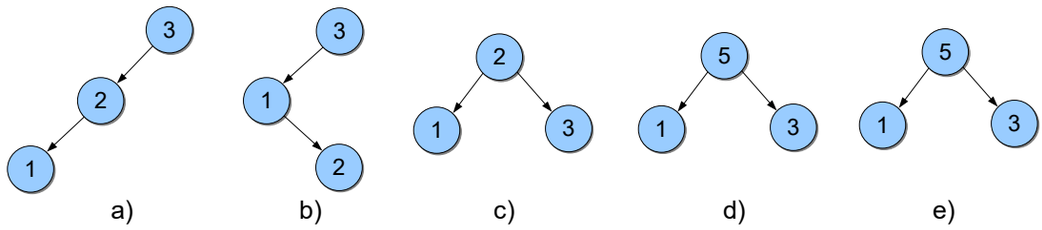
Facultad Politécnica – Ingeniería en Informática
Algoritmos y ED III - Año 2018 – 1er Semestre . Prof. Cristian Cappo
Arboles Binarios de Búsqueda – Tarea #4 - TP: 60

Trabajo a entregar por GRUPO vía EDUCA, hasta el VIERNES 16/marzo/2017 a las 23:55 hs. Cada ejercicio coloque en un subdirectorio separado. Indique claramente los integrantes de su grupo. El código debe poder probarse sin usar ningún IDE desde la linea de comandos. No incluya ninguna clase en package.

Ejercicio 1 (10p)
Resuelva los ejercicios 2 al 6 del ejercitario de la clase de Laboratorio BST dado el jueves 8/Marzo.

Ejercicio 2(10p)
Implemente en BST un **método estático** que dado dos BST retorne un enumerado que indique:

1) Si son iguales en forma y en valores (son idénticos). Asegurar que no sean la misma referencia.
2) Si son diferentes solo en forma (los elementos son iguales pero la ubicación es diferente) pero iguales en valores.
3) Si son diferentes en valores.



Ejemplos: *comparar(a,b)* es el caso 2, *comparar(b,c)* es el caso 2, *comparar(c,d)* es el caso 3, *comparar(c,c)* es un error, *comparar(d,e)* es el caso 1.

Indique el costo asintótico en términos de *O* de su implementación. En este caso se tiene dos tamaños a considerar, el tamaño del primer BST y el tamaño del segundo BST. Por tanto, la respuesta del tiempo debe estar basado en estos dos parámetros.

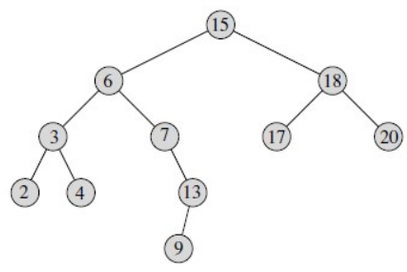
Ejercicio 3 (10p)
Implemente la función de búsqueda en un árbol binario (no precisamente BST) sin hacer uso de recursión ni estructuras auxiliares como pilas y colas. Demuestre el costo asintótico de su algoritmo. Su función recibe como parámetro la clave a buscar, y retorna el dato asociado a la clave buscada (ver abajo, el método *buscar*). Para ello cree un contenedor llamado ***ArbolBinario*** cuya definición se presenta abajo.

La clase `Nodo` es la siguiente:

```
class Nodo<T,D> {
        public K key;
        public D dato,
        public Nodo izq, der;
}
```

```
La clase ArbolBinario se definiría:
class ArbolBinario<K,D> {
        private Nodo<K,D> raiz;
        ArbolBinario() {
            raiz = null;
        }
        void agregar (K clave, D dato ) {
            ...
        }
        D buscar ( K clave ) {
            ...
        }
}
```

Ejercicio 4 (10p)
Amplíe la clase BST entregada por el profesor agregándole los siguientes métodos (asuma que las claves no se repiten). Haga que el BST sea genérico (uso de *Generics*)



a) **`agregar`** agregar un nodo del árbol (NO recursivo) (1p)
b) **`eliminar`** elimina un nodo del árbol (NO recursivo) (1p)
c) **`esLleno`** retorna *True* si es un el árbol lleno (2p)
d) **`esCompleto`** retorna *True* si el árbol esta completo (2p)
e) **`cntHojas`** retorna la cantidad de nodos "hojas" el árbol (2p)
   En el ejemplo del BST de la derecha, se retornaría 5
f) **`Sucesor`** retorna el menor valor siguiente a la clave analizada
   En el ejemplo del BST de la derecha, el sucesor de 3 es 4. El sucesor de 13 es 15.
g) **`Predecesor`** retorna el mayor valor previo a la clave analizada (1p)
   En el ejemplo del BST de la derecha, el predecesor de 13 es 9, el predecesor de 17 es 15. (1p)

Por cada método indique su costo temporal en términos de "O grande". Indique también el *costo espacial* para cada método. Proporcione los casos de prueba para verificar las implementaciones. Puntaje: implementación=75%, costo=25%

Ejercicio 5 (10p)

Nosotros podemos ordenar un arreglo de *n* elementos construyendo un BST conteniendo estos números (utilizando una sucesión de inserciones uno a uno en el árbol) y luego haciendo un recorrido simétrico (in-orden) sobre el mismo.

¿Cuál es el costo asintótico en el mejor y peor caso de este algoritmo de ordenación? Fundamente su respuesta. (3p)

¿Cómo podría evitar o minimizar el peor caso? (2p)

Implemente esta idea en Java y compruebe empíricamente el rendimiento de su algoritmo ordenando arreglos numéricos de tamaños definidos por la linea de comandos. Para ello incluya la tabla de rendimiento mostrada en la clase pasada (5p)

Por ejemplo: `$ java Ordenar_con_BST 1000 2000 3000 4000 5000 60000`

Ejercicio 6 (10p)

a) Demostrar que la longitud del camino interno de un árbol binario, en promedio, esta en *O(n log n)* donde *n* es la cantidad de nodos del árbol. Consulte en [WEISS2000] (cap. 18). (5p)

b) Demuestre formalmente que el algoritmo recursivo que calcula la altura de árbol binario de búsqueda esta en *O(n)*. Utilice como guía la demostración de ***Recorrido-In-Orden(x)*** realizada en clase. (5p)

Referencia

[WEISS2000] Weiss, Mark. Allen, Marroquín, O., Segura, C., & Verdejo, J. A. (2000). *Estructuras de datos en Java: compatible con Java 2*. Pearson-Addison Wesley.