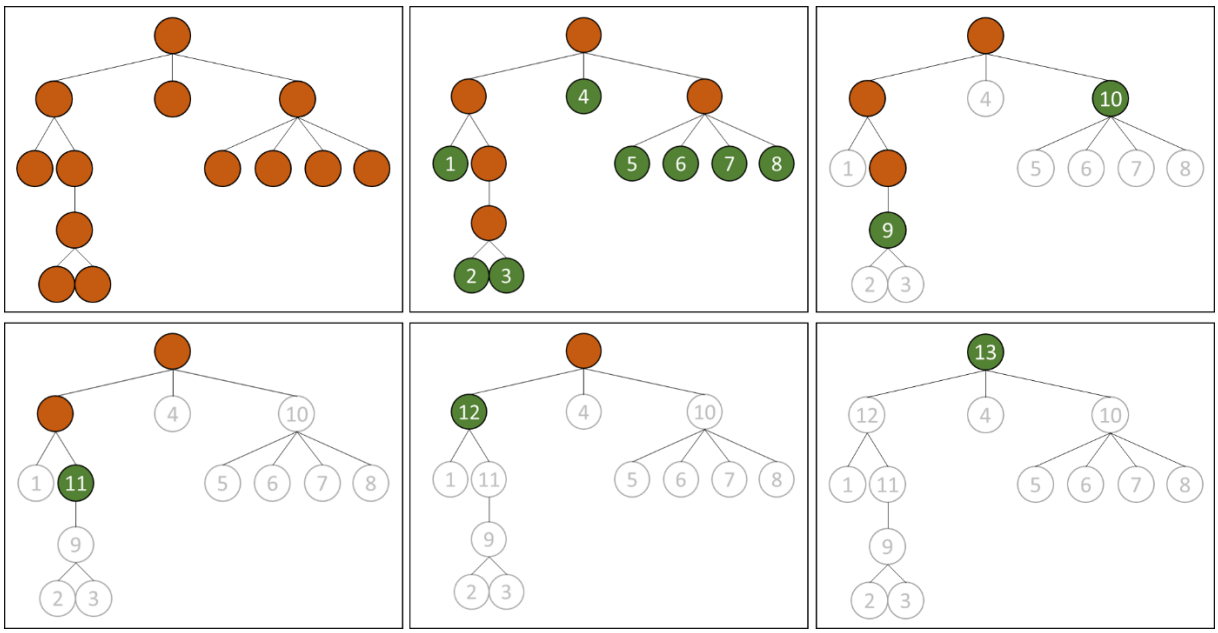
Grupo Weboo
Universidad de La Habana

Examen Final de Programación
Curso 2013-2014

Iterador de Árboles de las hojas a la raíz

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que descargó del sitio, ciérrelo ahora, extraiga todos los archivos en el escritorio, y siga trabajando desde ahí. Es un error común trabajar en la solución dentro del compactado, lo cual provoca que los cambios **no se guarden**. Si usted comete este error y entrega una solución vacía, **no tendrá oportunidad de reclamar**.

Se desea recorrer los nodos de un árbol de modo que los primeros nodos en iterarse sean las hojas de izquierda a derecha. Luego le seguirían aquéllos nodos que se volverían hojas si se eliminasen las hojas de la primera iteración y así sucesivamente hasta terminar en la raíz. Por ejemplo, en el árbol de la figura los números indican el orden en que se iteran los nodos.



Usted deberá implementar el tipo `Recorridos`, dentro del `namespace` `Weboo.Examen.Final` , que tiene el método genérico `IteraPrimeroHojas`. Sobre cada nodo se quiere realizar cierta operación y obtener un enumerable con los resultados. Para ello su método recibirá un delegado de tipo `Func<T, R>` donde `T` representa el tipo de elementos del árbol y `R` el tipo de valor resultante de cada operación. La signatura del método que usted debe implementar es como se muestra (Nota: a usted se le provee de una plantilla de VS que contiene esta definición, por tanto, usted no tiene que copiar esta definición en su solución).

```
namespace Weboo.Examen.Final
{
    public class Recorridos
    {
        public static IEnumerable<R> IteraPrimeroHojas<T, R>(Arbol<T> arbol, Func<T,
R> op)
        {
```

```
            /// Elimine esta línea inmediatamente...
            throw new NotImplementedException();
        }
    }
}
```

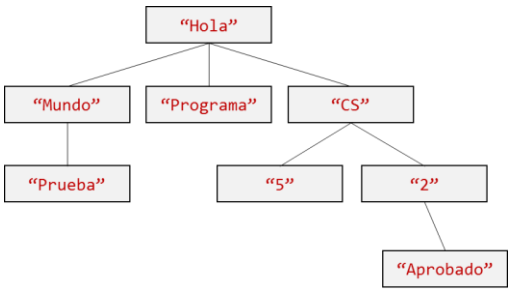
La definición de árbol n-ario se le provee en la biblioteca de clases **`Weboo.Arboles.dll`**. Según esta definición un árbol es inmutable, es decir, usted no puede agregar o eliminar nodos de un árbol, ni cambiar sus valores.

El siguiente código ilustra las funcionalidades del tipo `Arbol<T>`.

```
public class Arbol<T>
{
    public T Valor { get {…} }
    public IList<Arbol<T>> Hijos { get {…} }

    public Arbol(T valor, params Arbol<T>[] hijos)
    {
        …
    }
}
```

Ejemplos

A continuación se muestran algunos ejemplos que usted puede probar.



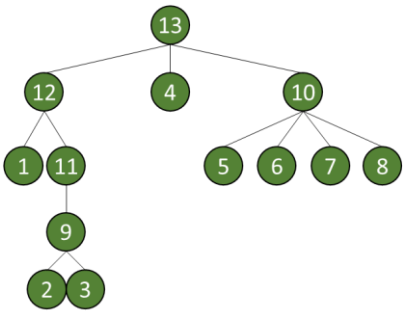

```
Arbol<string> a2 = new Arbol<string>("Hola",
                        new Arbol<string>("Mundo",
                            new Arbol<string>("Prueba")),
                        new Arbol<string>("Programa"),
                        new Arbol<string>("CS",
                            new Arbol<string>("5"),
                            new Arbol<string>("2",
                                new Arbol<string>("Aprobado"))));

IEnumerable<int> longitudes = Recorridos. IteraPrimeroHojas(a2, s => s.Length);
```

Recorrer este iterador y escribir los valores con

```
foreach (var x in longitudes)
 Console.Write("  {0},", x);
```

daría los valores `6, 8, 1, 8, 5, 1, 2, 4` ya que son respectivamente las longitudes de las cadenas `"Prueba", "Programa", "S", "Aprobado", "Mundo", "2", "CS"` y `"Hola"`

Para el árbol




```
Arbol<int> a3 = new Arbol<int>(13,
                    new Arbol<int>(12,
                        new Arbol<int>(1),
                        new Arbol<int>(11,
                            new Arbol<int>(9,
                                new Arbol<int>(2),
                                new Arbol<int>(3)))),
                    new Arbol<int>(4),
                    new Arbol<int>(10,
                        new Arbol<int>(5),
                        new Arbol<int>(6),
                        new Arbol<int>(7),
                        new Arbol<int>(8)));
```

El iterador

```
IEnumerable<bool> sonPrimos = Recorridos. IteraPrimeroHojas(a3, x => EsPrimo(x));
```

debe dar los valores `false, true, true, false, true, false, true, false, false, false, true, false, true`

El enumerable resultante debe transformar los valores del recorrido por “demanda”, es decir no puede aplicar la transformación, guardar los valores transformados en una estructura de datos y después devolver un iterador de dicha estructura. Por lo tanto, si el árbol a continuación (note que todos los nodos son cero)

```
Arbol<int> a = new Arbol<int>(0,
                    new Arbol<int>(0),
                    new Arbol<int>(0));
```

se invocara con el código

```
var e = Recorridos. IteraPrimeroHojas(a, x => 1 / x);
```

en donde la función para transformar el valor del nodo es la expresión lambda `x => 1 / x`, la ejecución de su método `IteraPrimeroHojas` no debe provocar ninguna excepción porque la transformación `1/x` aún no se ha aplicado. Solo cuando se intente iterar sobre dicho resultado haciendo

```
foreach (var x in e) // la excepción ocurre aquí al tratar de acceder al primero
    Console.WriteLine(x);
```

es que ocurrirá la excepción.