

## UT5-PD1

LN: Este método se trata de un booleano con dos parámetros como parámetro siendo estos una etiqueta y Etiqueta Padre respectivamente. Tiene la función de insertar una etiqueta bajo una etiqueta padre existente en el tree. Si la etiqueta padre no existe, la inserción no se realiza y devuelve "falso". Si la etiqueta padre se encuentra, la nueva etiqueta se inserta como una palabra descendiente del nodo correspondiente a la etiqueta padre, y el método devuelve "true".

### Precondiciones

- "unaEtiqueta" y "etiquetaPadre" no deben ser nulos
- El tree puede estar vacío

### Postcondiciones

- Se inserta un nuevo nodo en el árbol con la etiqueta "unaEtiqueta".
- "unaEtiqueta" se inserta como una nueva palabra descendiente de este nodo, devolviendo true.
- Si el nodo correspondiente a "etiquetaPadre" no existe, el método devuelve falso.

## Lección 6

TArbolTree. insertar (unaEtiqueta, etiquetaPadre : Comparable)  
com

Si raiz = nulo entonces

raiz ← nuevo TNodoTree();

Fon II

• TNodoTree nodoPadre ← raiz. buscarNodoTree(etiquetaPadre, taString);

Si nodoPadre <> nulo entonces

nodoPadre.insertar(unaEtiqueta, taString());

Devolver true

Fon II

Devolver false

Fon

Caso de prueba:

Caso 1: Inserta la raiz que es el primer elemento, pero en vez de asignarle un parámetro nulo del padre pone un dato de etiqueta, devuelve false.

Caso 2: Inserta varios elementos, primero la raiz con parámetros "", una Etiqueta) y los demás nodos hijos de este o sus descendientes de forma correcta pasando parámetros de padre correctos, dará como resultado true e insertará todos los nodos.

Caso 3: Inserta la raiz, luego se quiere insertar un nodo que tenga un padre que no está en el árbol, devuelve false y no lo inserta.

## Buscar

LN: Recorre todos los elementos si encuentra un nodo con la etiqueta, lo devuelve sino devuelve nulo.

## Precondiciones

- Ninguna

## Postcondiciones

- Devuelve el nodo si encuentra coincidencia
- Devuelve nulo si no encuentra coincidencia
- No se modifica el arbol

## seudocódigo:

TArbol tree

TArbol tree. buscar (palabra: String)

Tom

Si raiz <> nulo entonces

Devolver raiz. buscar (palabra)

Fin Si

Devolver O

## TNodo tree

TNodo tree. buscar (S: String)

Tom  
TNodo tree nodo ← tree

int comparaciones ← 0

Raiz. int c = 0; S.length(); ++ hacer

int indice ← getindice (S.charAt(c))

nodo ← nodo. hijos [indice]

comparaciones ++

Si nodo <> nulo entonces

Devolver comparaciones

Fin Si

Fin para

Devolver nodo. qPalabra ? comparaciones : -1

Fin

\*Arbol tree

• ~~Arbol~~ Tree. buscar (valorBuscado: String)

Si raiz = nulo entonces

devolver nulo

Sino

Devolver raiz. buscar (valorBuscado)

Fin N

Fin

\*\*NodoTree

• ~~Nodo~~Tree. buscar (etiquetaBuscada: String)

Si this.etiqueta.equals (etiquetaBuscada) entonces

devolver this

Fin N

Para ~~Nodo~~Tree (T) tipo: hijo Leer

~~Nodo~~Tree (T). resultados < fijo. buscar (etiquetaBuscada)

Si resultados < 7 nulo entonces

devolver resultados;

Fin N

Fin Para

devolver nulo

Fin

## Lector indentado

LN: Recorre el árbol por niveles agregando a un string los etiquetas que luego se van a devolver.

## Precondiciones

- Ninguna

## Postcondiciones

- Devuelve un string si el árbol no está vacío con las etiquetas de los nodos por niveles
- Devuelve nulo si está vacío.
- No modifica el árbol.

## Código

TNodeTree. LectorIndentado(nivel: int)

lcm

StringBuilder sb = nuevo StringBuilder()

Para int i = 0; i < nivel; i++) hacer

sb.append(" ")

Fin Para

sb.append(etiqueta).append("- " + m")

Para TNodeTree < T > hijos: hijos hacer

sb.append(hijo. LectorIndentado(nivel + 1))

Fin Para

Devolver sb.toString()

Fin

Arbol Tree. Busto/indentado(1)

lom

Si raiz = nulo entonces  
Devolver nulo

Lino

Devolver raiz. Busto/indentado(0):

Fim 1:

Fim

Caso de prueba

Caso 1: Se tiene un arbol con varios elementos, al borrar devuelve en string con todos los elementos ordenados por nivel.

Caso 2: Tenemos un arbol vacio, devuelve nulo

# UT5 - PD2

II

1) tree

raiz: nodo

Nodo

etiquetas: char

hijos: array nodo

páginas: array int

finPalabra: bool

2) LN; a partir de un texto, construye el índice del mismo en el tree, indicando los páginas donde aparecen las palabras.

Precondiciones

- Solo puede tener letras minúsculas las palabras, tampoco debe haber  $\tilde{m}$ .

Postcondiciones

- El tree tendrá nuevos nodos en función de la palabra agregada.

Lenguaje

Contexto  $\boxed{}$   $\leftarrow$  entrada.txt

tree  $\leftarrow$  nuevo tree

contador  $\leftarrow$  0

página  $\leftarrow$  0

Fin

for

Para cada palabra en texto [ ] hacer  
n. contador = 200 entonces

página++  
contador ← 0

for n

true. insertar (palabra, página)  
contador ++

Fin Para cada

Fin

,

Tree

Insertar (unaPalabra: string, pagina: int)

com

raiz. insertar (unaPalabra, pagina)

Fin

Nodo

Insertar (unaPalabra: Cadena, pagina: entero)

com

nodoActual ← thy

Para cada carácter c en unaPalabra Hacer

p ← obtenerHijo(c)

unHijo ← nodoActual. hijo(p)

Si unHijo = nulo entonces

unHijo ← nuevo TNode(c)

nodoActual. hijo(p) ← unHijo

Fin Si

nodoActual ← unHijo

Fin Para

nodoActual. establecer ← true

modoActual. indice. agregar (pagina)

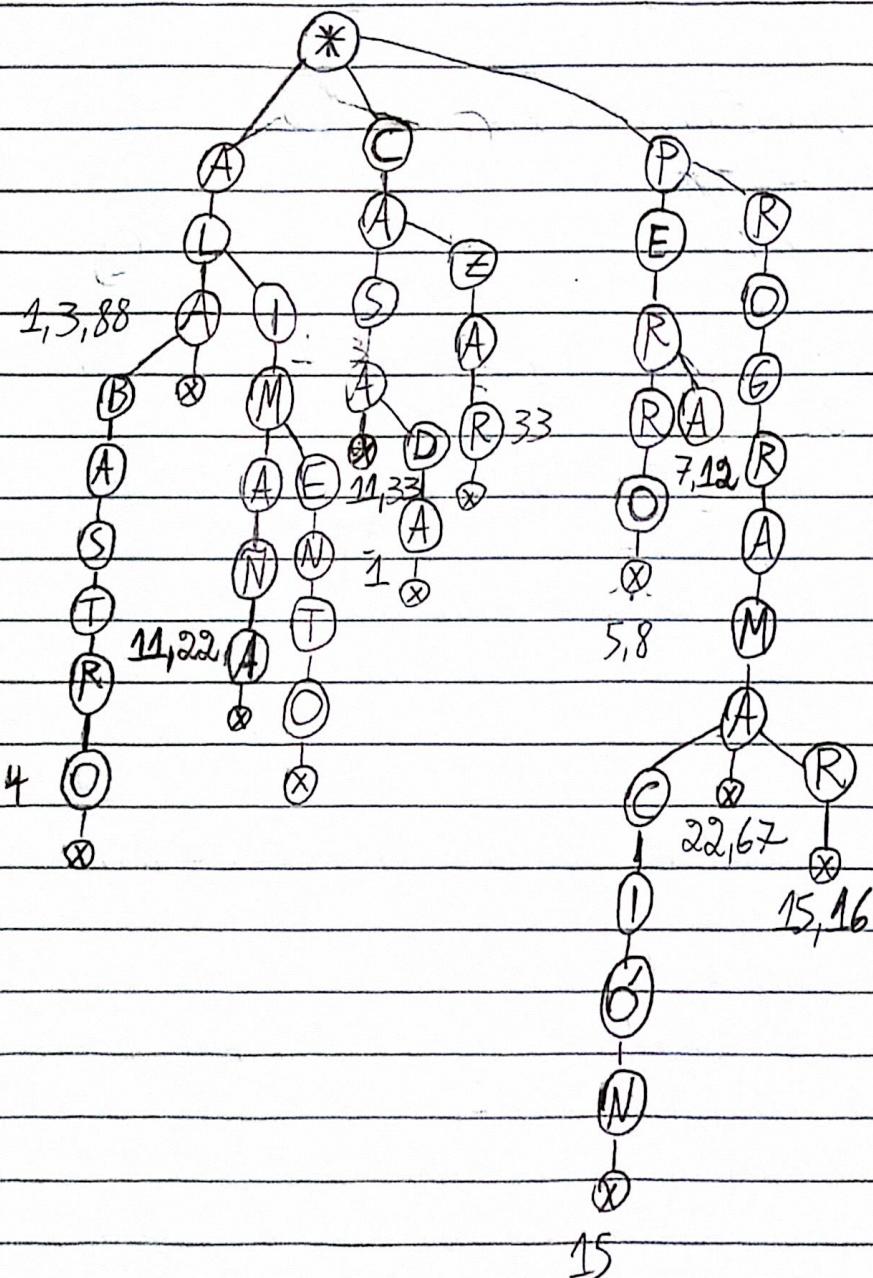
obtener hijo (un carácter: C)

Com

$P \leftarrow C.\text{valorASCII}() - a.\text{valorASCII}()$

Deshacer P

Fin



1) Realiza 8 comparaciones, una por cada letra de la palabra "Programa", ya que está en el árbol

2) Realiza 4 comparaciones antes de determinar que la palabra no está en el árbol.

3) Para insertar la palabra "cazadores" en el árbol, debemos crear nuevos nodos para cada letra de la palabra. Por lo tanto se realizan 9 comparaciones, para encontrar el lugar correcto para insertar la palabra en el árbol.

4) La altura del árbol es 12, ya que la palabra más larga del árbol es "Programación" que tiene 12 letras.

5) Su tamaño es de 47 nodos, cada letra de cada palabra en el árbol se representa como un nodo.

### Ejercicio 2

N: Dado un árbol, buscas una cadena y si existe devuelve los números de las páginas del libro en las que se encuentra la cadena.

### Precondiciones:

- Que los caracteres del árbol sean parte del alfabeto
- El árbol debe estar construido con letras solo mayúsculas o solo minúsculas
- La palabra pasada por parámetro debe ser del mismo tipo que las que están construyendo el árbol.

### Postcondiciones

- Si la raíz es nula devuelve nula,
- Si la raíz no es nula y la palabra existe, esta devolverá los índices de las páginas en la que está.
- Si la raíz no es nula y la palabra no se encuentra, devolverá nula.

Secuencias:

tree.buscarPalabra(unaPalabra)

comienzo

Si raiz < nulo entonces

Devolver raiz.buscarPalabra(un texto, unaPagina)

fin

Devolver nulo

Fin M

Fin

Node.buscarPalabra(unaPalabra)

comienzo

modoActual < hijo

- Para cada caracter c de unaPalabra hacer
- unHijo < modoActual.hijo([obtenerHijo(c)])

Si unHijo = nulo entonces

devolver nulo

fin

modoActual < unHijo

fin M

Fin para

Si modoActual.esFinDePalabra entonces

devolver pagina

Si no

Devolver nulo

Fin M

Fin

obtenerHijo (Carácter)

lcm

indice ← carácter.valorASCII - A.valorASCII  
devolver indice

Fin

extraerDePalabra()

lcm

devuelve fntPalabra

Fin