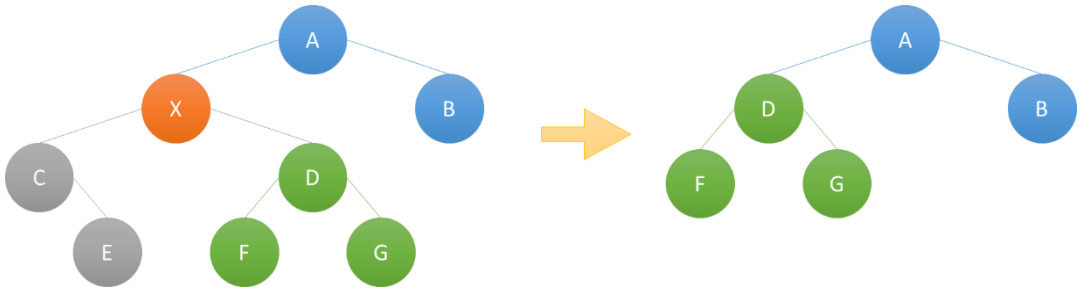
Grupo Weboo
Universidad de La Habana

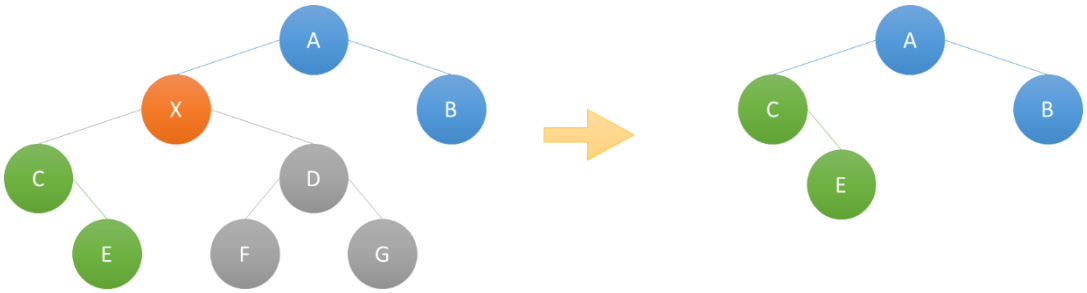
Examen Mundial de Programación
Curso 2014-2015

Simplificando Árboles

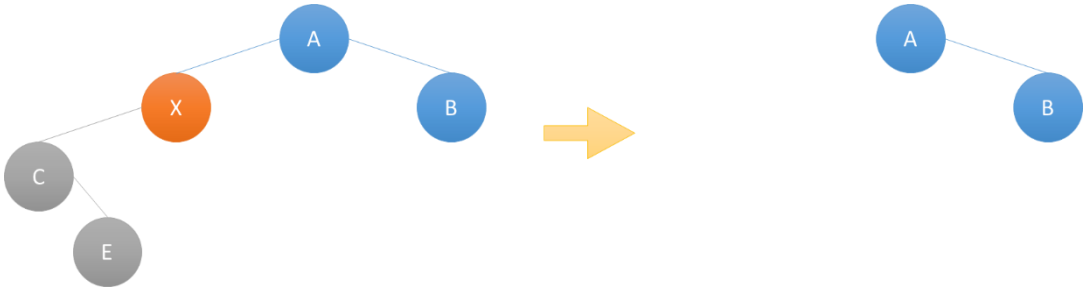
Sobre un árbol binario cualquiera se define la operación de **simplificación**, que consiste en eliminar un nodo arbitrario, reemplazándolo por uno de sus dos hijos, izquierdo o derecho (que puede ser o no `null`). Por ejemplo, en la figura siguiente, si se simplifica el nodo **X** (señalado en naranja) reemplazándolo por su hijo derecho (señalado en verde), se obtiene el árbol mostrado la derecha. Nótese que el nodo **X** y todo el subárbol de la izquierda (señalado en gris) desaparecen, ocupando su lugar el subárbol de la derecha con todos sus nodos.



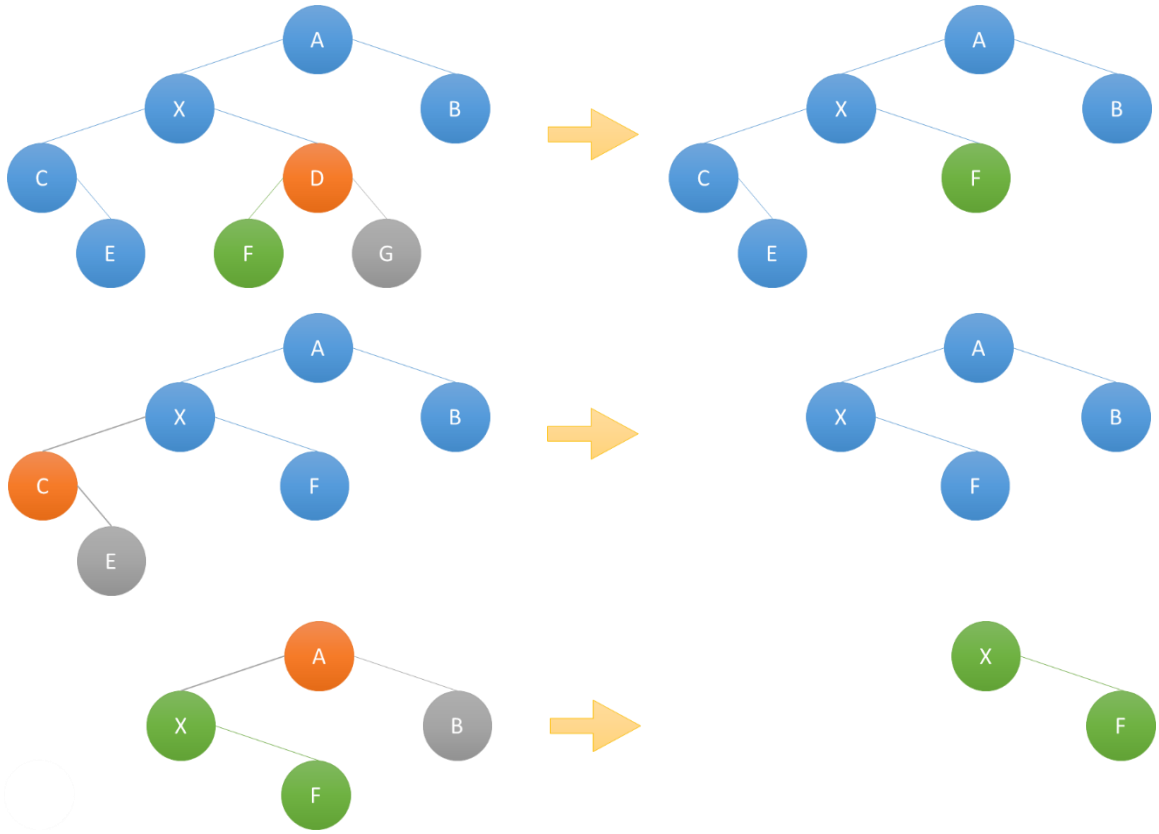
En caso contrario, si se reemplaza el nodo **X** por su hijo izquierdo, se obtiene entonces el árbol mostrado a continuación:



En caso de que el nodo a simplificar sea reemplazado por un hijo `null`, el resultado es equivalente a eliminar el nodo correspondiente. Por ejemplo, si en el caso anterior, el hijo derecho de **X** fuera `null`, y se decidiera simplificar **X**, reemplazándolo precisamente por el hijo derecho, el resultado es que se elimina por completo el subárbol que comienza en **X**.



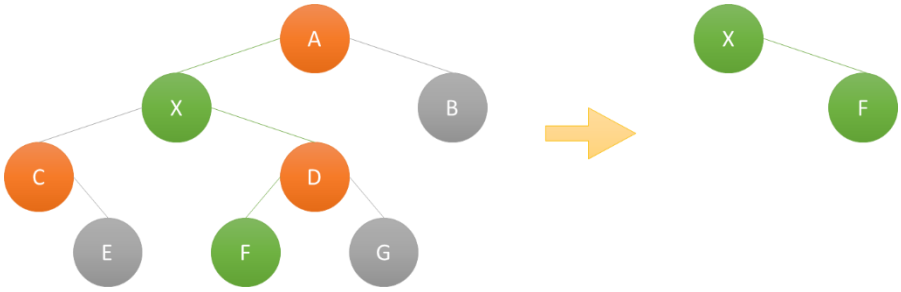
Aplicando una serie de operaciones de simplificación, es posible convertir un árbol en otro. Por ejemplo, en la siguiente secuencia, se aplican tres pasos de simplificación consecutivos.



En el primer caso, se elimina el nodo **D**, reemplazándolo por el hijo izquierdo **F**. En el segundo caso, se elimina el nodo **C** reemplazándolo por el hijo izquierdo. Como éste hijo es `null`, el nodo **C** y su hijo derecho **E** simplemente se eliminan. En el último paso, se simplifica el nodo raíz **A**, reemplazándolo por su hijo izquierdo **X**.

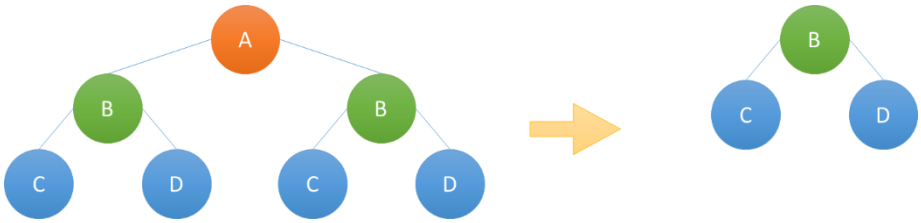
Su problema consiste en determinar la **menor cantidad de simplificaciones** que es necesario hacer para transformar cierto árbol **A** en otro árbol **B**. En caso de no ser posible esta transformación, el resultado será **-1**.

Por ejemplo, en la figura siguiente, la menor cantidad de simplificaciones necesarias para convertir el árbol de la izquierda en el árbol de la derecha es **3**, que son justamente los tres pasos mostrados en el ejemplo anterior, es decir, eliminar consecutivamente los nodos marcados en naranja, reemplazándolos por los hijos correspondientemente marcados en verde.

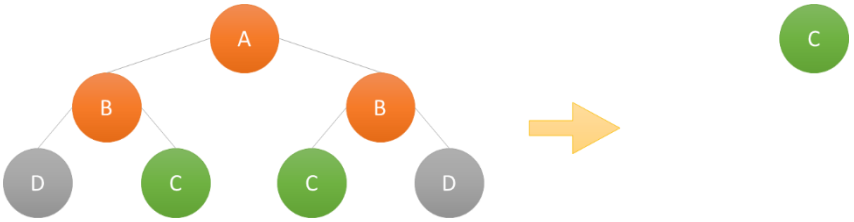


Note que es posible realizar esta misma transformación cambiando el orden de los pasos. Es posible eliminar primero la raíz **A**, luego el nodo **C**, y finalmente el nodo **D**; o eliminar estos nodos en orden inverso, obteniéndose el mismo resultado final. Sin embargo, no es posible, en el ejemplo anterior, realizar la transformación deseada con menos de **3** simplificaciones. De hecho, en este caso, como no existen valores repetidos, la única forma de realizar la transformación deseada es la mostrada, independientemente del orden.

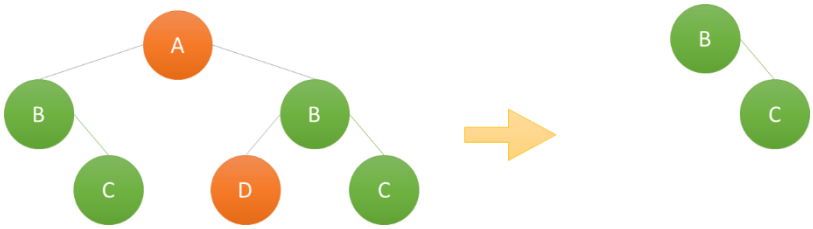
Sin embargo, en otros ejemplos, es posible que exista más de una secuencia de simplificaciones que realice la transformación correcta. Por ejemplo, si dos subárboles son iguales, existen al menos dos transformaciones posibles. En el ejemplo siguiente, cualquiera de los dos subárboles **B** pueden reemplazar al padre **A**, obteniéndose el mismo resultado.



Incluso si ambos subárboles no son exactamente iguales, es posible que exista una secuencia de simplificaciones equivalentes o simétricas, como en el caso del ejemplo siguiente. En este caso, es posible eliminar el nodo **B** de la izquierda manteniendo su hijo derecho, o el nodo **B** de la derecha manteniendo su hijo izquierdo, pero en ambos casos se realizan la misma cantidad de simplificaciones (**2**).

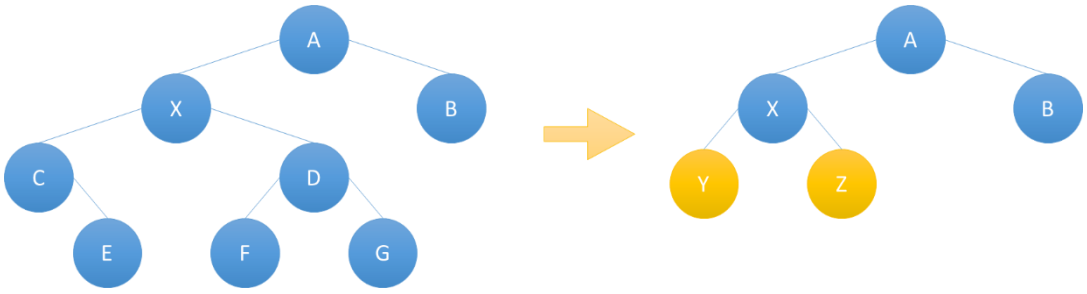


Por otro lado, es posible que existan varias formas de realizar la transformación, con un número distinto de simplificaciones requeridas. Por ejemplo, en el caso siguiente, manteniendo el hijo **B** de la izquierda solamente es necesaria una simplificación. Sin embargo, si se decide mantener el hijo **B** de la derecha, es necesario realizar una simplificación más para eliminar el nodo **D**. En este caso la respuesta correcta es **1**, que es el resultado de escoger al hijo **B** izquierdo para reemplazar a **A**.

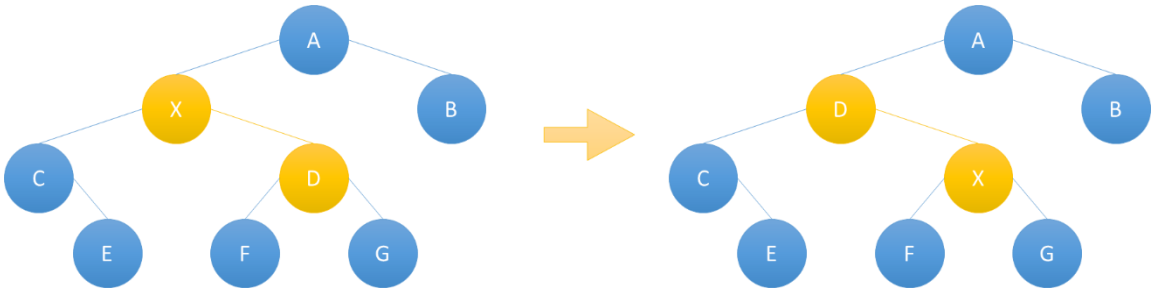


Finalmente, es posible que no exista ninguna secuencia de simplificaciones posible. En el caso más sencillo, si existen en el árbol objetivo nodos que no están presentes en el árbol original, evidentemente es imposible realizar la transformación, pues no existe forma de modificar el valor de un nodo, o añadir

nodos nuevos. Por ejemplo, en el siguiente caso los nodos **Y** y **Z** no están en el árbol de la derecha, por lo que no es posible realizar la transformación. En este caso la respuesta correcta es **-1**.



Otro caso en que no es posible realizar la transformación indicada, es cuando los nodos en el árbol objetivo tienen una relación distinta que en el árbol original. Por ejemplo, en el caso siguiente el nodo **D** aparece intercambiado con el nodo **X**. Dado que la operación de simplificación no permite modificar la relación padre-hijo, es imposible encontrar una secuencia de simplificaciones que transforme el árbol de la izquierda en el árbol de la derecha. En este caso, la respuesta correcta es **-1.**



Implementación

Usted deberá implementar el método siguiente, que devuelve la mínima cantidad de simplificaciones necesarias para convertir al árbol `a` en el árbol `b`:

```
public static class SimplificandoArboles
{
    public static int MinimaCantidadDeSimplificaciones<T>(IArbol<T> a, IArbol<T> b)
    {
        throw new NotImplementedException();
    }
}
```

La interfaz `IArbol<T>` representa un árbol binario, y tiene la siguiente forma:

```
public interface IArbol<T>
{
    T Valor { get; }
    IArbol<T> Izquierdo { get; }
    IArbol<T> Derecho { get; }
}
```

Esta interfaz se brinda en una biblioteca de clases adicional, a cuyo código usted no tiene (ni necesita) acceso. Se brinda además una implementación concreta de esta interfaz, en la clase `Arbol<T>`, que tiene la siguiente estructura:

```
public class Arbol<T>: IArbol<T>
{
    public Arbol(T valor, IArbol<T> izquierdo = null, IArbol<T> derecho = null)
    {
        Valor = valor;
        Izquierdo = izquierdo;
        Derecho = derecho;
    }

    public T Valor { get; private set; }
    public IArbol<T> Izquierdo { get; private set; }
    public IArbol<T> Derecho { get; private set; }
}
```

En la aplicación de consola que se brinda con ejemplos de prueba, se muestra como emplear esta clase. Por ejemplo, para probar el primer caso presentado en este documento, puede utilizar el siguiente código (presente en la aplicación de consola).

```
private static void Main(string[] args)
{
    var a = new Arbol<string>("A",
        new Arbol<string>("X",
            new Arbol<string>("C",
                null,
                new Arbol<string>("E")),
            new Arbol<string>("D",
                new Arbol<string>("F"),
                new Arbol<string>("G"))),
        new Arbol<string>("B"));

    var b = new Arbol<string>("X",
        null,
        new Arbol<string>("F"));

    Console.WriteLine(SimplificandoArboles.MinimaCantidadDeSimplificaciones(a, b)); // 3
}
```

Notas

- Se garantiza que nunca se invocará al método `MinimaCantidadDeSimplificaciones` con un argumento `null`.
- Su código debe funcionar con instancias de `IArbol` genéricas en cualquier tipo de dato, no solo `string`.
- La comparación entre dos árboles se realizará siempre por el valor de los nodos. Es decir, dos árboles son iguales si los nodos correspondientes en cada uno tienen los mismos valores. Es su responsabilidad implementar esta comparación de igualdad de forma conveniente. **No emplee** la implementación de `Equals` por defecto de la clase `Arbol<T>`.
- Recuerde que los casos de prueba existentes en la aplicación de consola **no son suficientes** para validar su implementación. Asegúrese de añadir todos los casos que considere necesarios. En

particular, recuerde añadir casos con tipos de datos distintos (`int`, `double`, `float`, `char`, `bool`, `string`).

- Recuerde guardar a menudo para evitar complicaciones por ausencia del fluido eléctrico.