Simplificando un Árbol

Se desean eliminar de un árbol los nodos que no cumplan cierta condición. Dicha eliminación debe hacerse de forma tal que el árbol resultante mantenga la relación de ancestro-descendientes del árbol original y que el recorrido en pre-orden siga siendo el mismo salvo los nodos que se han eliminado.

Se garantiza que la raíz siempre cumple la condición de modo que nunca será necesario eliminarla.

En el siguiente ejemplo se desea simplificar el árbol de la izquierda (Figura 1) eliminando los nodos que tienen un valor par lo que da como resultado el árbol que se muestra a la derecha (Figura 2).

Ejemplo:

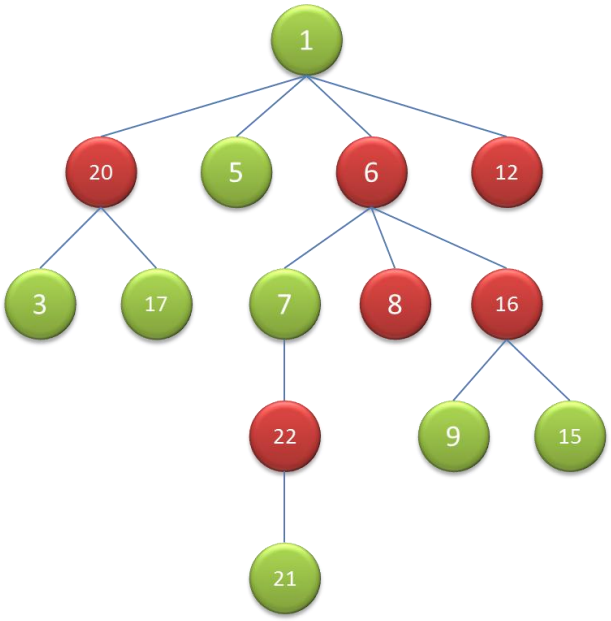

Figura 2 - Árbol original

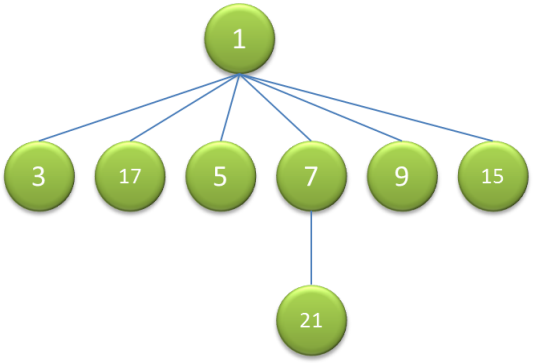

Figura 1 - Árbol simplificado como resultado de eliminar los nodos pares.

Nótese cómo en el árbol resultante se mantiene la relación ancestral original entre los nodos resultantes. Por ejemplo, los nodos 3 y 17 son ahora hijos directos de la raíz ya que el nodo padre, el 20, ha debido ser eliminado por tener valor par; en el caso de los nodos 9 y 15 han pasado a ser hijos directos del ancestro más cercano que cumplía la condición, en este caso la raíz. En el caso del nodo 21, pasa a ser hijo del ancestro más cercano que cumple la condición, el 7.

Figura 3 - Recorrido en pre-orden del árbol original

Figura 4 - Recorrido en pre-orden del árbol después de aplicada la simplificación

En la Figura 3 y Figura 4 puede apreciarse como una vez aplicada la simplificación la relación de orden entre los elementos dada por el recorrido en pre-orden se mantiene.

Usted deberá implementar un método que reciba los parámetros `Árbol<T> arbol` y `Condicion<T> condicion`, el método deberá simplificar el árbol recibido como parámetro de la manera descrita anteriormente. Los nodos que serán eliminados son aquellos que no cumplan la condición especificada.

```
public static void Simplifica<T>(Arbol<T> arbol, Condicion<T> condicion)
{
      …
}
```

A continuación se muestra un ejemplo de uso del método `Simplifica`:

```
//Construcción del árbol utilizado en el ejemplo
//ver Figura 1
Arbol<int> arbol = new Arbol<int>(1,
    new Arbol<int>(20,
        new Arbol<int>(3),
        new Arbol<int>(17)
    ),
    new Arbol<int>(5),
    new Arbol<int>(6,
        new Arbol<int>(7,
            new Arbol<int>(22,
                new Arbol<int>(21)
            )
        ),
        new Arbol<int>(8),
        new Arbol<int>(16,
            new Arbol<int>(9),
            new Arbol<int>(15)
        )
    ),
    new Arbol<int>(12)
);

//Se invoca el método Simplifica utilizando como condición un delegate
//anónimo que comprueba si el valor de la raiz es impar
Simplifica(arbol, delegate (int ar) { return ar % 2 == 1; });

//Imprime en consola el árbol simplificado
arbol.ImprimeArbol();
```

En el desarrollo de su solución usted deberá trabajar con la clase `Arbol<T>` y el delegado `Condicion<T>` que se le facilitan con este examen. Usted no puede crear su propia definición de las mismas. A continuación se describen ambos tipos:

```
public class Arbol<T>
{
    /// <summary
    /// Valor del nodo raiz
    /// </summary>
    public T Valor { get {…}}


    /// <summary>
    /// IEnumerable de Árboles Hijos. Devuelve una copia exacta de los hijos
    /// en el momento que es invocado
    /// </summary>
    public IEnumerable<Arbol<T>> Hijos { get{…} }

    /// <summary>
    /// Crea un árbol sin hijos con el valor especificado
    /// </summary>
    /// <param name="valor">valor para el nodo raiz</param>
    public Arbol(T valor){…}

    /// <summary>
    /// Crea un árbol con el conjunto de árboles especificados como hijos
    /// </summary>
    /// <param name="valor"></param>
    /// <param name="hijos"></param>
    public Arbol(T valor, params Arbol<T>[] hijos)
        : this(valor) {…}

    /// <summary>
    /// Poda el árbol eliminando el hijo especificado, utilizando la igualdad
    /// por referencia
    /// </summary>
    /// <param name="hijo">subárbol que será eliminado</param>
    public void EliminaHijo(Arbol<T> hijo) {…}

    /// <summary>
    /// Inserta un nuevo árbol dentro de la lista de árboles hijos.
    /// </summary>
    /// <param name="hijoAInsertar">Árbol a insertar</param>
    /// <param name="hijo">Árbol hijo delante del cual se quiere insertar el nuevo árbol.
    /// Si el parámetro es null o no se ecuentra dentro de la lista
    /// de árboles hijos, entonces el nuevo árbol se inserta al final de la lista de
    /// hijos</param>
    public void InsertaHijo(Arbol<T> hijoAInsertar, Arbol<T> hijo) {…}



    /// <summary>
    /// Añade un nuevo hijo al árbol. El hijo agregado pasará a ser el último hijo del
    /// árbol.
    /// </summary>
    /// <param name="child"></param>
```

```
    public void AgregaHijo(Arbol<T> hijo) {…}

    /// <summary>
    /// Enumera los elementos del árbol en Preorden
    /// </summary>
    public IEnumerable<T> Preorden { get {…}}

    /// <summary>
    /// Imprime un arbol en consola siguiendo un recorrido en Preorden
    /// </summary>
    public void ImprimeArbol(){…}

}

/// <summary>
/// Devuelve true si la condición que representa se
/// cumple para el parámetro especificado.
/// Algunos ejemplos de valores que se le pueden
/// asignar a este delegate:
///     Condicion<int> c = (int x)=> x % 2 == 1
///     Condicion<int> c = delegate (int ar) { return ar % 2 == 1; })
///     Condicion<int> c = EsImpar /*donde EsImpar es un método que recibe un entero
///     y determina si es impar*/
/// </summary>
/// <typeparam name="T">Tipo del parámetro</typeparam>
/// <param name=" elemento">Parámetro para evaluar la condición</param>
/// <returns>True si la condición se cumple</returns>
public delegate bool Condicion<T>(T elemento);
```