Estructuras arborescentes: árboles binarios

- Algunos de los ejercicios contenidos en este documento se han de resolver en el Jutge (en la lista correspondiente del curso actual); aquí están señalados con la palabra *Jutge*.
- En general, los ejercicios contenidos en este documento se presentan por orden de dificultad. Por ello, recomendamos resolverlos en el orden en el que aparecen. No se supervisarán los problemas del Jutge si antes no se han resuelto los ejercicios previos.

Continuamos con los ejercicios de uso de las clases especiales vistas en clase de teoría, en este caso los árboles binarios. Los árboles están implementados en la clase `BinTree`, que no es estándar. Su fichero `BinTree.hh` está en `$INCLUSIONS`. Si queréis solamente consultar la especificación de las operaciones, tenéis el fichero `BinTree.pdf` en la carpeta de la sesión.

Si observáis el código de la clase `BinTree`, veréis que se trata de una clase genérica, gracias al uso de `template`. Este mecanismo impide la compilación separada, pues el compilador de C++ no permite compilar código que use clases "templatizadas" sin ver las instanciaciones de las mismas. Por tanto la habitual división entre los ficheros `.hh` (para incluir) y `.cc` (para compilar) que hemos visto en otras clases no es útil. Por este motivo todo el código de la clase se encuentra en `BinTree.hh`. Este mecanismo es excepcional y su uso fuera del contexto de los `template` se considera inadecuado.

En la carpeta de la sesión se muestran ejemplos resueltos con árboles de enteros. Además se incluyen sendos ficheros `BinTreeIOint.hh` y `BinTreeIOint.cc`, con operaciones de lectura y escritura para árboles de enteros. Notad que dichas operaciones no pueden ser genéricas, pues dependen directamente del tipo del contenido de las estructuras. También hay ejemplos de como usar `tar` para hacer entregas en el *jutge* y como codificar ficheros `Makefile`.

Para probar la clase `BinTree` se pueden utilizar los mismos ejercicios que vimos para `list`: búsqueda en un árbol de pares de enteros o de estudiantes y modificar los elementos de un árbol de pares de enteros o estudiantes.

1.1. Ejercicios

En los apuntes de teoría tenéis una serie de ejemplos sobre `BinTree<int>`. Queremos obtener programas similares para árboles de pares de enteros y de estudiantes.

Cada ejercicio que aparece a continuación requiere programar una o más operaciones de árboles y un método `main` para probarlas. Podéis inspiraros en los materiales que aparecen en la carpeta de la sesión.

1.1.1. Ejemplo: Búsqueda en un árbol de pares de enteros (X20646) de la Lista *List & BinTree (Jutge)*

Probad varias situaciones distintas de búsquedas (que el elemento buscado sea la raíz, que sea una hoja, que no esté, etc.)

Además de la versión para el jutge, os proponemos más versiones:

Obtened una segunda versión en la que los árboles puedan tener elementos repetidos. En este caso, si el elemento buscado aparece varias veces, se ha de retornar la profundidad mínima, es decir, la de la aparición más próxima a la raíz. En caso de haber varias apariciones de profundidad mínima, se ha de retornar el compañero de la aparición de más a la izquierda.

Obtened una tercera versión en la que si se encuentra el elemento buscado, en lugar de obtener su profundidad mínima y su compañero, se modifique el árbol original de forma que éste se convierta en el subárbol cuya raíz sea la aparición de profundidad mínima del elemento buscado. Si el elemento buscado no está, el árbol ha de quedar vacío.

1.1.2. Ejercicio: búsqueda en un árbol de estudiantes (X65608) de la Lista *List & BinTree (Jutge)*

Este ejercicio se puede plantear en varias versiones, de dificultad creciente. El jutge solo contempla una de ellas, concretamente la tercera. Si queréis podéis resolverlas en orden.

Primera versión: dado un número entero y un árbol de estudiantes, sin DNIs repetidos, buscar si hay algun estudiante en el árbol que lo tenga como DNI. En caso de éxito hay que informar sobre la nota del estudiante. El resultado puede ser uno de éstos

```
El estudiante no está en el árbol

El estudiante está en el árbol, pero no tiene nota

El estudiante está en el árbol y su nota es <la que sea>
```

Segunda versión: además del resultado de la búsqueda anterior, se ha de retornar la profundidad a la que se encuentra el elemento encontrado, en su caso. La profundidad de un elemento es su distancia a la raíz. Suponed que la profundidad de la raíz de un árbol es cero.

Tercera versión: considerad que los árboles pueden tener elementos repetidos. En ese caso, si el elemento buscado aparece varias veces, se ha de retornar la profundidad mínima, es decir, la de

la aparición más próxima a la raíz. En caso de haber varias apariciones de profundidad mínima, se ha de retornar la información de la nota de la aparición de más a la izquierda.

Cuarta versión: si se encuentra el elemento buscado, en lugar de obtener su profundidad mínima y la información de su nota, se ha de modificar el árbol original de forma que éste se convierta en el subárbol cuya raíz sea la aparición de profundidad mínima del elemento buscado. Si el elemento buscado no está, el árbol ha de quedar vacío.

1.1.3. Ejercicio: modificar los elementos de un árbol

Para los siguientes ejercicios, producid dos soluciones, una en la que se obtenga un árbol nuevo a partir del original y otra en la que las modificaciones se apliquen directamente sobre el original.

1. Con la misma estructura del ejercicio anterior, escribid un programa que dado un árbol de pares de enteros le sume un valor *k* al segundo elemento de cada par.

2. Aplicad las ideas del problema anterior para redondear las notas de un árbol de estudiantes.

1.1.4. Ejercicio: podar un árbol (X50235) de la Lista *List & BinTree (Jutge)*

Dado un árbol binario de enteros a sin elementos repetidos y un entero x, eliminar de a el valor x y todos sus sucesores.