12 | Árbol binario ordenado

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 12.1

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 $A_1, A_2, ..., A_k$ son árboles con raíces

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

Definición 12.2

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 12.3

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.

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 balancedo. 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:

```
protected NodoBinario<C> creaNodo(C dato,
                                  NodoBinario<C> padre,
                                  NodoBinario<C> hijoI,
                                  NodoBinario<C> hijoD) {
  ...
}
```

   Este método creará a los nodos del tipo correspondiente y en futuras prácticas será sobreescrito 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:

```
protected NodoBinario<C> addNode(C e) {
  ...
}
```

   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 están 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?