Problema 3 de Programación
Curso 2012-2013
Matriz Esparcida

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.

La siguiente clase representa una celda de una matriz:

```
/// <summary>
/// Describe una celda de una matriz
/// </summary>
public class CeldaMatriz
{
    /// <summary>
    /// Crea una instancia de la clase CeldaMatriz
    /// </summary>
    /// <param name="fila">Fila donde se encuentra el valor</param>
    /// <param name="columna">Columna donde se encuentra el valor</param>
    /// <param name="valor">Valor</param>
    public CeldaMatriz(int fila, int columna, int valor)
    {
        Valor = valor;
        Columna = columna;
        Fila = fila;
    }

    /// <summary>
    /// Especifica la fila de la celda
    /// </summary>
    public int Fila { get; private set; }

    /// <summary>
    /// Especifica la columna de la celda
    /// </summary>
    public int Columna { get; private set; }

    /// <summary>
    /// Especifica o modifica el valor en la celda
    /// </summary>
    public int Valor { get; set; }

    /// <summary> Devuelve la representación literal de una celda </summary>
    /// <returns>Una cadena que represetna la celda</returns>
    public override string ToString()
    {
        return string.Format("{0} en ({1};{2})", Valor, Fila, Columna);
    }
}
```

NOTA: La clase descrita anteriormente ya está implementada como parte de los ensamblados que se proveen para la realización del examen. Esta clase ha sido implementada para usarse tal y como se ha descrito, usted no necesita realizar su propia implementación de la misma.

La siguiente `interface` ha sido diseñada para encapsular las operaciones básicas de una Matriz.

```
public interface IMatriz: IEnumerable<CeldaMatriz>
{
    /// <summary>
    /// Devuelve la cantidad de filas de la matriz
    /// </summary>
    int Filas { get; }

    /// <summary>
    /// Devuelve la cantidad de columnas de la matriz
    /// </summary>
    int Columnas { get; }

    /// <summary>
    /// Devuelve la cantidad de elementos distintos de 0
    /// </summary>
    int CantidadDeElementosNoNulos { get; }

    /// <summary>
    /// Inserta una celda en la matriz. Si los valores de fila o columna de la celda
    /// se salen de los límites especificados en el constructor el método debe lanzar
    /// una excepción del tipo IndexOutOfRangeException si  ya existía una
    /// celda en esa posición lo que se le hace es cambiar el valor. La inserción
    /// de un valor igual a 0 no debe adicionar una nueva celda, si la celda ya
    /// existía entonces deberá ser eliminada.
    /// </summary>
    /// <param name="celda">Nodo a insertar, provee fila, columna y valor</param>
    void Inserta(CeldaMatriz celda);

    /// <summary>
    /// Devuelve el valor en la fila,columna dada. Si los valores de fila o
    /// columna se salen de los límites especificados en el constructor
    /// el método debe lanzar una excepción del tipo IndexOutOfRangeException. Si
    /// nunca ha sido insertada una celda en dicha fila,columna se debe retornar
    /// el valor 0.
    /// </summary>
    /// <param name="fila">Fila donde se encuentra el valor</param>
    /// <param name="columna">Columna donde se encuentra el valor</param>
    /// <returns>Retorna el valor</returns>
    int ValorEn(int fila, int columna);

    /// <summary>
    /// Realiza la operacion de suma entre la matriz actual y la matriz que se
    /// recibe como parámetro, si la operación de suma provoca la inserción de
    /// valores nulos en la matriz, estas celdas deben ser excluidas, exactamente
    /// como cuando se hace en la inserción de un valor nulo
    /// </summary>
    /// <param name="otra">Matriz para sumar con la matriz actual. Esta matriz debe
    /// tener las mismas dimensiones que la matriz actual.</param>
    void Adiciona(IMatriz otra);

    /// <summary>
    /// Itera sobre los elementos de la fila especificada
```

```
    /// </summary>
    /// <param name="fila">Fila de la cual se quieren recuperar los elementos</param>
    /// <returns>Retorna la secuencia de elementos en la fila, el iterador debe
    /// incluir también los elementos iguales a 0</returns>
    IEnumerable<CeldaMatriz> Fila(int fila);

    /// <summary>
    /// Itera sobre los elementos de la columna especificada
    /// </summary>
    /// <param name="columna">Columna de la cual se quieren recuperar los elementos
    /// </param>
    /// <returns>Retorna la secuencia de elementos en la columna, el iterador debe
    /// incluir también los elementos iguales a 0</returns>
    IEnumerable<CeldaMatriz> Columna(int columna);
}
```

Problema

En el trabajo con matrices son comunes los problemas que involucran matrices de grandes dimensiones pero con realmente una pequeña cantidad de valores distintos de 0. La intención de este problema es no subutilizar la memoria usando un tradicional array bidimensional sino almacenar solamente estos valores diferentes de cero, pero preservando las operaciones que se pueden hacer con matrices. Usted debe realizar una implementación “inteligente” de la clase `Matriz` teniendo en cuenta lo especificado anteriormente.

```
public class Matriz:IMatriz
{
    /// <summary>
    /// Inicializa una instancia de la clase Matriz
    /// </summary>
    /// <param name="filas">Cantidad de filas de la matriz</param>
    /// <param name="columnas">Cantidad de columnas de la matriz</param>
    public Matriz(int filas, int columnas)
    {

    }

    /// <summary>
    /// Devuelve la cantidad de filas de la matriz
    /// </summary>
    public int Filas { get { } }

    /// <summary>
    /// Devuelve la cantidad de columnas de la matriz
    /// </summary>
    public int Columnas { get { } }

    /// <summary>
    /// Devuelve la cantidad de elementos distintos de 0
    /// </summary>
    int CantidadDeElementosNoNulos { get{} }

    /// <summary>
    /// Inserta una celda en la matriz. Si los valores de fila o columna de la celda
    /// se salen de los límites especificados en el constructor el método debe lanzar
    /// una excepción del tipo IndexOutOfRangeException si ya existía una
    /// celda en esa posición lo que se le hace es cambiar el valor. La inserción
    /// de un valor igual a 0 no debe adicionar una nueva celda, si la celda ya
```

```
    /// existía entonces deberá ser eliminada.
    /// </summary>
    /// <param name="celda">Celda a insertar, provee fila, columna y valor</param>
    public void Inserta(CeldaMatriz celda)
    {

    }

    /// <summary>
    /// Devuelve el valor en la fila,columna dada. Si los valores de fila o
    /// columna se salen de los límites especificados en el constructor
    /// el método debe lanzar una excepción del tipo IndexOutOfRangeException. Si
    /// nunca ha sido insertada una celda en dicha fila,columna se debe retornar
    /// el valor 0.
    /// </summary>
    /// <param name="fila">Fila donde se encuentra el valor</param>
    /// <param name="columna">Columna donde se encuentra el valor</param>
    /// <returns>Retorna el valor</returns>
    public int ValorEn(int fila, int columna)
    {

    }

    /// <summary>
    /// Realiza la operacion de suma entre la matriz actual y la matriz que se
    /// recibe como parámetro, si la operación de suma provoca la inserción de
    /// valores nulos en la matriz, estas celdas deben ser excluidas, exactamente
    /// como cuando se hace en la inserción de un valor nulo
    /// </summary>
    /// <param name="otra">Matriz para sumar con la matriz actual</param>
    public void Adiciona(IMatriz otra)
    {

    }

    /// <summary>
    /// Itera sobre los elementos de la fila especificada
    /// </summary>
    /// <param name="fila">Fila de la cual se quieren recuperar los elementos</param>
    /// <returns>Retorna la secuencia de elementos distintos de 0 en la fila
    /// </returns>
    public IEnumerable<CeldaMatriz> Fila(int fila)
    {

    }

    /// <summary>
    /// Itera sobre los elementos de la columna especificada
    /// </summary>
    /// <param name="columna">Columna de la cual se quieren recuperar los elementos
    /// </param>
    /// <returns>Retorna la secuencia de elementos distintos de 0 en la columna
    /// </returns>
    public IEnumerable<CeldaMatriz> Columna(int columna)
    {

    }

    /// <summary>
    /// Implementación de la Interfaz IEnumerable<CeldaMatriz>
    /// El objetivo es devolver todos los elementos no nulos de la matriz
    /// </summary>
```

```
    public IEnumerator<CeldaMatriz> GetEnumerator()
    {
        return celdas.GetEnumerator();
    }

    /// <summary>
    /// Implementación implícita de la interfaz IEnumerable
    /// </summary>
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}
```

ACLARACIONES:

- Su implementación debe contener un único constructor, con dos parámetros: filas y columnas. Dicho constructor será usado en la evaluación automática para crear instancias de esta clase. En caso de no contener este constructor todos los casos de prueba se considerarán fallidos.
- La matriz debe ocupar realmente memoria solo por los valores diferentes de 0.
- El objetivo del problema es no incurrir en gastos innecesarios de memoria, por lo que el uso de un `array` bidimensional para la implementación de este ejercicio sería incorrecto.
- Los iteradores `Fila` y `Columna` deben devolver todos los valores de la fila y la columna que sean distintos de 0.
- Los métodos `Inserta` y `ValorEn` deben aceptar cualquier combinación de fila, columna que esté dentro de los límites especificados en el constructor, en caso contrario se debe lanzar una excepción del tipo `IndexOutOfRangeException`
- Nótese que en método Adiciona el tipo del parámetro que es `IMatriz`. Sería incorrecto asumir que el tipo dinámico de este parámetro será necesariamente su propia implementación, por lo que deberá limitarse a utilizar en ese caso solamente los métodos definidos en la `interface`.
- Si la matriz recibida en el método adiciona no tiene las mismas dimensiones que la matriz actual se debe lanzar una excepción del tipo `InvalidOperationException`.
- Note que la interfaz `IMatriz` implementa a su vez la interfaz `IEnumerable<CeldaMatriz>`. El objetivo de este `IEnumerable` es devolver todos los elementos no nulos de la matriz. El orden en que son devueltos los elementos no es relevante.

Ejemplos

Ejemplo 1:

```
int cienMillones = 100000000;

//crea una matriz de tamaño 10^8 x 10^8
IMatriz matriz = new Matriz(cienMillones, cienMillones);

//Inserción de 3 valores aislados en la matriz
matriz.Inserta(new CeldaMatriz(0, 0, 10));
matriz.Inserta(new CeldaMatriz(1, 0, 20));
matriz.Inserta(new CeldaMatriz(cienMillones - 1, 0, 30));

Console.WriteLine(matriz.ValorEn(1,0)); //imprime 20
Console.WriteLine(matriz.ValorEn(cienMillones-1,0));//imprime 30
```

```
Console.WriteLine(matriz.ValorEn(5,5)); //imprime 0

Console.WriteLine("Imprimiendo valores en la fila 0:");
foreach (CeldaMatriz celda in matriz.Fila(0))
{
    Console.WriteLine(celda);
}
//Note que la fila 0 solamente contiene un valor distinto de 0
//por lo que se imprime:
//10 en (0;0)

Console.WriteLine("Imprimiendo valores en la columna 0:");
foreach (CeldaMatriz celda in matriz.Columna(0))
{
    Console.WriteLine(celda);
}
//Se imprime:
//10 en (0;0)
//20 en (1;0)
//30 en (99999999;0)

Console.WriteLine("Imprimiendo valores en la columna 5:");
foreach (CeldaMatriz celda in matriz.Columna(5))
{
    Console.WriteLine(celda);
}
//Note que la columna 5 no existen valores distintos de 0
//por lo que no se imprime nada

//eliminar elemento en la posición 1;0
matriz.Inserta(new CeldaMatriz(1,0,0));

Console.WriteLine("Imprimiendo valores en la columna 0:");
foreach (CeldaMatriz celda in matriz.Columna(0))
{
    Console.WriteLine(celda);
}
//Se imprime:
//10 en (0;0)
//30 en (99999999;0)


Console.WriteLine(matriz.ValorEn(cienMillones,cienMillones));
//lanza excepcion del tipo IndexOutOfRangeException
```

Ejemplo 2:

```
int milMillones = 1000000000;

//crea una matriz de tamaño 10^9 x 10^9
IMatriz matriz = new Matriz(milMillones, milMillones);
matriz.Inserta(new CeldaMatriz(0, 0, 10));
matriz.Inserta(new CeldaMatriz(1, 0, 20));
matriz.Inserta(new CeldaMatriz(2, 0, -40));
matriz.Inserta(new CeldaMatriz(milMillones - 1, 0, 30));

Console.WriteLine(matriz.CantidadDeElementosNoNulos); //imprime 4

IMatriz matriz2 = new Matriz(milMillones, milMillones);
matriz2.Inserta(new CeldaMatriz(0, 0, 10));
matriz2.Inserta(new CeldaMatriz(0, 1, 50));
matriz2.Inserta(new CeldaMatriz(1, 1, 20));
```

```
matriz2.Inserta(new CeldaMatriz(2, 0, 40));
matriz2.Inserta(new CeldaMatriz(milMillones - 1, milMillones - 1, 30));
Console.WriteLine(matriz2.CantidadDeElementosNoNulos); //imprime 5

Console.WriteLine();

matriz.Adiciona(matriz2);

//iteracion sobre todos los elementos no nulos de la matriz
foreach (CeldaMatriz celdaMatriz in matriz)
{
    Console.WriteLine(celdaMatriz);
}
//Se imprime (el orden de los elementos no es importante):
//20 en (1;0)
//30 en (999999999;0)
//20 en (0;0)
//50 en (0;1)
//20 en (1;1)
//30 en (999999999;999999999)

//notese como no se imprime la celda (2;0) producto de que fue
//anulada como resultado de la suma.

Console.WriteLine();

Console.WriteLine(matriz.CantidadDeElementosNoNulos); //imprime 6
```