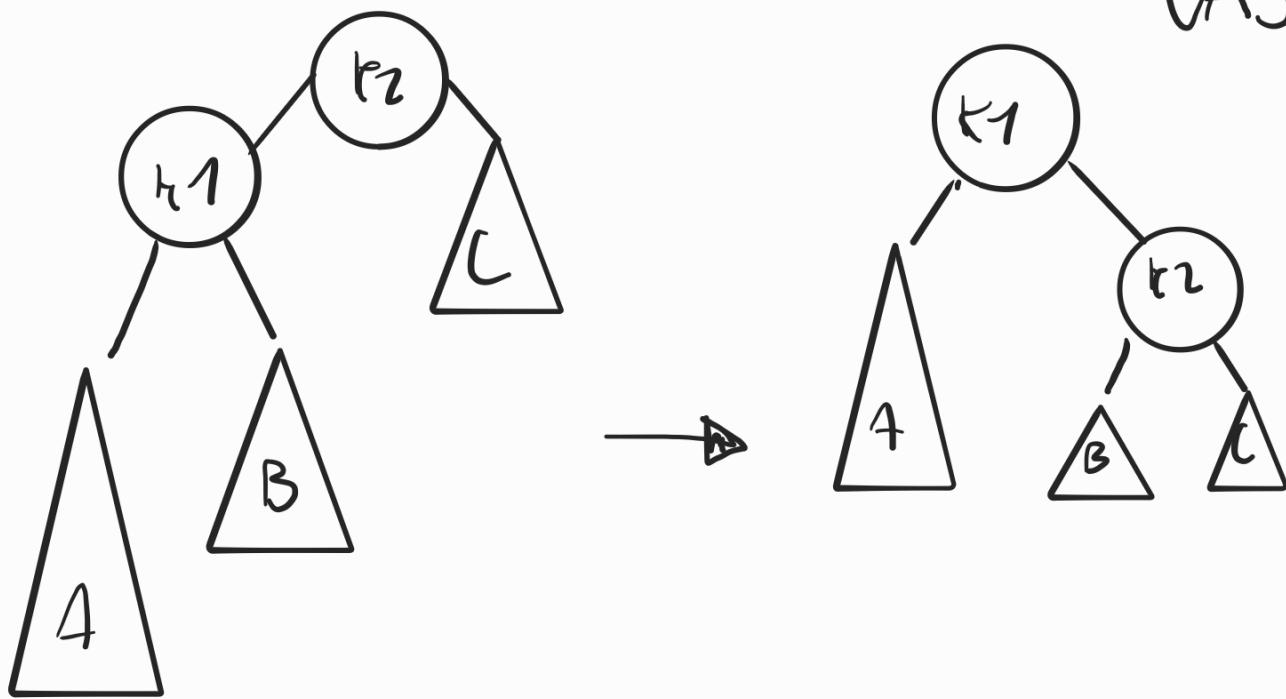
 retornar nodo.altura

Método ActualizarAltura()

altura = 1 + maximo(altura(izq), altura(der))

ROTACIONES

CASO 1



Caso 1 LL

TElemento AB rotacionLL(TElemento AB k2)

{

TElemento AB $k1 = k2 \cdot \text{hip}^{l2q};$

$k2 \cdot \text{hip}^{l2q} = k1 \cdot \text{hip}^{\text{Deri}}$

$k1 \cdot \text{hip}^{\text{Deri}} = k2$

return k1;

}

CASO 4 RR

T Elementos AB station RR (TElementos AB k1)

{

TElementosAB $k_L = k_1 \cdot h_{1g0Der};$

$k_1 \cdot h_{1g0Der} = k_2 \cdot h_{1g0l2q};$

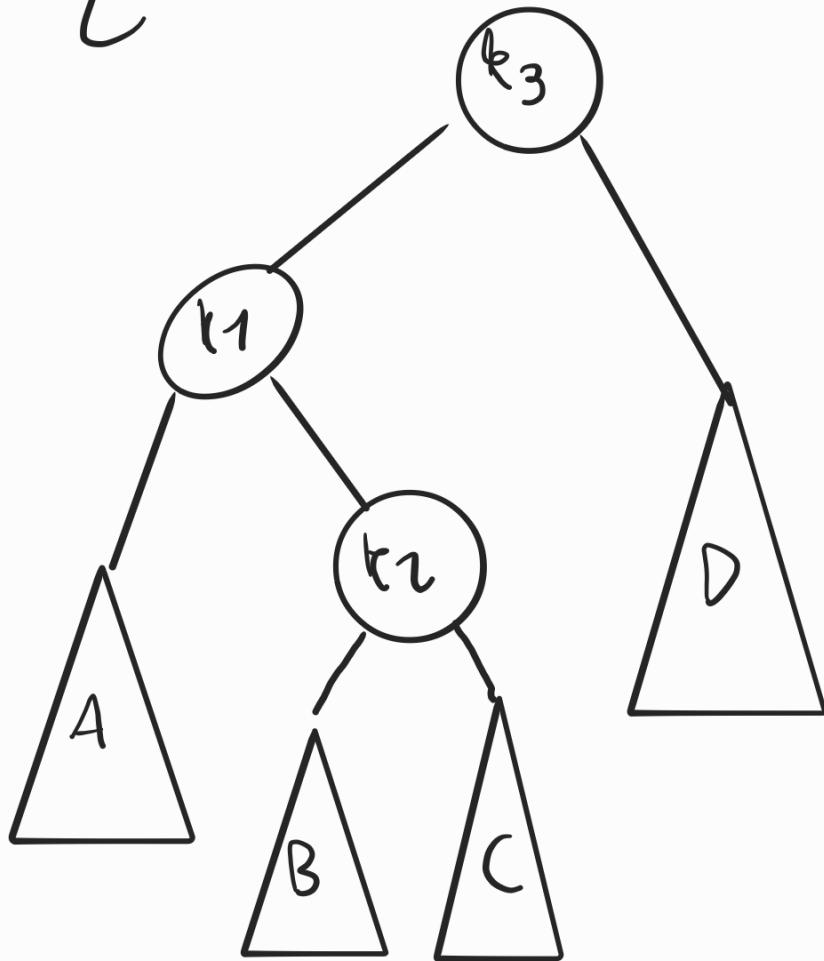
$k_2 \cdot h_{1g0l2q} = k_1$

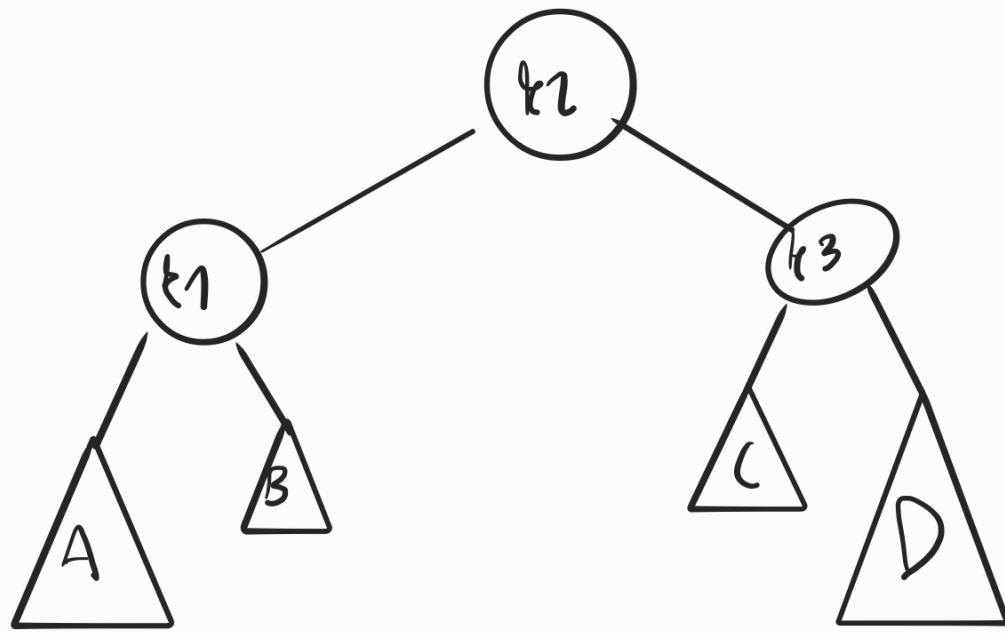
return k2

}

CASO 2

LR





CASO 2 LR

TElementos AB rotacion LR(TElementos k3)

```
{
    k3.hijoIzq = rotacionRR(k3.hijoIzq);
    return rotacionLL(k3);
}
```

CASO 3 RL

TElementos AB rotacion RL(TElementos k1)

```
{
    k1.hijoDer = rotacionLL(k1.hijoDer);
    return rotacionRR(k1);
}
```

BUSCAR ETIQUETA

Buscar(etiqueta): TElemento ABInarb
COM

Resultado = nulos

Si etiquetaqueta = Etiqueta ENTONCES

Resultado = this

Sino

Si etiquetaqueta < Etiqueta ENTONCES

Si Hijo Izquierdo <> NULS ENTONCES

Resultado = Hijo Izquierdo. Buscar(etiqueta)

Sino Finsi

Si etiquetaqueta > etiqueta ENTONCES

Si Hijo Derecho <> NULS ENTONCES

Resultado = Hijo Derecho. Buscar(etiqueta)

Finsi

Finsi

Finsi

Devolver Resultado

Fin

INSESION EN ABB

II De Arbol Binario

Inser Tar (un Elemento Arbol Binario)

COM

Si Raiz = nulo ENTonces

Raiz = un Elemento Arbol Binario

SINO

Raiz . Inser Tar (un Elemento Arbol Binario)

FINSI

FIN

De T Elementos AB

Inser Tar (un Elemento Arbol Binario)

COM

Si etiqueta = un Elemento Arbol Binario,
etiqueta

SALIR

FINSI

De Arbol Binario

Lenguaje natural:

Este procedimiento recibe un (ElementoArbolBinario) y lo inserta en el arbol

- Si el arbol esté vacío ($\text{Raiz} = \text{null}$),
lo asigna como raíz
- Si hay raíz, delega la operación al
método insertar de la raíz
($\text{Raiz.insertar}(\dots)$)

Precondiciones

- Un ElementoArbolBinario no es null
- El AB está correctamente creado

Post Condicones

- El nodo queda insertado en el arbol, respetando
el orden del ABB
- Si la raíz estaba vacía, ahora apunta al
nuevo nodo
- Si había raíz, el nodo fue ubicado correctamente
bajo ella

Tiempo de ejecución

Depende de lo que haga el método Raiz.insertar(...), es decir, el caso siguiente. Este nivel en sí solo tiene complejidad O(1) porque solo verifica y delega.

De TElemento AB

Lenguaje natural

Este método inserta un nodo en un subárbol.

- Si la clave ya existe, no se inserta
- Si es menor, se va a la izquierda
 - Si no hay hijo izquierdo, se le asigna
 - Si hay, se llama recursivamente
- Si es mayor, se va a la derecha
 - Mismo criterio que el izquierdo

Precondiciones

- Un $\text{ElementoArbolBinario}$ no es null
- La clave del nuevo nodo (Etiqueta) es comparable y no null

Postcondiciones

- Si la clave del nuevo nodo no existía, se inserta correctamente en el lugar que le corresponde según el orden binario
- Si la clave ya existía, el árbol no se modifica
- Se mantiene la propiedad del ABB

Tiempo de Ejecución

Árbol balanceado (mejor caso)

$O(\log n)$

Árbol desbalanceados

$O(n)$

Si un Elem. \rightarrow etiqueta < etiqueta ENTONCE

Si Hijo Izquierdo = nulo ENTONCES

Hijo Izquierdo \leftarrow un Elem...

SINO Hijo Izquierdo, insertar
(un Elemento Arbol Binario)

FIN SI

SINO

Si Hijo Derecho = NULO

Hijo Derecho \leftarrow Un Elem...

SINO Hijo Derecho.insertar (unEle...)

FINSI

FINSI

FIN

BUSQUEDA Y ELIMINACION EN ABB

Arbol Binario. Eliminar (UnaEtiqueta)

COM

Si Raiz <> nulo ENTonces

Raiz \leftarrow Raiz.Eliminar (UnaEtiqueta)

SINO

mensaje "arbol vacio"

FIN

Elemento AB. eliminar (UnaEtiqueta); de
tipo TElementoAB

COM

(1) Si UnaEtiqueta < etiqueta ENTonces

Si hijolizq < nulo ENTonces

hijorizq \leftarrow hijolizq.eliminar (UnaEtiqueta)

FIN SI

retornar (this)

FIN SI

(2) Si UnaEtiqueta > etiqueta ENTONCES
Si hijosDer <> nulo ENTONCES
 hijosDer \leftarrow hijosDer, eliminar(UnaEtiqueta)
 FINSI
 retornar(this)
FINSI

(3) retornar quitaElNodo()

FIN

TElementosAB quitaElNodo: de Tipo TElementosAB
Comenzar

(1) Si hijosIzq = nulo ENTONCES
 retornar hijosDer

(2) Si hijosDer = nulo ENTONCES
 retornar hijosIzq

(3) // es un nodo completo

el Hijo ← hijo Izq

el Padre ← this

mientras el Hijo.hijoDer <> null hacer

el Padre ← el Hijo

el Hijo ← el Hijo.hijoDer

FIN MIENTRAS

Si el Padre <> this entonces

el Padre.hijoDer ← el Hijo.hijoIzq

el Hijo.hijoIzq ← hijoIzq

FIN SI

el Hijo.hijoDer ← hijoDer

retornar el Hijo

FIN

RECORRIDAS ARBOL GENERICOS - PREORDEN

TA (Arbol Genericos - pre Orden ;

COM

Si Raiz <> nulos

 Raiz . preOrden ;

 FIN SI

FIN

FNodosArbolGenericos . preOrden ;

COM

 Imprimir (etiqueta);

 vnHijo ← Primer Hijo;

 Mientras vnHijo <> NULO HACER

 vnHijo . preOrden ;

 vnHijo ← vnHijo . hermanoDerecho ;

 FIN MIENTRAS

FIN

POSTORDEN

TArbol Genericos. postOrden;

COM

Si raiz <> nulos hacer

raiz. postOrden;

FIN SI

FIN

TNodoArbol Generico. postOrden;

COM

vnHijo ← PrimerHijo

Mientras vnHijo <> null hacer

vnHijo. postorden

vnHijo ← vnHijo. hermano Derecho

FIN MIENTRAS

IMPRIMIR (et1 greta)

FIN

BUSQUEDA EN TRIES

En la clase NodoTrie buscar(string Palabra): se devuelve el nodo que corresponde al ultimo carácter del argumento, el nodo nulo si ese argumento no está en el TRIE.

COM

nodo Actual ← this

Para cada carácter car de una Palabra hacer

vnHijo ← nodoActual.ObtenerHijo(car)

Si vnHijo = nulo entonces
devolver nulo

SINO

nodo Actual ← vnHijo

FINSI

FIN PARA CADA

SI nodoActual.esFinalDePalabra entonces

Actual vs other models

CHS

new vs old

FINS

FIN

INSERSIÓN EN TRIES

Comenzar

 nodoActual ← this

 Para cada carácter car de
 una Palabra hacer

 · UnHijo ← nodoActual. obtenerHijo(car)

 · Si UnHijo = nulo entonces

 · UnHijo ← crear nuevo nodo Trie

 · nodoActual.agregar(UnHijo, car)

 FINSI

 nodoActual ← UnHijo

FIN PARA CADA

nodoActual.esFinDePalabra = TRUE

FIN

IMPRIMIR TODAS LAS STRINGS del TRIE

En la clase NodoTrie imprimir
(string cadena, nodo tree nodoActual)
imprime por consola todas las
palabras contenidas en el subarbol
que tiene como raiz al nodo pasado
anteriormente.

COM

SI nodoActual == null entonces
SI nodoActual.esFINDEPalabra ENTonces
 dar salida por pantalla a la cadena

FIN SI

Para cada hijo de nodoActual hacer
 imprimir (cadena + caracter del hijo, hijo)

FIN para cada

FIN SI

FIN

Obtener Altura Arbol BB

Precond

- Que el arbol no esté vacio y exista

Post

- Altra Arbol = altura raiz

Casos de Prueba

- Si arbol tiene solo raiz la altura es 0
- Si esta vacio es -1
- Si tiene mas nodos devuelve la altura de la raiz.

COM

$A \leftarrow -1, B \leftarrow -1$

Si $hijo Izq \neq \text{nulo}$

$A \leftarrow \text{hijo Izq}.obtenerAltura$

FIN SI

Si $hijoDer \neq \text{nulo}$

$B \leftarrow \text{hijoDer}.obtenerAltura$

FIN SI

Devolver $\max(A, B) + 1$

FIN

OBTENER TAMAÑO ABB

Pre: Que el arbol exista y no sea nulo

Post: Tamaño Arbol = Tamaño Raiz

Casos de Prueba

Si el arbol no tiene raiz el tamaño es 0

Si el arbol solo tiene raiz el tamaño es 1

Si el arbol tiene mas nodos devolver el tamaño de la raiz

Obtener Tamaño Elemento

COM

Tamaño \leftarrow 1

Si hijolza \neq nulo

Tamaño_{s+1} = hijolza. Obtener Tamaño

FIN SI

Si hijoDerecho <> null

Tamaño += hijoDerecho, obtenerTamaño

FIN SI

Devuelve Tamaño

FIN

ObtenerTamañoArbol

COM

Si raíz = null

Devuelve 0

SINO

Devuelve raíz, ObtenerTamaño

FIN SI

FIN

Obtener Índice Curso

Método obtenerÍndiceWISO(WISO wisoID: String)
→ TablaBB <Alumno>

Inicio

wiso ← null

// Paso 1: Busca el wiso con el ID

Para cada wiso en this.cursos hacer

Si wiso.id = cursoID entonces

wiso ← ese wiso

Salir del ciclo

FIN SI

FIN PARA

// Paso 2: Validación

Si wiso == null entonces

devolver null

FIN SI

// Paso 3 : Crear Arbol

arbol ← nuevo TArbolBB<Alumnos>()

// Paso 4 : Insertar alumnos del WISO

Para cada alumno en WISOS.alumnos

hacer

clave ← alumnos.nombre + alumnos.apellido

arbol.insertar(clave, alumno)

FIN PARA

devolver arbol

FIN

Lenguaje natural

Recibe un WISO ID y busca ese WISO
en una lista de WISOS.

Si lo encuentra

1. Crea un ABB

2. Inserta los alumnos en el ABB

3. Utiliza como clave la concatenación
del nombre + apellido de cada
alumno para ordenar

4. Devuelve el árbol

Si el WSG no existe, devuelve null

Precondiciones

- El parámetro WSGID no es null
- La lista this.cursos está correctamente inicializada
- Cada WSG tiene una lista de alumnos válida
- Cada alumno tiene definido nombre y apellido

Post condiciones

- Se devuelve un Árbol con todos los alumnos del WSG ordenados

- Si el WSO no se encuentra, se devuelve null
- El arbol contiene solo los alumnos de ese WSO

Analisis del tiempo de ejecución

- n = cantidad de WSOs
- m = cantidad de alumnos del WSO

Paso 1

- Se recorre toda la lista
- El peor caso, no esta: $O(n)$
- En el mejor caso, es el primero: $O(1)$

Paso 2

Mejor caso (arbol balanceado): $O(m \log m)$

Peor caso (desbalanceado): $O(m^2)$

Tiempo to tal

Mejor caso	$O(h) + O(m \log n)$
Pesr Caso	$O(1) + O(m^2)$

PARCIAL CALCULAR PARENTESCO

función CalculaParentesco(raiz, p1, p2) devuelve
(entero, raiz)

pila1 \leftarrow pila vacía

raiz.buscarLinea(p1, nombre, pila1)

pila2 \leftarrow pila vacía

raiz.buscarLinea(p2, nombre, pila2)

si pila1.esVacia() o pila2.esVacia()

mostrar "Personas no existentes"

devolver (0, "desconocido")

FINSI

sonIguales \leftarrow verdadero

mientras sonIguales \wedge no pila1.esVacia()
 \wedge no pila2.esVacia() hacer

$a \leftarrow \text{pila1.pop}()$

$b \leftarrow \text{pila2.pop}()$

Si $a = b$ entonces

son iguales \leftarrow Falso

FNSI

FN MIENTRAS

Devolver(pila1.largo + 1) + (pila2.largo + 1)

funcion buscarLinea(nombre, pila) devuelve booleano

Si $this.nombre = nombre$ entonces

pila.push(this)

devolver verdadero

Fin Si

esAncestro \leftarrow falso

Si $this.madre = null$ entonces

esAncestro \leftarrow this.madre.buscarLinea(nombre, pila)

Si esAncestro entonces

pila.push(this)

devolver verdadero

FIN SI

FIN SI

Si this.padre=null entonces

esAncestro ← this.padre.buscarLinea(nombre, pil)

si esAncestro entonces

pila.push(this)

devolver verdadero

FIN SI

FIN SI

devolver falso

RECORRIDO INORDEN A B B

De TElementoAB

public string inOrden()

COM

tempStr := ""

Si hijoIzq = null ENTONCES

tempStr := hijoIzq.inOrden()

FINSI

tempStr := tempStr + Imprimir()

Si hijoDer = null ENTONCES

tempStr := tempStr + hijoDer.inOrden()

FINSI

retornar Str

FN

De TArbolBB

public String InOrden()

COM

Si raiz == null ENTONCES

retornar "arbol vacio"

SINO

retornar raiz. InOrden()

FM

VERIFICAR SI UN ABB es SIMETRICO

Funcion isMirror(arbol1, arbol2):

Si ambos arbol1 y arbol2 son null:

Retornar verdadero

Si solo uno de los dos es null:

Retornar falso

Retornar isMirror (arbol1.izquierdo,) \wedge
arbol2.derecho)

isMirror (arbol1.derecho,
arbol2.izquierdo)

Funcion isSymmetric(raiz):

Si raiz es null:

Retornar verdadero

Retornar isMirror(raiz.izquierdo,raiz.derecho)

OBTENER NIVEL

TELENTOS AB

Función obtenerNivel (clave, nivelActual) → entero

Iniciar

Si clave < etiqueta entonces
 retornar nivelActual

SINO si clave < etiqueta entonces
 si hijo Izq ≠ null entonces
 retornar hijo Izq . obtenerNivel (clave,
 nivelActual +1)

FINSI

SINO clave > etiqueta
 si hijoDer ≠ null entonces
 retornar hijoDer . obtenerNivel (clave)
 nivelActual +1)

FINSI

FINSI

 retornar -1 // clave no encontrada

FIN

De TArbol BB

funcion ObtenerNivel (clave) \rightarrow entero

Si raiz es nulo

devuelve 0

devolver raiz. ObtenerNivel(clave, 0)

CONECTAR NODOS DEL MISMO NIVEL

Función ConnectNodes(raíz):

Si raíz = null:

retornar

crear cola q

Encolar root

mientras q no este vacía:

size ← tamaño de q

prev ← null

Repetir size veces:

nodo ← desencolar

Si prev = null:

prev.nextRight ← nodo

prev.nodo

Si nodo.izquierdo = null

Encolar nodo.izquierdo

Si $nodo_derecho = null$

Encolar $nodo_derecho$

prev.nextRight = null

SERIALIZAR Y DESERIALIZAR UN ABB

Función serialize(nodo, lista):

Si nodo es null:

Agregar -1 a lista

Retornar

Agregar nodo.valor a lista

serialize(nodo.izquierdo, lista)

serialize(nodo.derecho, lista)

Función deserialize(lista, índice)

Si índice apunta a -1:

Avanzar índice

Retornar null

crear nodo con valor lista[índice]

Avanzar índice

nodo.izquierdo ← deserialize(lista, índice)

nodo.derecho ← deserialize(lista, índice)

Retornar nodo

TRIES - CONTAR CUANTAS
PALABRAS SON SUFIJOS
o PREFIJOS de PALABRAS
EN OTRA LISTA.

Función suffixPrefixString(s1, s2)
contador ← 0

Para cada cadena s en s_2 :

Para cada cadena t en s_1 :

Si t empieza con s o termina
con s

contador ← contador + 1

romper bucle interno

Retornar contador

