UT4_PD3

Ejercicio #1 - Método INSERTAR

Lenguaje natural

1. El método recibe un nodo nuevo como parámetro.
2. Si el árbol no tiene raíz, se le asigna el nuevo nodo como raíz y se termina el algoritmo.
3. Si el valor del nuevo nodo es mayor que el valor del nodo actual, se verifica que:
    a. ¿Es nulo el hijo derecho del actual? → En caso positivo el nuevo nodo se convierte en el hijo derecho del nodo actual y se termina. En caso negativo llamo recursivamente al método pasando por parámetro su hijo derecho.
4. Si el valor del nuevo nodo es menor que el valor del nodo actual, se hace lo mismo que en el paso anterior, pero con el hijo izquierdo.
5. Si el valor del nodo nuevo es igual al valor del nodo actual, no se hace nada.

Precondiciones

- El nodo recibido no es nulo.
- El valor que contiene no es nulo y es del mismo tipo que los demás nodos del árbol, por lo que su valor es comparable.

Postcondiciones

- El nodo se inserta correctamente:
    - A no ser que el árbol esté vacío. El nuevo nodo ahora tiene un nodo padre **x** presente en el árbol tal que el nuevo nodo es hijo izquierdo o derecho de este.
    - Si el valor del nuevo nodo es mayor que el de **x**, entonces es su hijo derecho, en lo contrario es su hijo izquierdo.

- Si el nuevo nodo no tiene hiijos, entonces queda insertado como hoja en el árbol.

```
insertarNodo(nuevoNodo: nodoBinario) → void
COM
   SI nuevoNodo.valor > this.valor ENTONCES
      SI this.der == nulo ENTONCES
         this.der ← nuevoNodo
   SINO
    this.der.insertarNodo(nuevoNodo)
   FIN SI
   SINO SI nuevoNodo.valor < this.valor ENTONCES
      SI this.izq == nulo ENTONCES
         this.izq ← nuevoNodo
    SINO
       this.izq.insertarNodo(nuevoNodo)
    FIN SI
   FIN SI
FIN

-------------------------------------------
// METODO DE ARBOL
-------------------------------------------
insertarNodoEnArbol(nuevoNodo: nodoBinario) → void
COM
   SI this.raiz == nulo ENTONCES
      this.raiz ← nuevoNodo
 SINO
   this.raiz.insertarNodo(nuevoNodo)
 FIN SI
```

Análisis de tiempo de ejecución

En cada llamada del método, se dividie el problema en dos, ya que si se avanza por uno de los dos hijos de un nodo, se descarta tener que buscar en cada uno de los nodos de uno de los dos subárboles del **actual**. Esto le da al algoritmo un tiempo de ejecución promedio de → $\Theta(logN)$.

No obstante, en el peor caso, el árbol está desbalanceado de una manera que se “degeneró” en una lista enlazada, en ese caso lo recorre secuencialmente → $O(N)$

Por otro lado, en el mejor de los casos, el árbol no tiene raíz, por lo que sólo se necesita asignar el nuevo nodo como raíz del árbol. → $O(1)$

Ejercicio #2 - Contar hojas

Lenguaje natural

Recorro recursivamente el árbol en POSTORDEN y me fijo si el nodo es nulo, en tal caso retorna 0. Si el nodo actual es hoja o, retorno 1. → CASO BASE

Visito primero todo el subárbol izquierdo, luego el derecho, sumando las hojas de cada uno.

Precondiciones

- El árbol no puede ser nulo

Postcondiciones

- El árbol se mantiene igual que antes, es decir que no se modifica.
- Retorna un entero que representa la cantidad de nodos sin hijos.

Pseudocódigo

```
contarHojas(nodo: nodoBinario) → int
COM
   SI nodo == nulo ENTONCES //O(1)
      RETORNAR 0 //O(1)
   FIN SI

   hojasIzq = contarHojas(nodo.izq)
   hojasDer = contarHojas(nodo.der)

   SI nodo.izq == nulo Y nodo.der == nulo ENTONCES // O(1)
      RETORNAR hojasIzq + hojasDer + 1 // O(1)
   FIN SI
   SINO
```

```
        RETORNAR hojasIzq + hojasDer // O(1)
    FIN SINO
FIN
```

Análisis de tiempo de ejecución

El tiempo de ejecución es $O(n)$ porque debe visitar necesariamente todos los nodos del árbol.

Ejercicio #3 - Suma elementos

Lenguaje natural

Recorro el árbol en POSTORDEN de manera recursiva. Si el nodo que recibe es nulo, retorna 0. → **CASO BASE**

Luego suma el valor total de los elementos de los subárboles izquierdo y derecho llamándose a sí mismo con los hijos izquierdo y derecho del nodo actual como parámetros.

Finalmente retorna la suma de todos los elementos de sus descendientes más su propio valor.

Precondiciones

- Recibe un nodo cuyo valor almacenado no es nulo
- El nodo es beige

Poscondiciones

- El nodo no se modifica.
- Se retorna la suma de todos los valores almacenados en cada uno de los nodos del árbol.

```
sumaValores(nodo: nodoBinario) → int
COM
    SI nodo == nulo ENTONCES
        RETORNAR 0
    FIN SI

    valorIzq ← sumaValores(nodo.izq)
```

```
    valorDer ← sumaValores(nodo.der)

    RETORNAR valorIzq + valorDer + nodo.valor
FIN
```

Análisis de tiempo de ejecución

Al igual que el algoritmo anterior, el tiempo de ejecución es $O(n)$ porque debe visitar necesariamente todos los nodos del árbol. Aunque en el mejor de los casos el nodo es nulo, y el algoritmo se finaliza con una operación de retorno ($O(1)$).

Ejercicio #4 - Cantidad de nodos en nivel

Lenguaje natural

- Se recibe como parámetros el árbol (o nodo raiz) y el nivel donde se desea contar la cantidad de nodos.
- Se inicia desde el nodo raíz, y se verifica recursivamente la cantidad de nodos por nivel en cada subárbol. Se recorre el árbol en POSTORDEN, porque visita primero el subárbol izquierdo, luego el derecho y finalmente procesa el nodo actual.
- En cada llamada se disminuye el nivel hasta que se llegue al nivel objetivo →CASO BASE
- Al final se retorna la suma de la cantidad de nodos de cada subárbol

Precondiciones

- Se recibe un árbol binario válido y no vacío
- Se recibe un nivel de tipo entero y mayor o igual a cero

Postcondiciones

- Se retorna la cantidad de nodos en el nivel especificado
- Si el nivel que recibe como parámetro es mayor que el nivel máximo del árbol, se retorna 0

Pseudocódigo

```
contarNodosNivel(nodo: NodoBinario, nivel: int) → int
COM
   SI nodo == nulo ENTONCES
      RETORNAR 0
   FIN SI

   nodosIzq ← contarNodosNivel(nodo.izq, nivel -1)
   nodosDer ← contarNodosNivel(nodo.der, nivel -1)

   SI nivel == 0 ENTONCES
      RETORNAR 1
   FIN SI

   RETORNAR nodosIzq + nodosDer
FIN
```

Análisis de tiempo de ejecución

Dado que el algoritmo recorre todos los nodos en Postorden como los anteriores, se trata de un tiempo de ejecución de orden líneal → $O(n)$