MATCOM

Grupo Weboo
Universidad de La Habana

Examen Mundial de Programación
Curso 2018-2019

Procesamiento de Sonido

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que se le entregó, 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.

Los *sistemas de procesamiento de sonido* analizan, modifican o sintetizan las señales recibidas como entrada. Las señales son representaciones digitales de ondas sonoras y consisten en una secuencia de números enteros que miden la cantidad de oscilaciones por segundo (frecuencia) en cada muestra tomada. Generalmente, estos sistemas se definen como secuencias de transformaciones, o **filtros**, sucesivos que se les deben aplicar a las señales. Este mecanismo facilita el tratamiento de las señales, pues permite conectar diversas técnicas (eliminación de ruido, reajuste de valores, compresión, normalización, etc.) en un mismo proceso.

Se desea implementar emulador de un sistema de procesamiento de sonidos que permita monitorear su comportamiento sobre el tiempo.

```
public interface IProcesadorDeSonido
{
    /// Cantidad de filtros en la línea de procesamiento.
    int CantidadDeFiltros { get; set; }

    /// Agrega un filtro al final de la línea de procesamiento.
    int AgregaFiltro(TipoDeFiltro tipo);

    /// Encola una señal de sonido en la línea de procesamiento.
    void AgregaAudio(Audio audio);

    /// Tiempo que necesita el filtro indicado en terminar la señal de
    /// sonido que está procesando.
    int TiempoDeProcesamientoPendiente(int indiceDeFiltro);

    /// Le indica al simulador que ejecute una cierta cantidad de unidades
    /// de tiempo y devuelve las señales de sonido terminadas en ese tiempo.
    IEnumerable<Audio> ProcesaAudio(int tiempo);

    /// Devuelve el audio en procesamiento por cada uno de los filtros.
    IEnumerable<Audio> AudioPorFiltro { get; }

    /// Devuelve las señales de sonido en cola, en el orden inverso en
    /// que fueron encoladas.
    IEnumerable<Audio> AudioEnCola { get; }

}
```

La clase `Audio` representa una señal de sonido y se compone de un identificador (**`Nombre`**) y un array de números enteros (**`Frecuencias`**).

Los filtros son componentes que transforman las señales. Un filtro recibe una señal como entrada y **siempre comprueba** si puede o debe transformar sus frecuencias. Aplica la transformación a **sus frecuencias**, en caso de ser posible, y luego devuelve la misma señal transformada (**la misma referencia**). Tanto la etapa de comprobación como la de transformación consumen tiempo. En todos los filtros, el proceso de comprobación consume tantas unidades de tiempo como la cantidad de frecuencias que tenga la señal (**longitud del array**). La cantidad de unidades de tiempo que demora la etapa de transformación depende de los valores o estructura de las frecuencias de la señal y del propio filtro. **Si el filtro no puede o debe aplicar la transformación, esta etapa no consume tiempo**. Existen tres tipos de filtros: ***Reajuste***, ***Compresión*** y ***Normalización***.

- *Reajuste*: El objetivo de este filtro es rotar hacia la derecha **la menor cantidad de veces posible** el array de frecuencias hasta ubicar al **menor elemento** del array en la primera posición. La cantidad de unidades de tiempo que demora este filtro en completar su transformación es el número de rotaciones necesarias. Este filtro no se deberá aplicar si el menor elemento está en la primera posición.
- *Compresión*: El filtro de compresión reduce la cantidad de frecuencias de la señal. Este toma cada valor en posiciones pares del array y lo promedia con el valor de su posición siguiente, dando origen a un nuevo array con la mitad de frecuencias que tenía el anterior. Demora una cantidad de unidades de tiempo igual al mayor elemento del array. El mismo no se puede aplicar cuando el array de frecuencias es de longitud impar.
- *Normalización*: Con este filtro calculamos las diferencias relativas respecto al menor elemento del array. Consume una cantidad de unidades de tiempo igual a la suma de las frecuencias resultantes. Si existe algún valor igual a 0 o solo hay un elemento no se puede aplicar el filtro.

El mecanismo que se quiere modelar aquí es una línea de procesamiento. La línea de procesamiento es una secuencia de filtros que transforman sucesivamente las señales. El filtro inicial recibe la señal como se le suministra al sistema. A continuación del primero, el resto de los filtros reciben como entrada la salida de su predecesor, teniendo que esperar a que este termine su procesamiento. El último filtro devuelve la señal luego de haber recibido todas las transformaciones del sistema.

La interfaz `IProcesadorDeSonido` modela este mecanismo y cuenta con diversos métodos y propiedades para conocer el estado del sistema durante la emulación.

El sistema se inicia como una línea vacía, sin filtros definidos. Para agregar un nuevo filtro al final de la secuencia (o como primer elemento) utilizamos el método **`AgregaFiltro`**, debiendo especificar su tipo. Mediante la propiedad **`CantidadDeFiltros`** podemos conocer cuántos filtros hay en la línea de procesamiento.

El método **`AgregaAudio`** nos permite incluir una **nueva señal** de sonido en el sistema. Si el primer filtro está libre, se ubica inmediatamente aquí. Si está ocupado por otra señal entonces se agrega a la cola del sistema. Las señales de sonidos serán **procesadas en el orden en que fueron agregadas** al sistema. Sin embargo, la propiedad **`AudioEnCola`** nos ofrece las **señales de sonido** que se encuentran **en cola**, en el **orden inverso a como fueron agregadas**.

El método **`TiempoDeProcesamientoPendiente`** devuelve **cuántas unidades de tiempo** le **faltan** al filtro, indicado como parámetro, **para completar** la **tarea actual** sobre la señal que está procesando. En caso de que el filtro esté **comprobando** si la señal no requiere transformación, deberá devolver el tiempo que falta para completar esta fase. Si, por el contrario, el filtro está

transformando la señal, se devolverá el tiempo que resta para finalizar la tarea. **Si el filtro no está procesando ninguna señal, debe devolver -1**.

Podemos conocer cada una de las señales que están procesando los filtros, de acuerdo con el orden de la línea, consultando la propiedad **`AudioPorFiltro`**. Si un filtro no está procesando ninguna señal de sonido, entonces para esta posición se devolverá **`null`**.

Finalmente, la emulación se realiza con el método **`ProcesaAudio`**. Dicho método le indica al emulador que procese la cantidad de unidades de tiempo indicada como parámetro. Las señales **terminadas** de procesar se **devuelven según su orden de finalización**. En cuestiones de tiempo todo filtro de la línea de procesamiento depende de su sucesor y predecesor. Un filtro **no puede comenzar** su procesamiento **hasta** que el que le **precede no haya terminado**. Igual ocurre si un filtro completa su trabajo y el que le **sigue** todavía **no ha terminado**. **En ambos casos deben esperar**, sin hacer nada, **las unidades de tiempo de más que tomó su correspondiente vecino**.

Las transformaciones se le aplican a las frecuencias de la señal una vez que se haya completado el tiempo de la transformación.

El sistema no consume tiempo pasando las señales de la cola de señales al primer filtro, ni tampoco pasando las señales entre filtros.

Usted debe haber recibido junto a este documento una solución de Visual Studio con dos proyectos: una biblioteca de clases (*Class Library*) y una aplicación de consola (*Console Application*). Usted debe completar la implementación de la clase `ProcesadorDeSonido` en el *namespace* `Weboo.Examen` que ya implementa la interface `IProcesadorDeSonido`. En la biblioteca de clases encontrará la siguiente definición:

```
namespace Weboo.Examen
{
    public class ProcesadorDeSonido : IProcesadorDeSonido
    {
        public ProcesadorDeSonido()
        {
            throw new NotImplementedException();
        }

        ...
    }
}
```

NOTA: Los casos de prueba que aparecen en este proyecto son solamente de ejemplo. Que usted obtenga resultados correctos con estos casos no es garantía de que su solución sea correcta y de buenos resultados con otros ejemplos. De modo que usted debe probar con todos los casos que considere convenientes para comprobar la validez de su implementación.

Usted **puede asumir** que:

- Todo array de frecuencias tendrá al menos un elemento.
- Las señales de sonido recibidas en el método **`AgregaAudio`** serán diferentes de **`null`**.
- No se procesarán señales de sonido sin haber agregado al menos un filtro.
- Los índices del método **`TiempoDeProcesamientoPendiente`** serán válidos.
- El tiempo de procesamiento siempre será positivo.

- Los enumerables devueltos en los métodos de la interfaz `IProcesadorDeSonido` se consumirán inmediatamente (a continuación de la llamada al método) y una única vez.

Definiciones complementarias:

```
public class Audio
{
    public string Nombre { get; set; }

    public int[] Frecuencias { get; set; }
}
public enum TipoDeFiltro
{
    Reajuste,
    Compresion,
    Normalizacion
}
```

En la aplicación de consola **`Tester`**, recibirá un código de ejemplo con el que puede probar su solución. Este código no garantiza la correctitud de su respuesta.

NOTA: Todo el código de la solución debe estar en este proyecto (biblioteca de clases), pues es el único código que será evaluado. Usted puede adicionar todo el código que considere necesario, pero no puede cambiar los nombres del namespace, clase o método mostrados. De lo contrario, el probador automático fallará.