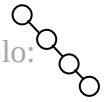
UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACION
ALGORITMOS Y ESTRUCTURAS DE DATOS

Caracas, 27 de Mayo de 2014

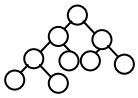
PARCIAL #3

1. Indique si la siguiente afirmación es verdadera o falsa, y justifique su respuesta (1pto. c/u)

a. La altura máxima de un árbol de búsqueda que puede formarse con *n* claves distintas es justamente *n*-1.

Cierto…este es el caso de un árbol de búsqueda degenerado. Ejemplo: 

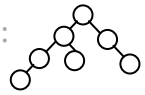
b. Un árbol AVL es perfectamente balanceado.

Falso. Ver contraejemplo: 

c. La operación de inserción en árboles AVL es más ineficiente en tiempo que en árboles de búsqueda por requerir rotaciones simples y dobles.

Falso. La inserción en AVL es O(logN), mientras que en árboles de búsqueda es O(N). Así que es más eficiente en árboles AVL.

d. Un árbol AVL es de altura mínima.

Falso. Ver contraejemplo: 

e. Los árboles sólo pueden recorrerse en forma recursiva.

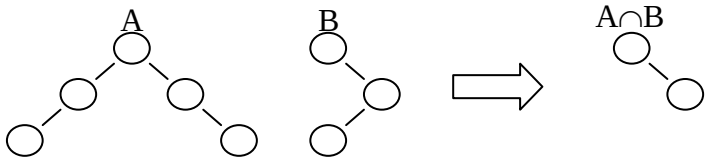
Falso. Algunos algoritmos (como el de inserción en un árbol de búsqueda) se pueden hacer en forma iterativa, "bajando" por la rama donde se va a insertar. Para aquellos algoritmos que requieren recorrer todo el árbol, se puede utilizar una pila para simular la recursión con un algoritmo iterativo. En general, cualquier algoritmo recursivo puede convertirse en iterativo utilizando una pila, tal y como lo vimos en clase.

f. Un árbol binario de altura *h* debe tener $2^{h+1} - 1$ nodos para poder ser considerado de altura mínima.

Falso. Ver contraejemplo:  tiene altura 1, pero no tiene 3 nodos.

2. Dados dos árboles binarios A y B, realice una función que devuelva la intersección de ambos árboles. Asuma que cada nodo contiene únicamente los apuntadores izquierdo y derecho (ignore el campo info). (5 ptos.)

Ejemplo:




```
Function Intersect(Nodo ^A, Nodo ^B) : Nodo ^
        If (A==NULL or  B==NULL) then
                Return NULL;
        Else
                Nodo ^x = new Nodo;
                X^.izq = Intersect(A^.izq, B^.izq);
                X^.der = Intersect(A^.der, B^.der);
                Return x;
        Endif
EndFunction
```

3. La junta de condominio desea saber cuántas personas viven en cada uno de los apartamentos del edificio X, para tener un control más eficiente en materia de seguridad. La junta de condominio desea que ud. plantee una estructura de datos eficiente para satisfacer los siguientes requerimientos:

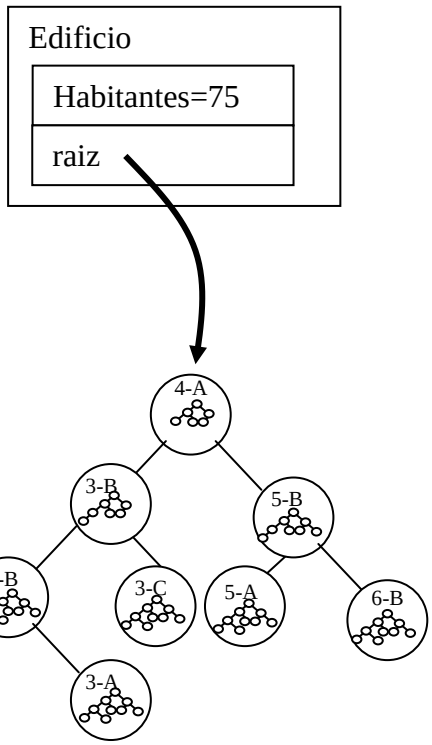
a) Operaciones de insertar/eliminar apartamento en orden log(n) a lo sumo. Cada apartamento está identificado por un string de la forma "piso-letra". Ejemplo, "3-A". Incluya cualquier información adicional que deba tener un apartamento para satisfacer el resto de los requerimientos.
b) Dado un apuntador de apartamento imprimir todas las personas que viven allí en orden alfabético.
c) Dada la identificación de un apartamento (ejemplo "3-A"), insertar/eliminar una persona de dicho apartamento en orden log(max(n,m)), donde n es el número de apartamentos, y m el número de personas que habitan en dicho apartamento.
d) Determinar en O(1) cuántas personas viven en el edificio.

Considere que la cantidad máxima de personas que pueden vivir en cada apartamento no está predefinido, ya que los apartamentos en X superan los $1000m^2$. Pese a que se conoce el número de apartamentos de X, se quiere una solución general que pueda ser instanciado en cualquier otro edificio, por lo que el número de apartamentos no es constante. Especifique la estructura de datos tanto gráficamente como formalmente (2 puntos), y explique cómo se resuelve cada requerimiento respetando la complejidad en tiempo indicada (1 pto. c/u).

- Para que la inserción y eliminación sea O(logN), debemos utilizar un árbol AVL. Así que los apartamentos pueden estar ordenados en un árbol AVL según su clave de tipo string, de la forma "piso-letra". La inserción o eliminación un árbol AVL es O(logN), por lo que el requerimiento (a) se satisface.
- Para el requerimiento (b), debemos tener por cada apartamento el conjunto ordenado de sus habitantes. Si estos habitantes se disponen en un árbol AVL, el recorrido in-order de sus nodos nos dará sus habitantes de manera ordenada.

- Para el requerimiento (c), para que la inserción/eliminación de un habitante sea O(logN), los habitantes de un apartamento deben estar en un AVL.
- Finalmente, para el requerimiento (d), basta tener un campo dentro de la clase Edificio que nos diga cuántos habitantes están registrados.

En definitiva, la estructura sería la siguiente:

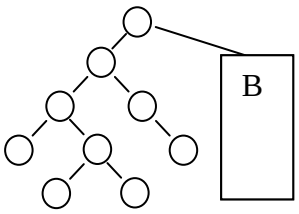


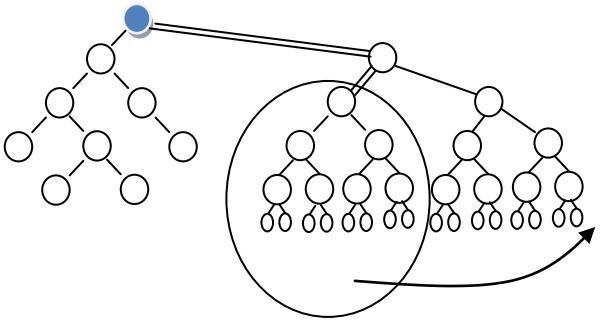

```
Class Edificio
        Integer habitantes;
        Apartamento ^raíz;     // raíz de un AVL
        // …métodos
EndClass

Type Record Apartamento
        String piso_numero;
        Habitante ^raíz_hab;  // raíz de un AVL
        Apartamento ^izq, ^der;
EndClass

Type Record Habitante
        String nombre;
        Habitante ^izq, ^der;
EndRecord
```

4. Dado el siguiente árbol binario, complete el árbol B y colóquele las claves enteras a todos los nodos para que el árbol resultante sea AVL (1 pto.), y luego realice una inserción en dicho árbol tal que requiera de una rotación doble en el nodo raíz (1 pto.). Dibuje el árbol resultante después de la rotación doble, e indique los factores de balance resultantes (1 pto.)





Al insertar en cualquier hoja de este subárbol generaría una rotación doble en la raíz (nodo azul)