Programación II

Práctica 04: Arboles

Versión del 01/04/2017

Ejercicio1: Arbol Binario<T>

A medida que realizamos TADs más complejos, es necesario definir TADs auxiliares como soporte del TAD que se quiere definir.

Vamos a implementar un modelo del árbol binario especificado en la clase teórica.

EspecificacíonTAD AB<T>

```
AB<T>() {}
Nodo<T>agregar(T elem) {}
Integer cantNodos()

T buscar(T elem) {}  //devuelve null si no encuentra
Boolean pertenece(T elem)
Integer altura()
T minimo()        // Asumir para este método que T = Integer
T iesimo()   // según inorden, no se permiten estruct auxiliares!

Adicionales

T moda()  //  tiene la misma cantidad antes y después de T
Boolean balanceado()
Nodo<T> eliminar(T elem)  //devuelve el nodo eliminado
Boolean invariante   // es verdadero si lo cumple
```

a) Implementar el TAD AB<T> o AB<Integer>
b) Calcular la complejidad de los método

Recordar los siguientes ejemplos para justificar la complejidad, donde n es la cantidad de nodos:
-Si no pasa por todos los nodos ni quiera una vez, es a lo sumo O(n), siendo log(n) ***si puedo demostrar*** que en cada recursión/iteración "descarto" la mitad de los nodos
-Si paso por todos los nodos exactamente 1 vez, es O(n)
-Si para cada nodo paso a lo sumo n veces, es $O(n^2)$

Ejercicio2: Árbol Binario de búsqueda

Implementar ABB<T> extendiendo AB<T>

A efectos prácticos, para no tener que definir en esta práctica la comparación entre elementos de tipo T, implementar el ABB de Enteros únicamente: ABB<Integer> extends AB<Integer>

a) ¿Cuales métodos hubo que sobrescribir?
b) Calcular la complejidad de todos los métodos para el caso de si el ABB esta completo
    a. ABB completo
    b. ABB no completo

Ayuda: Para la justificación del calculo del orden para el ítem c, utilizar los apuntes de las clases teóricas (o de la bibliografía [Cormen2001]).

Ejercicio4

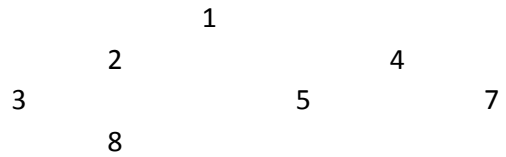
Implementar los siguientes métodos y calcular su complejidad:

```
void balancear()
ABB balancear() // Devuelve un nuevo árbol.
```

Que luego de ejecutar balancear, el método balanceado debe devolver ***true***.

```
int ramaMasCorta()
```

Dado una instancia de AB devuelve la longitud de la rama más corta comenzando desde la raíz y llegando hasta una hoja.



Ejemplo1

En el Ejemplo1 la rama más corta es 1,4,7 o 1,4,5

Ejercicio6: Arbol-n-ario de búsqueda

A diferencia del árbol binario, que tiene hasta dos nodos por nivel, el arbol n-ario tiene hasta **n** nodos por nivel.

Además, el nodo padre tiene que ser menor que todos los nodos hijos.

Ayuda: Especificar el TAD NodoN primero

EspecificaciónTAD ANB<T>

```
ANB int(Integer n){}          // Constructor: hasta n hojas por nivel
NodoN<T> eliminar(T i) {}
Void agregar(T i) {}
NodoN<T> buscar(T i) {}
```

Ejemplo para un árbol de orden 3(n = 3).

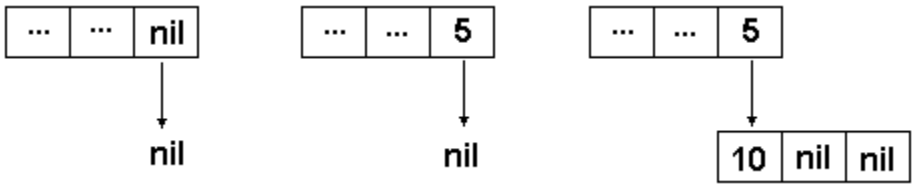


Figura 1

En la Figura1 agregamos el 5, lo cual completa el nodo y genera un nuevo nodo vacio.

Luego agregamos e 10 al comienzo del nuevo nodo.

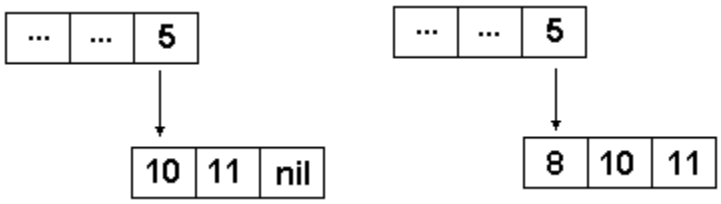


Figura2

En la Figura2 agregamos el 11.

Luego, cuando se agrega el 8, desplazamos el 10 y el 11 para poner el 8 al comienzo.

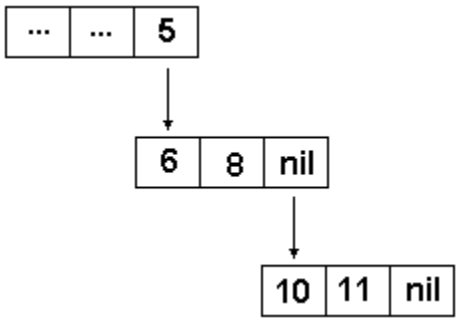


Figura3

En la Figura3 cuando agregamos el 6, tenemos que hacer dos cosas:

1) Generar un nuevo nodo
2) Repartir 6,8, 10 y 11 entre los dos nodos disponibles.

En general se intenta dejar la mitad de los elementos en cada nodo.

En este caso tenemos cuatro elementos, asi que dejaremos dos elementos en cada nodo.

Implementar el TAD ANB, calcular el orden de las operaciones.

Ayuda: El orden de complejidad tendrá dos variables: La cantidad de nodos y la cantidad de nodos por nivel.