Multi-Conjuntos

Exámen Mundial de Programación II - Curso 2022

NOTA: *Antes de comenzar asegúrese de descompactar el archivo* ***exam.zip*** *y abrir la solución* ***multiset.sln*** *en su editor. Asegúrese también de que su código compila, y la aplicación de consola ejecuta (debe lanzar una excepción). Recuerde que todo el código a evaluar debe ir en el archivo* ***Exam.cs*** *de la aplicación de consola* ***exam****.*

Un multi-conjunto es un conjunto que puede contener tanto elementos simples de un tipo genérico como otros conjuntos. Lo representaremos mediante la interfaz `IMultiSet<T>`, que se muestra a continuación:

```
public interface IMultiSet<T> : IEnumerable<T>
{
    // Cantidad de elementos total en el multi-conjunto
    int Count { get; }

    // Añadir un elemento al multi-conjunto
    void Add(T element);

    // Verificar si un elemento pertenece al multi-conjunto
    bool Contains(T element);

    // Crear un nuevo multi-conjunto anidado
    IMultiSet<T> CreateSubset();

    // Encontrar todos los elementos que cumplen con el filtro
    IEnumerable<T> Find(Predicate<T> filter);

    // Eliminar una o más referencias de un elemento
    void Remove(T item, int depth);
}
```

Para crear un nuevo multi-conjunto vacío, usted debe implementar el método `CreateMultiSet` de la clase estática `Exam` (en el archivo `exam.cs`):

```
public static class Exam
{
    // ...

    public static IMultiSet<T> CreateMultiSet<T>()
    {
        // Borre este código y devuelva una nueva instancia de su clase
        throw new NotImplementedException();
    }
}
```

Una manera de ejemplicar el uso de esta estructura es para representar organizaciones con sub-organizaciones. De esta forma, para crear un nuevo multi-conjunto vacío de `string` que represente la facultad, se ejecutaría la siguiente línea.

```
IMultiSet<string> matcom = Exam.CreateMultiSet<string>();
```

Para añadir elementos a un multi-conjunto se usa el método `Add`:

```
matcom.Add("Guinovart");
matcom.Add("Idania");
```

La propiedad `Count` devuelve la cantidad total de elementos en el multi-conjunto.

```
Debug.Assert(matcom.Count == 2);
```

La cosa se empieza a poner interesante cuando creamos un subconjunto del multi-conjunto original y le añadimos elementos.

```
IMultiSet<string> pro = matcom.CreateSubset();

pro.Add("Piad");
pro.Add("Leynier");
pro.Add("Mauricio");

Debug.Assert(pro.Count == 3);
```

Como estos elementos son también parte del conjunto original, la propiedad `Count` de `matcom` debe reflejar también este cambio:

```
Debug.Assert(matcom.Count == 5);
```

El método `Contains` sirve para verificar si un elemento está en el multi-conjunto (incluyendo todos los multi-conjuntos anidados):

```
Debug.Assert(matcom.Contains("Guinovart"));
Debug.Assert(matcom.Contains("Leynier"));
Debug.Assert(!pro.Contains("Idania")); // Este no está en pro
```

Por supuesto, debe ser posible crear cualquier cantidad de subconjuntos de un mismo multi-conjunto raíz, y así recursivamente en cualquier nivel.

```
IMultiSet<string> matematica = matcom.CreateSubset();

matematica.Add("Yudivian");
matematica.Add("Celia");
matematica.Add("Idania");
```

Nótese que `"Idania"` se ha añadido también a un subconjunto, esto será relevante más adelante.

El método `Find` devuelve todos los elementos que cumplen con un predicado:

```
string[] shortNames = matcom.Find(s => s.Length <= 5).ToArray();
Debug.Assert(shortNames.Length == 2);
```

Además, como `IMultiSet` implementa la interfaz `IEnumerable`, también es posible recorrer todos los elementos que existen en el multi-conjunto:

```
foreach(var x in matcom)
{
    Console.WriteLine(x);
}

// Guinovart
// Idania
// Piad
// Leynier
// Mauricio
// Yudivian
// Celia
```

Por supuesto, si un elemento se añade más de una vez, no debe aparecer duplicado, incluso si se añade a un subconjunto.

En el caso anterior, `"Idania"` es miembro de la raíz `matcom` y también del subconjunto `matematica`. Por lo tanto, debe contarse una sola vez en `matcom` tanto en el `Count` como en todos los recorridos. Sin embargo, debe aparecer también una sola vez en `matematica` porque fue explícitamente añadido ahí.

```
foreach (var x in matematica)
{
    Console.WriteLine(x);
}

// Yudivian
// Celia
// Idania
```

Por último, el método `Remove` permite eliminar una o más referencias a un elemento en diferentes niveles del multi-conjunto. Si `depth == 0` entonces se debe eliminar el elemento solamente si está agregado explícitamente en el nivel actual.

```
matcom.Remove("Idania", 0);
// Aunque se eliminó de la raíz, este elemento sigue existiendo
// en el subconjunto `matematica`
Debug.Assert(matematica.Contains("Idania"));
// Y por tanto el `Count` de matcom no cambia
Debug.Assert(matcom.Count == 7);
```

Si `depth == 1`, entonces se eliminan todas las referencias a dicho elemento en el nivel actual y el primer nivel de los subconjuntos directamente incluidos en el actual, y así sucesivamente. Si `depth < 0` entonces es equivalente a decir que `depth` tiene un valor infinito, por tanto elminirá todas las referencias del

elemento, incluyendo la del nivel actual (en caso de existir) y en cada multi-conjunto anidado que pertenezca a un multi-conjunto que descienda del nivel actual.

NOTA: Recuerde que el criterio de igualdad por defecto a usar en `.NET` está definido por el método estático `Object.Equals`.

Ejemplos de prueba

En la aplicación de consola encontrará un ejemplo de prueba muy similar a lo que hemos visto hasta ahora, que le permitirá verificar que los métodos básicos funcionan.

NOTA: El ejemplo de prueba es insuficiente para garantizar que su código está 100% correcto. Es su responsabilidad adicionar tantos casos de prueba como considere necesario para garantizar la correctitud de su solución.

¡Éxitos a todos!