

Práctica 8

Árboles AVL

Estructuras de datos

INSTRUCCIONES ESPECIALES

Lee las instrucciones de la práctica sobre árboles binarios ordenados, pero no programes las clases que se te piden aquí. Programa directamente `ÁrbolAVL` y `NodoAVL`. Estas dos clases deberán implementar las interfaces `ÁrbolBinarioOrdenado<C>` y `NodoBinario<E>`. Recuerda que tus estructuras deberán requerir `<C extends Comparable<C> >` en la declaración de la clase.

Antecedentes: Árboles Binarios Ordenados

META

Que el alumno domine el manejo de información almacenada en un *árbol binario ordenado*.

OBJETIVOS

Al finalizar la práctica el alumno será capaz de:

- Visualizar el uso correcto de referencias para implementar estructuras tipo árbol.
- Manejar con familiaridad el almacenamiento de datos en una estructura utilizando referencias.
- Implementar con mayor habilidad métodos recursivos y/o iterativos para manipular estructuras de datos con referencias.

ANTECEDENTES

Definición 1

Un árbol es una colección de elementos llamados *nodos*, uno de los cuales se distingue como *raíz*, junto con una relación de «paternidad» que impone una estructura jerárquica sobre los nodos Vargas 1998.

Formalmente:

1. Un solo nodo es, por sí mismo, un árbol. Ese nodo es también la raíz de dicho árbol.

2. Supóngase que n es un nodo y que A_1, A_2, \dots, A_k son árboles con raíces n_1, n_2, \dots, n_k , respectivamente. Se puede construir un árbol nuevo haciendo que n se constituya en el padre de los nodos n_1, n_2, \dots, n_k . En dicho árbol, n es la raíz y A_1, A_2, \dots, A_k son los *subárboles* de la raíz. Los nodos n_1, n_2, \dots, n_k reciben el nombre de *hijos* del nodo n .

Definición 2

Un **árbol binario** se puede definir de manera recursiva:

1. Un **árbol binario** es un árbol vacío.
2. Un nodo que tiene un elemento y dos **árboles binarios**: izquierdo y derecho.

Definición 3

Un *árbol binario ordenado* contiene elementos de un tipo C tal que todos ellos son comparables mediante una relación de orden. En un árbol ordenado cada nodo cumple con la propiedad siguiente:

1. Todo dato almacenado a la *izquierda* de la raíz es *menor* que el dato en la raíz.
2. Todo dato almacenado a la *derecha* de la raíz es *mayor o igual* que el dato en la raíz.

DESARROLLO

En esta práctica implementarás no sólo un árbol binario ordenado, sino uno que servirá como clase base para los árboles balanceados de las prácticas siguientes. Por ello algunas de las decisiones de diseño que se incorporarán en esta práctica podrían parecer extrañas, pero sus beneficios saldrán a la luz más adelante.

Como un auxiliar para esta práctica, se provee una interfaz gráfica que dibuja los árboles si implementan correctamente las interfaces en el código adjunto. Esta interfaz es un paquete que funciona de forma muy semejante a `junit`, por lo que, si deseas agregar pruebas nuevas para tus árboles puedes hacerlo. Observa los ejemplos en el archivo `ed/visualización/demos/DemoÁrbolesBinariosOrdenados.java`.

Para ver los árboles de manera gráfica, se provee un paquete que los dibuja en pantalla, con la condición de que se encuentren en un estado consistente. El código siguiente muestra el uso del decorador `@DemoMethod` para graficar los árboles.

Listing 1: Extracto de `DemoÁrbolesBinariosOrdenados.java`

```
1 package ed.visualización.demos;
2
3 import ed.estructuras.nolineales.ÁrbolBOLigado;
4 import ed.visualización.dibujantes.DibujanteDeÁrbolBinarioOrdenado;
5 public class DemoÁrbolesBinariosOrdenados extends Demo {
6     private DibujanteDeÁrbolBinarioOrdenado dibujante;
7
8     @DemoMethod(name = "Árbol_vacío")
9     public String dibujaÁrbol0() {
10         ÁrbolBOLigado árbol;
11         árbol = new ÁrbolBOLigado();
```

```

12     dibujante = new DibujanteDeÁrbolBinarioOrdenado();
13     dibujante.setEstructura(árbol);
14     return dibujante.drawSVG();
15 }
16 }

```

Para implementar este **árbol binario ordenado** se deben programar las siguientes clases:

- **NodoBOLigado**. Esta clase debe implementar la interfaz `NodoBinario<C>` y sus elementos son del tipo genérico `<C extends Comparable<C>>`. `NodoBinario<E>` extiende `Nodo<E>` por lo que tendrás que implementar todos los métodos que requieren estas interfaces.
- **ÁrbolBOLigado**. Esta clase debe implementar la interfaz `ÁrbolBinarioOrdenado<C>`; contendrá todo el código aplicable a cualquier árbol binario ordenado. No tiene que estar balanceado. Obsérvese que `ÁrbolBinarioOrdenado< C extends Comparable <C> >` extiende `ÁrbolBinario<E>`, que a su vez extiende `Árbol<E>`. Por lo tanto tu clase `ÁrbolBOLigado < C extends Comparable<C> >` debe implementar todos los métodos definidos por las tres interfaces. `Árbol<E>` también extiende a `Collection<E>`, por ello algunos métodos ya se encuentran implementados en la clase `ColeccionAbstracta<E>`, te conviene extenderla.

1. Comienza por programar la clase `NodoBOLigado` y asegúrate de que compile bien.
2. Continúa con la clase `ÁrbolBOLigado`. Agrégale un método:

```

1     protected NodoBinario<C> creaNodo(C dato,
2                                     NodoBinario<C> padre,
3                                     NodoBinario<C> hijoI,
4                                     NodoBinario<C> hijoD) {
5         ...
6     }

```

Este método creará a los nodos del tipo correspondiente y en futuras prácticas será sobrescrito por clases herederas de ésta. Recuerda crear a todos tus nodos con este método en lugar de con su constructor.

3. Antes de programar al método `add`, crea un método auxiliar que hará el grueso del trabajo. Este método es:

```

1     protected NodoBinario<C> addNode(C e) {
2         ...
3     }

```

Este método debe devolver al nodo recién agregado. Los detalles los puedes implementar en el árbol, si tu implementación es iterativa (que es más eficiente), o en el nodo si optas por la versión recursiva. Úsalo después para implementar las otras versiones de agregar que se te solicitan.

4. Crea otro método auxiliar que te permita encontrar al primer nodo que contenga al dato pasado como parámetro y lo devuelva.
5. Para remover nodos también debes usar un método auxiliar. Aquí utilizaremos un truco curioso para el futuro: este método debe devolver el nodo que se quite del árbol. El último padre de este nodo debe eliminar su referencia hacia él, pero este nodo debe quedarse con la referencia a quien fue su padre. De cualquier modo cuando dejemos de usar este nodo esa referencia desaparecerá, pero lo necesitaremos un poco más para balancear árboles en clases futuras.

Indicaciones adicionales:

1. Observa también que `ÁrbolBinario<E>` solicita un método que devuelve el nodo raíz, éste fue necesario para que el paquete de dibujo pudiera realizar su tarea en forma eficiente. El paquete de dibujo necesita acceso a la estructura del árbol pues eso es lo que va a dibujar. Es un caso de uso distinto al de un programador que sólo quiere al árbol para que almacene sus datos en orden y se los devuelve eficientemente.
2. Los métodos `contains`, `remove` y `add` deben cumplir con la complejidad de $O(\log(n))$.
3. Para el método `iterator` utiliza el recorrido inorden, no implementes ni `add` ni `remove`.

PREGUNTAS

1. Si se añaden los números del 1 al 10 en orden y luego se pregunta si el 10 está en el árbol ¿cuál es la complejidad?
2. Si se añaden los números en un orden aleatorio ¿cuál es la complejidad promedio de preguntar por el 10?

Práctica: 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 4

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.
- c) $|altura(T_i) - altura(T_d)| \leq 1$

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:

Listing 2: Extracto de DemoArbolesAVL.java

```

1 package ed.visualización.demos;
2
3 import ed.estructuras.nolineales.ÁrbolAVL;
4 import ed.visualización.dibujantes.DibujanteDeÁrbolBinarioOrdenado;
5 public class DemoArbolesAVL extends Demo {
6     private DibujanteDeÁrbolBinarioOrdenado dibujante;
7     @DemoMethod(name = "DesbalanceLL")
8     public String dibujaLL() {
9         ÁrbolAVL<String> árbol;
10        árbol = new ÁrbolAVL<>();
11        dibujante = new DibujanteDeÁrbolBinarioOrdenado();
12        dibujante.setEstructura(arbol);
13        árbol.add("C");
14        árbol.add("B");
15        árbol.add("A");
16        return dibujante.drawSVG();
17    }
18 }
```

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. Programa en **NodoAVL** los métodos para realizar rotaciones izquierda y derecha sobre el nodo y para balancearlo, dado que su factor de balanceo es 2 ó -2.
 2. **ÁrbolAVL** sobrescribe los métodos para agregar y eliminar de tal modo que se agreguen los pasos para balancear el árbol. Observe 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**.
 3. 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.

FORMA DE ENTREGA

- Asegúrense de borrar todos los archivos generados, incluyendo archivos de respaldo usando `ant clean`.
- Copien el directorio `Practica8` dentro de un directorio llamado `<apellido1_apellido2>` donde `apellido1` es el primer apellido de un miembro del equipo y `apellido2` es el primer apellido del otro miembro. Por ejemplo: `marquez_vazquez`.
- Borren el directorio `libs` en la copia.
- Agreguen un archivo `reporte.pdf` dentro del directorio `Practica8` de la copia, con el nombre completo de los integrantes del equipo, las repuestas a las preguntas de la sección anterior y cualquier comentario que quieran hacer sobre la práctica.
- Compriman el directorio `<apellido1_apellido2>` creando

`<apellido1_apellido2>.tar.gz`.

Por ejemplo: `marquez_vazquez.tar.gz`.

- Suban este archivo en la sección correspondiente del aula virtual. Basta una entrega por equipo.

FORMA DE EVALUACIÓN

Para su calificación final se tomarán en cuenta los aspectos siguientes:

- 70 % Calificación generada por las pruebas automáticas. Usaremos nuestros archivos, por lo que si realizan modificaciones a sus copias, éstas no tendrán efecto al momento de calificar.
- 10 % Reporte con respuestas a las preguntas.
- 10 % Documentación completa y adecuada. Entrega en el formato requerido, sin archivos `.class`, respaldos, bibliotecas de `JUnit` u otros no requeridos.
- 10 % Revisión manual del código, para verificar que se cumpla con las especificaciones, que no se haya copiado, etcétera.

REFERENCIAS

Vargas (1998). *Estructuras de datos y Algoritmos*. Trad. por Jorge Lozano Moreno (colaboración de Guillermo Levine Gutiérrez) América Vargas Villazón. 438 págs.