Subárboles maximales

> **NOTA**: Antes de comenzar, descompacte el proyecto entregado, y vaya a la carpeta `exam`, archivo `Exam.cs`, y al final de la clase estática `Exam` encontrará dos propiedades donde debe devolver su nombre y grupo. Esto es imprescindible para ser evaluado, pues será el nombre y grupo que se usará en el acta del examen. Todo lo que usted implemente (funciones, clases auxiliares, etc) que de alguna forma se use en el método a implementar, debe encontrase en el archivo `Exam.cs`, dentro del único namespace que el mismo contiene. Para evaluar su solución solo se trandrá en cuenta dicho archivo, cualquier otro será descartado lo que puede resultar en un error de compilación si se usa alguna funcionalidad implementada en ellos.

Usted debe haber recibido una solución de .NET Core con una biblioteca de clases llamada `trees`. Dentro encontrará un archivo `Trees.cs` donde verá las siguientes definiciones.

Una interfaz que define un árbol n-ario genérico:

```
public interface ITree<T>
{
    T Value { get; }
    IEnumerable<ITree<T>> Children { get; }
}
```

Un delegado que representa un predicado arbitrario sobre un tipo genérico:

```
public delegate bool Predicate<T>(T item);
```

Además en la solución encontrará un proyecto de consola llamado `exam` con un archivo `Exam.cs`, dentro verá una clase estática `Exam` donde deberá implementar su solución al siguiente problema.

A partir de las dos definiciones anteriores, usted debe implementar el método `MaximalSubtreesWhere`, que devuelve todos los subárboles maximales tales que todos sus elementos (incluyendo el valor de la raíz del subárbol) cumplen con cierto predicado:

```
public static IEnumerable<ITree<T>> MaximalSubtreesWhere<T>(
    ITree<T> tree, Predicate<T> predicate)
{
    // ponga su código aquí
    throw new NotImplementedException();
}
```

Definición: Un subárbol es simplemente cualquier descendiente del árbol pasado como argumento al método, incluyendo el propio árbol. Un subárbol maximal es aquel que cumple que no es hijo de ningún otro subárbol que cumpla con la misma propiedad.

Se garantiza que ningún argumento será `null`, pero si es posible que ningún valor cumpla con el predicado, en cuyo caso su respuesta debe ser un *enumerable vacío*, nunca un valor `null`.

Ejemplo

Por ejemplo, suponga el árbol siguiente:

```
5
 - 2
   - 3
   - 6
 - 4
   - 0
   - 8
```

Y suponga que el predicado es `lambda x: x % 2 == 0`, o sea, todos los valores pares.

Entonces los nodos con valor `2`, `6`, `4`, `0` y `8` cumplen con la condición del predicado. Sin embargo, como el nodo con valor `3` no la cumple, el subárbol con raíz en `2` no debe ser devuelto, pues no todos los elementos cumplen con el predicado.

Por otro lado, ambos subárboles con raíz en `0` y `8` cumplen que todos sus elementos pasan el predicado (ambos subárboles tienen un solo elemento), pero como el subárbol con raíz en `4` también lo cumple, y es padre de los dos anteriores, ellos no son maximales.

Por lo tanto, la respuesta para este ejemplo sería un enumerable donde se devuelven el subárbol con raíz en `6` (que es hoja) y el subárbol con raíz en `4`.

Requisitos

Usted debe garantizar que su solución cumple los siguientes requisitos:

- Los subárboles se devuelven en el mismo orden en que aparecerían en un recorrido en *pre-orden*.
- El enumerable devuelto es *lazy*, o sea, cada subárbol se devuelve lo antes posible. Esto significa que su solución debe funcionar para árboles potencialmente infinitos (lo cuál será probado).
- En el enumerable devuelto están exactamente las mismas instancias de `ITree<T>` que recibió como argumento, o sea, usted nunca necesita crear nuevas instancias de árboles.
- Asegúrese que todo el código necesario para ejecutar su solución está en el archivo `Exam.cs`. Si necesita métodos o tipos adicionales, defínalos en ese mismo archivo.

- Bajo ningún concepto puede modificar la definición de `ITree<T>` o de `Predicate<T>`, ni la signatura del método `MaximalSubtreesWhere`. De hacerlo su examen dará error de compilación y no será posible reclamar.

Sugerencias

- Utilice la instrucción `yield return` para devolver los subárboles a medida que los descubre, en vez de almacenarlos todos en un array o lista y devolverlo al final (lo cuál no funcionará con árboles infinitos).
- Como ya es usual en este exámen no hay casos de prueba, es su responsabilidad adicionar en la aplicación de consola los casos de prueba y testear su solución tanto como considere necesario. Si para ello necesita su propia implementación de `ITree<T>`, siéntase libre, pero ese código en particular no tiene que entregarlo.