UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos **Trabajo Práctico 6**
Cursada 2014

Árboles Binarios de Búsqueda y AVL

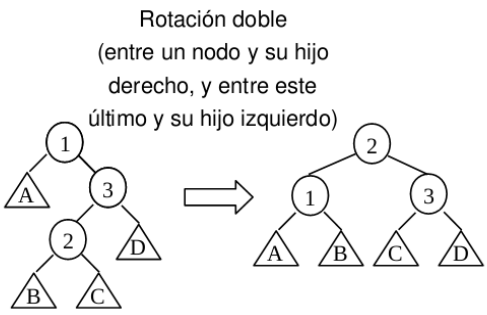
Ejercicio 1

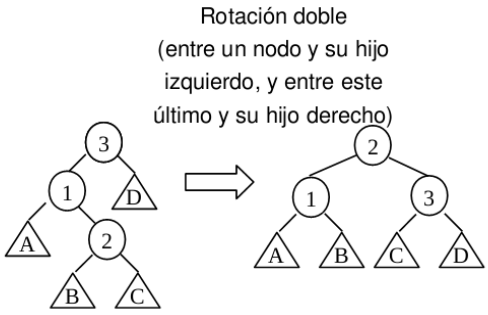
a) Muestre en papel (dibuje) las transformaciones que sufre un árbol binario de búsqueda (inicialmente vacío) al insertar cada uno de los siguientes elementos: 3, 1, 4, 6, 8, 2, 5, 7.
b) Muestre como queda el árbol al eliminar los elementos: 7, 1 y 6.
c) A partir de un árbol binario de búsqueda nuevamente vacío, muestre las transformaciones que sufre al insertar cada uno de los siguientes elementos: 5, 3, 7, 1, 8, 4, 6.
d) Dibuje como queda el árbol al eliminar los elementos: 5, 3 y 7.
e) ¿Qué puede concluir sobre la altura del árbol a partir de a) y c)?

Ejercicio 2

a) Muestre las transformaciones que sufre un árbol AVL (inicialmente vacío) al insertar cada uno de los siguientes elementos: 40, 20, 30, 38, 33, 36, 34, 37. Indique el tipo de rotación empleado en cada balanceo.
b) Dibuje como queda el árbol al eliminar los elementos: 20, 36, 37, 40. Considere que para eliminar un elemento con dos hijos lo reemplaza por el mayor de los más pequeños.

Como ayuda, se muestra a continuación las rotaciones dobles para este tipo de árbol.



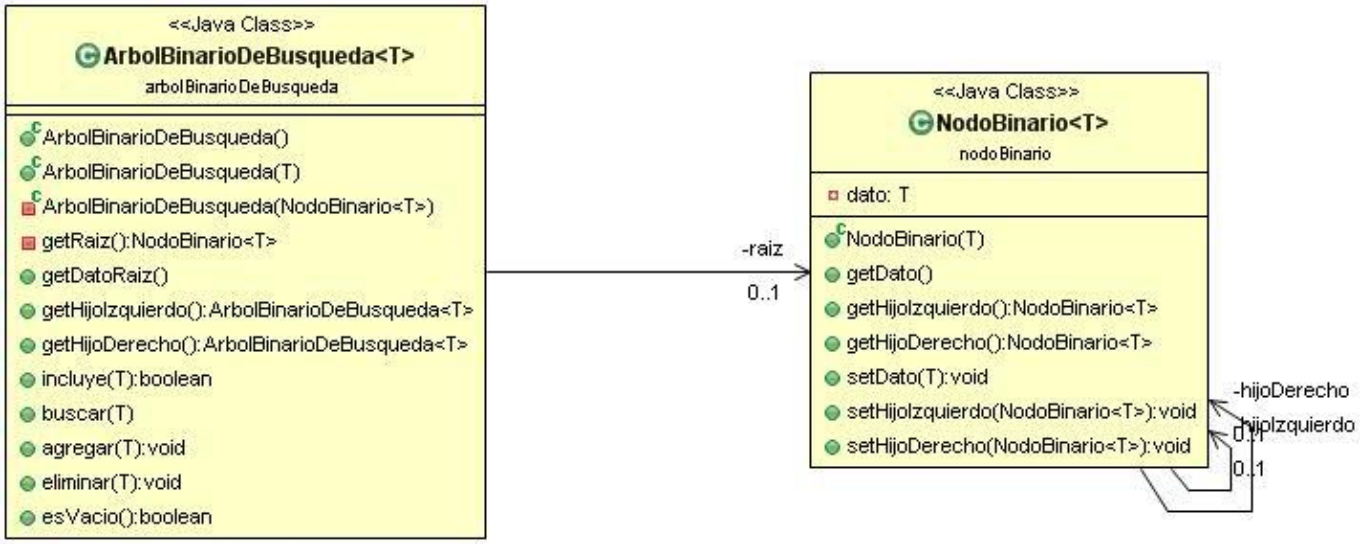


Ejercicio 3

Considere la siguiente especificación de la clase **ArbolBinarioDeBusqueda** (con la representación hijo izquierdo e hijo derecho):

UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos **Trabajo Práctico 6**
Cursada 2014



Aclaración:
En la imagen no aparece la definición de la variable de tipo en la clase ArbolBinarioDeBusqueda. La misma es **<T extends Comparable<T>>**, lo cual indica que la clase representada por la variable T implemente la interface Comparable.
Las clases que implementan la interface *java.lang.Comparable*<*T*> permiten que sus instancias se puedan comparar entre sí. Para lograr esto, deben implementar el método compareTo(T), el cual retorna el resultado de comparar el receptor del mensaje con el parámetro recibido. Este valor se codifica con un entero, el cual presenta la siguiente característica:

= 0: si el objeto receptor es igual al pasado en el argumento.
> 0: si el objeto receptor es mayor que el pasado como parámetro.
< 0: si el objeto receptor es menor que el pasado como parámetro.

La descripción de cada método es la siguiente:

El constructor **ArbolBinariodeBusqueda()** inicializa un árbol binario de búsqueda vacío con la raíz en null.

El constructor **ArbolBinariodeBusqueda(T dato)** inicializa un árbol que tiene como raíz un nodo binario de búsqueda. Este nodo tiene el dato pasado como parámetro y ambos hijos nulos.

El constructor **ArbolBinariodeBusqueda(NodoBinario<T> nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. Este método es privado y se podrá usar en la implementación de las operaciones sobre el árbol.

El método **getRaiz():NodoBinario <T>** retorna el nodo ubicado en la raíz del árbol.

El método **getDatoRaiz():T** retorna el dato almacenado en el NodoBinario raíz del árbol, sólo si el árbol no es vacío.

El método **esVacio(): boolean** indica si el árbol es vacío (no tiene dato cargado).

Los métodos **getHijoIzquierdo():ArbolBinariodeBusqueda<T>** y **getHijoDerecho():ArbolBinariodeBusqueda<T>** retornan los árboles hijos que se ubican a la izquierda y derecha del nodo raíz respectivamente. Están indefinidos para un árbol vacío.

El método **incluye (T dato)** retorna un valor booleano indicando si el dato recibido está incluido en el árbol.

El método **buscar (T dato):T** retorna el valor almacenado en el árbol que es igual al dato recibido.

a) Analice las implementaciones en JAVA de las clases anteriores brindadas por la cátedra y complete los siguientes métodos los cuales les falta su implementación:

El método **agregar (T dato)** agrega el dato indicado al árbol. En caso de encontrar un elemento igual dentro del árbol, reemplaza el existente por el recibido.

El método **eliminar (T dato)** elimina el dato del árbol.

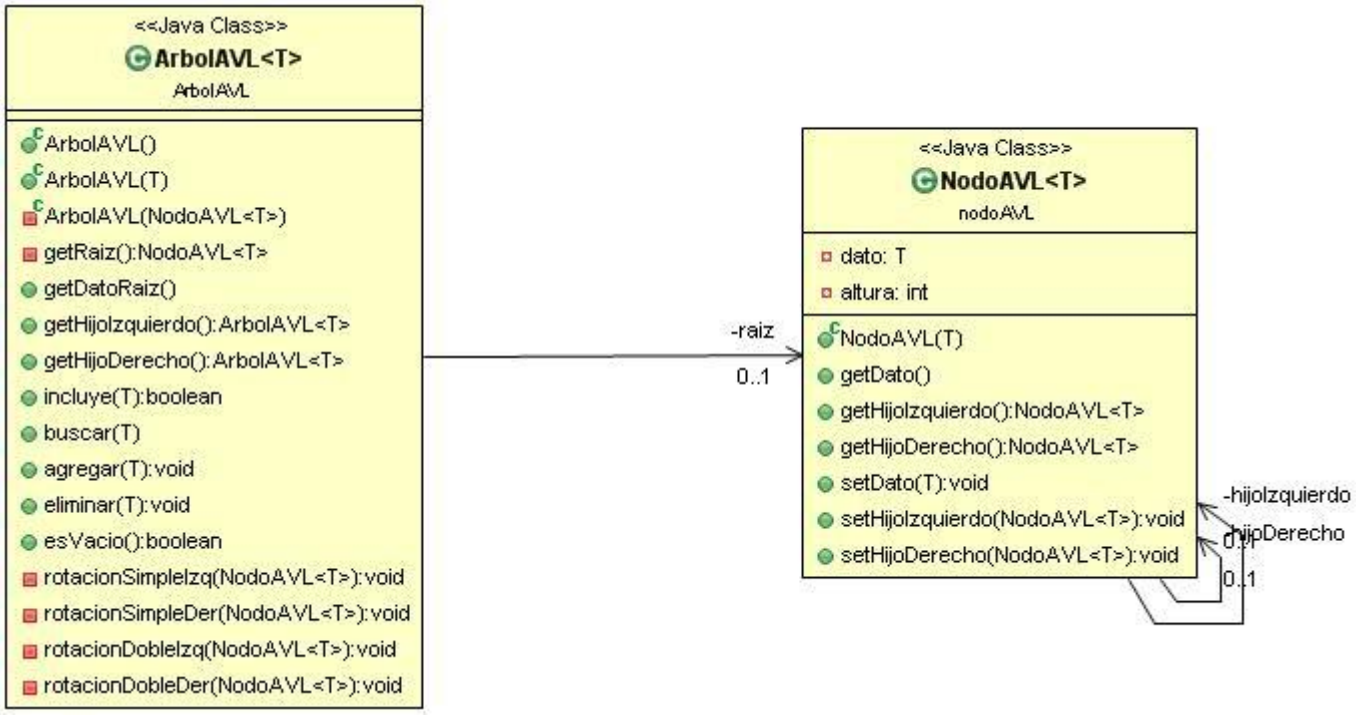
Nota: Tener presente que a diferencia del TP anterior donde se utilizan tipos genéricos, en ABB y AVL no se pueden almacenar cualquier objeto, ya que estos necesitan ser comparables para poder ordenarlos dentro de la estructura.

Ejercicio 4
Implemente la clase **Iterador<T>** que permite recorrer el contenido del árbol en orden. La misma tiene los métodos **hasNext():boolean** y **next():T** que permiten acceder al contenido del árbol siguiendo el orden de los elementos.
A continuación se muestra un método que imprime el contenido de un árbol binario de búsqueda de Strings referenciado por la variable *abb*:

```
Iterador<String> it = new Iterador<String>(abb);
while (it.hasNext()) {
        String valor = it.next();
        System.out.print(valor);
}
```

Ejercicio 5
Verifique las implementaciones realizadas en los ejercicios previos utilizando las pruebas de JUnit provistas.

Ejercicio 6
Considere la siguiente especificación de la clase **ArbolAVL** (con la representación hijo izquierdo e hijo derecho):



La especificación de cada operación es análoga a las del ejercicio anterior, con la salvedad que en este caso se agregar las diferentes rotaciones que incluyen estos árboles.

a) Analice las implementaciones en JAVA de las clases anteriores brindadas por la cátedra y complete los métodos los cuales les falta su implementación.

Ejercicio 7
Agregue a la clase **ArbolAVL** el método **esMinimal():boolean**. Un árbol AVL es minimal de altura h (es decir, si se saca algún nodo deja de ser AVL o de tener altura h) si el número mínimo de nodos queda especificado por la siguiente recurrencia:

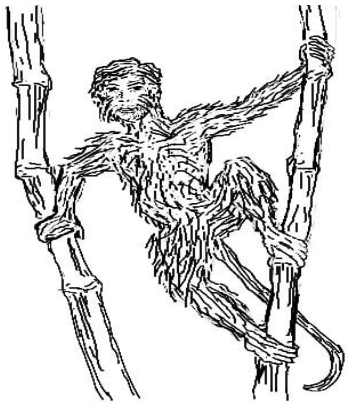
Si necesita, podría definir un método privado, en la clase ArbolAVL, **cantidadDeNodos():int** y otro **min(int h):int**, para resolver **esMinimal():boolean**.

UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos **Trabajo Práctico 6**
Cursada 2014

Anexo Ejercicios Parciales

Ejercicio 1

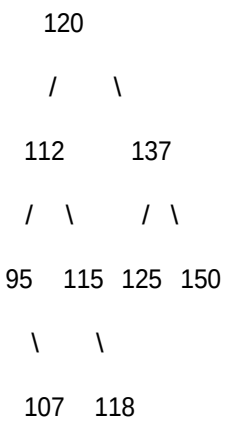


Había una vez un chimpancé llamado *Luchu Bandor,* cuyo significado era "Mono Playboy". *Luchu* estaba infelizmente casado con *Bunty Mona,* una chimpancé muy bonita pero de baja estatura. *Luchu* era alto y guapo, se sentía incómodo cuando estaba con *Bunty* en lugares públicos, ya que la gente los miraba a ellos continuamente. En un momento dado, *Luchu* no pudo soportar más esta situación y decidió hacer justicia a su nombre. Él comenzó a buscar una nueva esposa en el "Colegio Nacional de Señoritas Chimpancés". Cada día *Luchu* se subía a unas cañas de bambú y esperaba a que el ejercicio matutino empezara. Desde allí podía ver a todas las chimpancés haciendo su rutina de ejercicio diario. Ahora, *Luchu* estaba buscando a una chimpancé más alta pero que sea más baja que él, y también estaba interesado en aquella chimpancé un poco más alta que él. Sin embargo, alguien de su misma altura no la consideraba.

Luchu pudo modelar la situación descripta a través de un árbol AVL, el cuál contenía todas las alturas de las señoritas chimpancés que él había observado durante un cierto período de tiempo.

Su trabajo consiste en ayudar a *Luchu* para encontrar a las dos mejores chimpancés de acuerdo al criterio de selección establecido: la chimpancé más alta de las más bajas que él y la más baja entre las más altas que él.
Usted debe implementar un método en la clase árbol AVL, considerando que recibe como parámetro la altura de *Luchu* y debe devolver las alturas de las dos chimpancés buscadas ordenados de manera creciente. En el caso que sea imposible encontrar alguna de estas dos alturas devuelva un valor igual a 0 para la menor y 999 para la mayor. (0 y 999 no son alturas válidas, no están en el árbol)
Importante: considere que en el árbol existe una altura igual a la altura de *Luchu*.

Ejemplo



Las alturas del árbol están en cm.

- Si la altura de *Luchu* fuera 112, debe devolver : 107, 115
- Si la altura de *Luchu* fuera 120, debe devolver : 118, 125
- Si la altura de *Luchu* fuera 95, debe devolver : 0, 107
- Si la altura de *Luchu* fuera 150, debe devolver : 137, 999
- Si la altura de *Luchu* fuera 107, debe devolver : 95, 112
- Si la altura de *Luchu* fuera 118, debe devolver : 115, 120

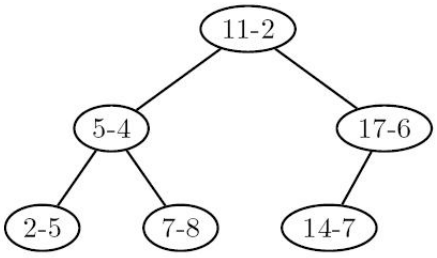
UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos **Trabajo Práctico 6**
Cursada 2014

Ejercicio 2
Un **Treap** es un árbol binario de búsqueda donde los nodos almacenan datos que tienen asociados una prioridad. Cada nodo u del **Treap** obedecen la siguiente propiedad:

- Si v es hijo izquierdo de u, entonces v.dato < u.dato.
- Si v es hijo derecho de u, entonces v.dato > u.dato.
- Si v es hijo de u, entonces v.prioridad > u.prioridad.

Según la definición anterior, la siguiente figura ilustra un ejemplo de un **Treap**.



La inserción en este tipo de estructura se realiza de forma idéntica a la de un árbol binario de búsqueda común y luego se debe verificar si hay que realizar algún tipo de rotación para satisfacer la propiedad antes enunciada.

Inserte par dato-prioridad (3-3) y (18-1) en el **Treap** mostrado en la figura y dibuje el árbol resultante de cada operación.

Ejercicio 3
Considere el siguiente Árbol Binario de Búsqueda ABB de la **Figura 1**:

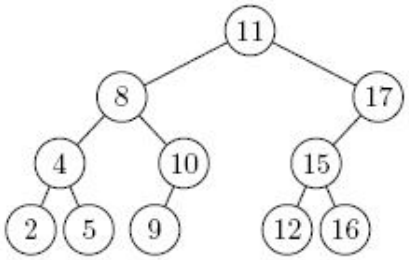


Figura 1: Arbor Binario de Búsqueda ABB.

a) Dibuje el árbol resultante luego de agregar 14 a ABB.
b) Dibuje el árbol resultante luego de eliminar 8 a ABB.
c) Implemente un método que determine si se cumple la **Propiedad 1**, retornando **true** en caso que se cumpla y **false** en caso contrario.

Propiedad 1*:"La propiedad es verdadera si y solo si por cada nodo de un árbol binario de búsqueda la cantidad de nodos del subárbol izquierdo y del subárbol derecho difieren al menos en 1"*