13 | Árboles AVL

Meta

Que el alumno domine el manejo de información almacenada en una *Árbol binario ordenado balanceado*.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Visualizar cómo se almacenan los datos en una estructura no lineal.
- Analizar la eficiencia de un árbol autobalanceado.
- Dominar el uso de referencias para conectar los nodos de una estructura de datos.
- Experimentar el uso de la herencia de orientación a objetos para reutilizar algoritmos, refactorizando el código de la práctica anterior según se requiera.

Antecedentes

Definición 13.1

Un **Árbol AVL** es un árbol binario tal que, para cada nodo, el valor absoluto de la diferencia entre las alturas de los árboles izquierdo y derecho es a lo más uno. En otras palabras:

1. Un árbol vacío es un árbol AVL.
2. Si T es un árbol no vacío y T_i y T_d sus subárboles, entonces T es AVL si y sólo si:
   a) T_i es AVL.
   b) T_d es AVL.

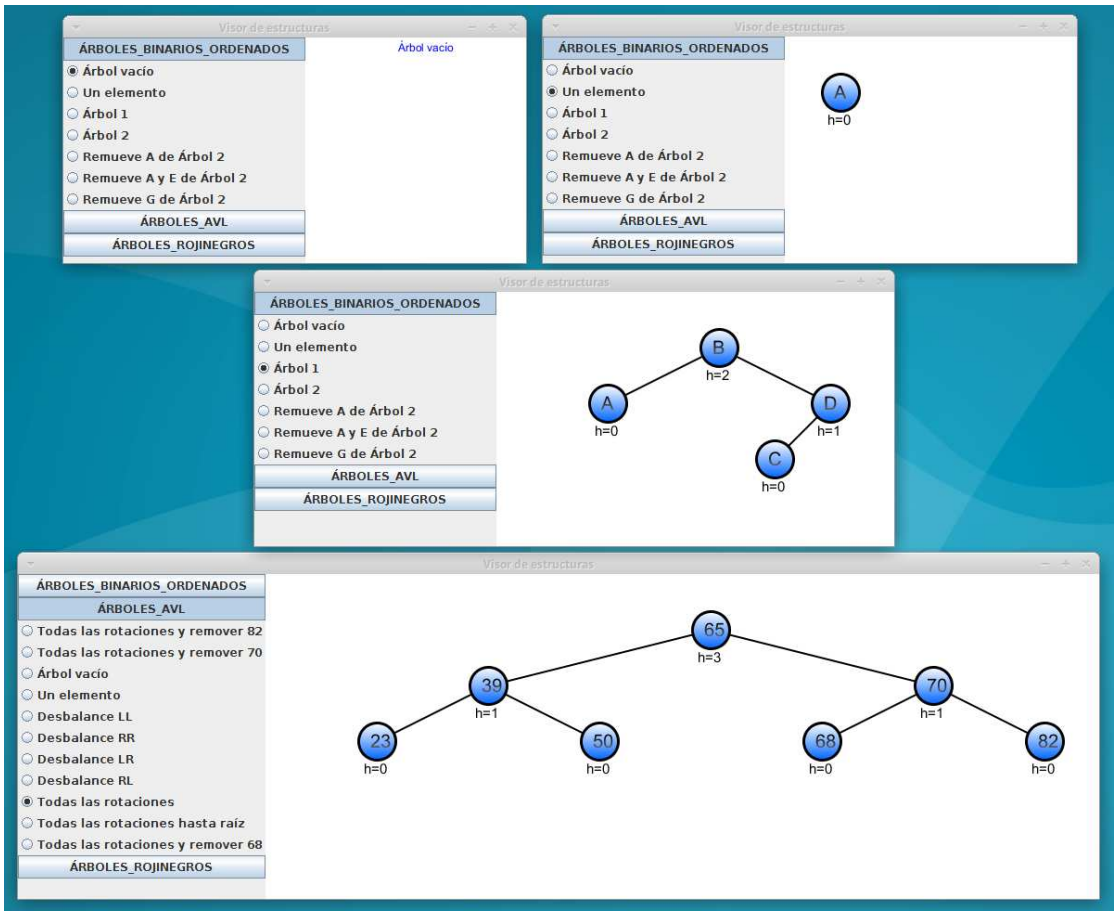

Figura 13.1 Muestra de cómo se debe ver el visor si los árboles están programados correctamente.

c) $|\text{altura}(T_i) - \text{altura}(T_d)| <= 1$

Los árboles AVL toman su nombre de las iniciales de los apellidos de sus inventores, Georgii Adelson-Velskii y Yevgeniy Landis.

Desarrollo

Para ver los árboles de manera gráfica, se provee un paquete que facilita mostrarlos. Como ejemplo vemos el siguiente código:

Para implementar un **Árbol AVL** se deben implementar las siguientes clases:

- `NodoAVL`. Esta clase debe implementar la interfaz `NodoBinario`, hereda de `NodoBOLigado`.

- `ÁrbolAVL`. Esta clase extiende `ÁrbolBOLigado`. Obseva que no se define una interfaz nueva, pues `ÁrbolAVL` no define un tipo de dato abstracto, por el contrario es una forma [eficiente] de implementar un árbol binario ordenado.

1. En `NodoAVL` sobreescribe el método `getAltura()` de forma que tome tiempo $\mathcal{O}(1)$. Para lograr esto necesitarás un atributo en la clase, la cual tendrás que ir manteniendo cada que se haga un rebalanceo.

   Una sugerencia para que esto no afecte la visualización de tus árboles es: primero agregar la variable e ir actualizando su valor en los métodos correspondientes al rebalanceo y, hasta después de haber programado el resto de los métodos y de verificar que funcionan, sobrecargar `getAltura()` y revisar que las alturas se estén manteniendo de manera correcta.

2. Programa en `NodoAVL` los métodos para:

   a) Realizar rotaciones izquierda y derecha sobre el nodo.

   b) Para balancear un nodo, dado que su factor de balanceo es 2 ó -2.

3. En `ÁrbolAVL` sobreescribe los métodos para agregar y eliminar de tal modo que se agreguen los pasos para balancear el árbol. Observa que no es necesario eliminar el código que ya tenías para agregar y remover los nodos. Sólo falta recorrer el árbol desde el nodo modificado, hacia arriba, actualizando alturas y revisando los factores de balanceo. Recuerda que, si al balancear cambia la raíz del árbol, debes actualizar el atributo correspondiente en la clase `ÁrbolAVL`.

4. Obseva que el iterador de la práctica pasada sigue funcionando.

Preguntas

1. Explique cómo se sabe si se hace una rotación izquierda, derecha o una doble rotación, para balancear el árbol.