Laboratorio A.E.D.

Lars-Åke Fredlund
lfredlund@fi.upm.es
Tonghong Li
tonghong@fi.upm.es
Manuel Carro Liñares
mcarro@fi.upm.es
Germán Puebla Sánchez
german@fi.upm.es
Pablo Nogueira
pnogueira@fi.upm.es

Viernes 11:00-13:00

Entrega

- La fecha límite para optar a la máxima nota es
  Viernes 7 de diciembre de 2012, a las 13:00 horas
- La nota no baja el Lunes 10 de diciembre, debido al cierre de la facultad.
- El fichero que hay que subir es `BinTreeWithDelete.java`
- La entrega se hace a través de la siguiente URL:
  `http://lml.ls.fi.upm.es/~entrega`
- El paquete `binaryTreesWithDelete` esta documentado con Javadoc en
  `http://babel.ls.fi.upm.es/~fred/courses/aed/binaryTreesWithDelete/`
- El proyecto debe compilar sin errores, cumplir la especificación y pasar el `Tester`.

Configuración

- Arrancad Eclipse.
- Cread un paquete `binaryTreesWithDelete` en el proyecto `aed`, dentro de `src`.
- Aula Virtual → AED → Sesiones de laboratorio → Laboratorio 8 → codigo_lab8.zip (formato zip).
- Importad al paquete `binaryTreesWithDelete` los fuentes que habéis descargado
- Ejecutad `Tester`. Veréis que lanza una excepción:

```
Testing findAll...

findAll({10,*}) returned null?
Exception in thread "main" java.lang.Error
at binaryTreesWithDelete.Tester.check_exists(Tester.java:212)
at binaryTreesWithDelete.Tester.doTest(Tester.java:60)
at binaryTreesWithDelete.Tester.main(Tester.java:38)
```

Tareas para hoy

Hoy trabajaremos otra vez con arboles binarios de búsqueda. La clase `BinTreeWithDelete<E>` extiende `LinkedBinaryTree<E>` con dos métodos que debéis completar:

- `PositionList<E> findAll(E element)` devuelve una lista con los elementos del árbol que son iguales a `element`.
- `void delete(Position<E> pos)` borra el elemento del nodo `pos`, y deja el arbol con un nodo menos.

Un árbol binario de búsqueda es *ordenado*: el elemento de un nodo es mayor que todos los elementos de su subárbol izquierdo, y menor o igual que todos los elementos de su subárbol derecho.

Restricciones y permisos

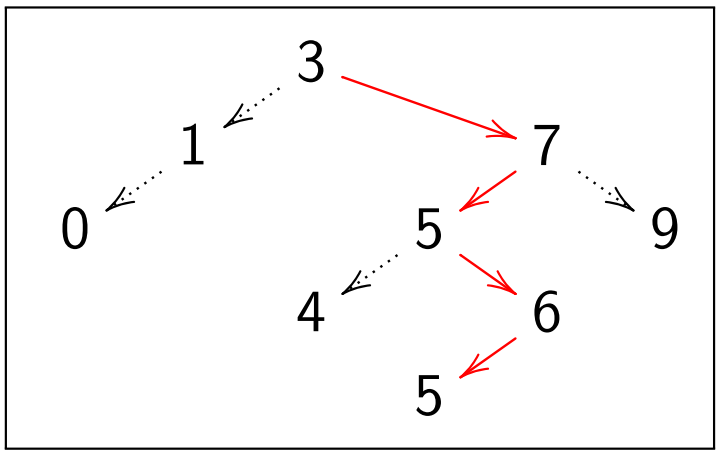
- Esta permitido añadir nuevos métodos, atributos privados, o variables locales.
- Sólo esta permitido usar los siguientes métodos de la clase `LinkedBinaryTree<E>`:

  `addRoot`, `insertLeft`, `insertRight`, `hasLeft`, `hasRight`, `isEmpty`, `isExternal`, `isInternal`, `isRoot`, `left`, `right`, `root`, `parent`, `sibling`, `size`, `remove`, `replace`

  El uso de otros métodos de la clase queda prohibido.
- A los objetos de tipo `Position` sólo esta permitido llamar a `element()`.

PositionList<E> findAll(E element)

- El método debe devolver una lista con todos los elementos del árbol iguales a `element`.
- Para comparar dos elementos debe usarse el comparador `cmp` que es un atributo publico en la clase `BinTreeWithDelete`.
- Para obtener una puntuación mejor el método debe ser *eficiente*. Usad el hecho de que el árbol está ordenado para limitar la parte del árbol que se recorre.
- Ejemplo: dado el árbol



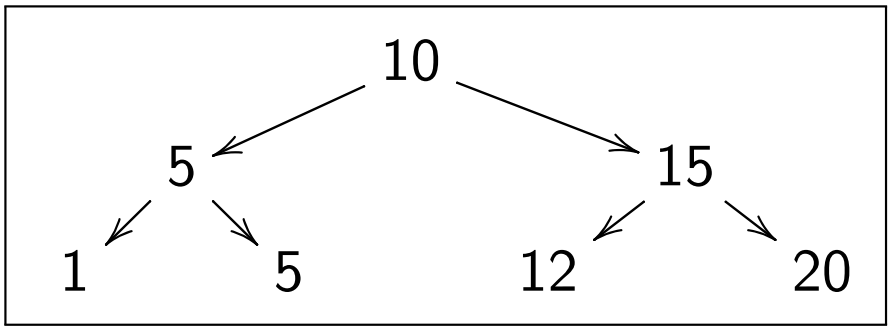
`findAll(5)` devuelve la lista 5,5 y el camino óptimo se indica usando el color rojo.

`void delete(Position<E> pos)`

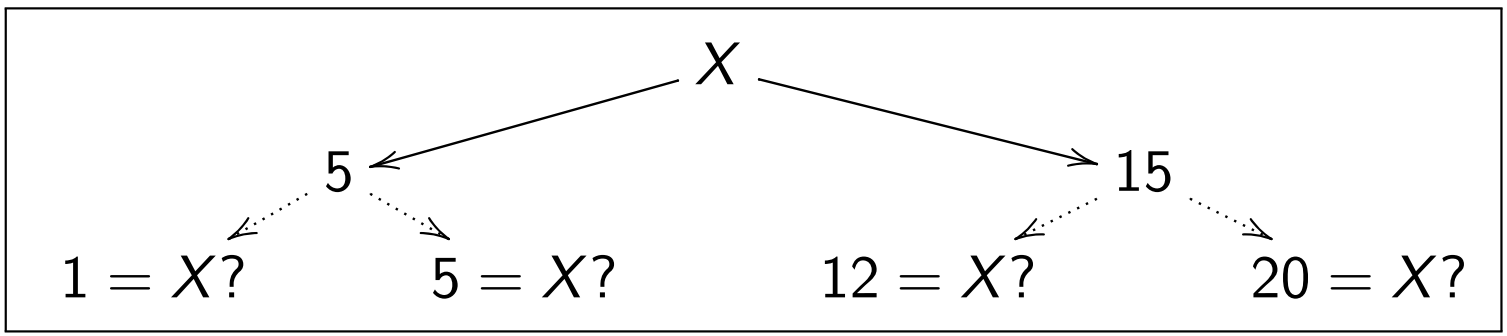
- El método debe borrar el elemento en el nodo `pos`, y dejar el árbol con un nodo menos, pero el árbol debe seguir *ordenado*.
- Observad que el método `remove(Position<E> pos)` de la clase `LinkedBinaryTree`, que podeis utilizar, sólo puede borrar un nodo que tiene uno o cero hijos.
- Para borrar un nodo con dos hijos debe buscarse otro nodo que es fácil de borrar (p.ej.,no tiene dos hijos), y cuyo elemento se puede mover al nodo `pos` de forma que el árbol siga ordenado.

`void delete(Position<E> pos)`: ejemplo

Dado el árbol



- Queremos borrar el elemento 10
- Buscamos otro nodo X, borramos este nodo, y reemplazamos 10 con el elemento del otro nodo.



- **Pero**, el árbol tiene que ser *ordenado*.