C06: Árboles genealógicos

Estructuras de Datos
Facultad de Informática - UCM

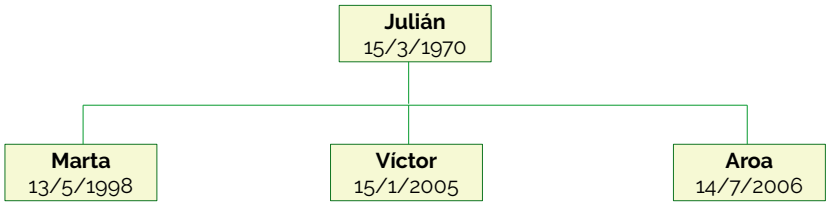
Este ejercicio consta de una única entrega, que debe enviarse al problema *DOMjudge* con identificador `C06` antes del **viernes 13 de marzo a las 23:55**.

La entrega consiste en un único fichero `.cpp` que se subirá a *DOMjudge*. Puedes subir tantos intentos como quieras. Se tendrá en cuenta el último intento con el veredicto `CORRECT` que se haya realizado antes de la hora de entrega por parte de alguno de los miembros del grupo.

No olvides poner el nombre de los componentes del grupo en el fichero `.cpp`. Solo es necesario que uno de los componentes del grupo realice la entrega.

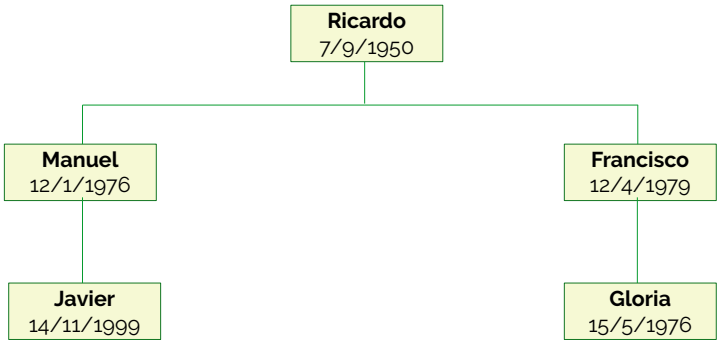
Evaluación: Este ejercicio se puntuará de 0 a 10. Para poder obtener una calificación superior a 0 es necesario obtener un veredicto `CORRECT`.

Un árbol genealógico es un diagrama en el que se representan los descendientes de un individuo:



Normalmente, los árboles genealógicos no solo representan los descendientes naturales de una persona, sino también sus cónyuges. No obstante, por simplificar, vamos a representar únicamente los descendientes naturales. En el ejemplo anterior tenemos, en la raíz del árbol, a Julián, que tiene tres hijos/as: Marta, Víctor, y Aroa.

Ahora consideremos el siguiente árbol:



¿Observas algo raro? El autor de este árbol debe de haberse confundido al transcribir las fechas, porque es imposible que Gloria haya podido nacer antes que Francisco, su padre. Decimos que un árbol es *absurdo* si la fecha de nacimiento de uno de los descendientes de una persona es menor o igual que la fecha de nacimiento de su progenitor.

El objetivo de este ejercicio es realizar un programa que lea de la entrada varios casos de prueba, cada uno de ellos representando un árbol genealógico. El programa debe determinar, para cada caso de prueba, si el árbol leído es absurdo o no. Para ello sigue los siguientes pasos:

1. Crea una clase `Date` para representar fechas. Debe contener tres atributos privados: `day`, `month` y `year`. Implementa, además:
   - Un constructor por defecto, que inicialice los tres atributos a una fecha arbitraria.
   - Un constructor que reciba tres parámetros (día, mes y año) e inicialice los correspondientes atributos.
   - Métodos que sobrecarguen los operadores < y <=.
   - Un método `void read(std::istream &in)` que lea una fecha del flujo de entrada `in` pasado como parámetro. Puedes suponer que la fecha a leer está en el formato `dia/mes/año`. Este método debe asignar los valores leídos de la entrada a los atributos de la clase.
2. Escribe una clase `FamilyTree` para representar árboles genealógicos:

   ```
   class FamilyTree {
   public:
     FamilyTree();
     ~FamilyTree();

     void read(std::istream &in);
     bool nonsense() const;

   private:
     struct FamilyTreeNode {
       Date date;
       std::vector<FamilyTreeNode *> children;
     };

     FamilyTreeNode *root;

     // métodos auxiliares que necesites
   };
   ```

   Un árbol genealógico está formado por varios nodos (`FamilyTreeNode`), siendo uno de ellos el raíz (`root`). Cada nodo representa a una persona. El campo `date` contiene la fecha de nacimiento de la persona y el campo `children` contiene una lista de punteros a los nodos hijo. En caso de no tener descendientes, esta lista será vacía.

   Implementa los siguientes métodos de la clase:

   - Un constructor por defecto, que inicializa `root` a `nullptr`.
   - Un destructor, que libera la memoria ocupada por todos los nodos del árbol.
   - Un método `read()` que lee un árbol del flujo de entrada pasado como parámetro. El formato de la entrada se describe en la sección *Entrada* de este enunciado.
   - Un método `nonsense()` que determina si él árbol genealógico es absurdo.

   Puedes utilizar los métodos privados auxiliares que consideres necesarios.
3. Determina el coste en tiempo del método `nonsense()` en función del número de nodos del árbol.

4. Escribe un programa que lea una serie de árboles genealógicos y determine, para cada uno de ellos, si el árbol es absurdo o no. En las secciones siguientes se describen el formato de entrada y de salida.
5. Sube el programa al problema *DOMjudge* con identificador `C06`.

Entrada

La entrada contiene una serie de casos de prueba. Cada uno de ellos define un árbol, comenzando por el nodo raíz. Cada nodo comienza por la fecha de nacimiento, seguida de un número `m` ($0 \leq$ `m` ≤ 200) que indica la cantidad de nodos hijo. Después se describen recursivamente los árboles hijo, utilizando el mismo formato que para el padre.

La entrada finaliza con un `0`, que no se procesa.

Salida

Para cada caso se escribirá una línea con la cadena `SI` si el árbol es absurdo, o `NO` en caso contrario.

Entrada de ejemplo

Importante: Por motivos de claridad, a continuación se muestra cada caso de prueba separado en varias líneas, una por nodo, utilizando distintos niveles de sangrado para mostrar las relaciones entre nodos padre y nodos hijo. **Esto no tiene por qué ser así en la entrada proporcionada por el juez.** De hecho, tanto en el fichero `sample.in` proporcionado en el Campus Virtual como en la entrada utilizada por el problema correspondiente en *DOMjudge*, **cada árbol se expresa en una única línea**.

```
15/3/1970 3
  13/5/1998 0
  15/1/2005 0
  14/7/2006 0
7/9/1950 2
  12/1/1976 1
    14/11/1999 0
  12/4/1979 1
    15/5/1976 0
1/1/1950 1
  1/1/1950 0
1/1/1950 1
  31/12/1949 0
1/1/1950 1
  2/1/1950 0
0
```

Salida de ejemplo

```
NO
SI
SI
SI
NO
```