Programación II
Tarea de Acumuladores booleanos

Introducción

En conjunto con el presente enunciado se les entregan las clases de testeo "AcumuladoresTest.java" desarrollada para JUnit 4.

Se deben implementar todos los métodos solicitados, usando acumuladores booleanos donde sea posible y necesario, y se debe verificar que todas las pruebas terminan sin errores, o bien, que todas las pruebas estén en verde.

Nota:

- Es muy importante que antes de programar se entienda el problema planteado y, si es posible, se divida en problemas más pequeños.
- Es muy importante que usen nombres declarativos para variables y métodos, y que el código esté bien organizado. Es decir, que sea fácil de leer.
- Recordar que cuando se habla de posiciones (fila 1, fila 2, columna 3) a la hora de programarlo se empieza a contar desde 0.
- Pasar los tests no garantiza que la implementación sea correcta, pero lo hacemos porque "*no pasar todos los test significa que la implementación no es correcta".*

Ejercicio 1

Implementar la función:

```
public boolean todosMultiplosEnAlgunaFila(int[][] mat, int num) { }
```

Que dado una matriz de enteros y un número, y que verifique si existe alguna fila donde todos sus elementos sean múltiplos del número recibido por parámetro.
Si la matriz está vacía o si el número no es positivo, devuelve falso.

Por ejemplo: para la matriz

2 4 3
6 3 9
6 2 7

Si le paso el número 3, devolverá verdadero porque en la fila del medio todos son múltiplos de 3.
Por otro lado, si le paso el número 2, devolverá falso porque no hay ninguna fila con todos sus elementos múltiplos de 2.

Ejercicio 2

Implementar la función:

```
public boolean hayInterseccionPorFila(int[][] mat1, int[][]mat2) { }
```

Que dado 2 matrices se verifica si hay intersección entre las filas de cada matriz, fila a fila.
Si las matrices tienen distinta cantidad de filas o si alguna matriz está vacía, devuelve falso.

Por ejemplo: para las siguientes matrices

mat1	**mat2**	**mat3**	**mat4**
1 2 3 4	4 5 6 7 8	3 8 9 0	1 2 3 4
3 4 5 6	7 8 9 3 5	1 2 7 8	3 4 5 6
7 8 9 0	2 4 7 1 2	1 2 3 4	

Si pasamos **mat1** y **mat2**, devolverá verdadero porque para cada fila la intersección es: {4}, {3,5} y {7}.
Si pasamos **mat1** y **mat3**, devolverá falso porque las filas 2 y 3 no tienen intersección.
Si pasamos **mat1** y **mat4**, devolverá falso porque no tienen la misma cantidad de filas.

Ejercicio 3

Implementar la función:

```
public boolean algunaFilaSumaMasQueLaColumna(int[][] mat, int nColum) { }
```

Que dado una matriz y el índice de una columna, se verifica si existe alguna fila cuya suma de todos sus elementos sea mayor estricto que la suma de todos los elementos de la columna indicada por parámetro.
Si el índice de la columna es inválido o la matriz está vacía, devuelve falso.

Por ejemplo: para la matriz

2 4 3
5 8 9
6 2 12

Si le pasamos la columna 0 (**en java los índices de fila y columna comienzan en 0**), nos devolverá Verdadero porque la columna { 2, 5, 6} suma 13 y tanto la fila 1 como la fila 2 suma más que 13.
Sin embargo, si le pasamos la columna 2 como parámetro, nos devolverá Falso porque la columna {3, 9, 12} suma 24 y las filas suman 9, 22 y 18 respectivamente. Ninguna fila suma más que 24.

Ejercicio 4

Implementar la función:

```
public boolean hayInterseccionPorColumna(int[][] mat1, int[][]mat2) { }
```

Que dado 2 matrices se verifica si hay intersección entre las columnas de cada matriz, columna a columna.
Si las matrices tienen distinta cantidad de columnas o alguna matriz está vacía, devuelve falso.

Por ejemplo: para las siguientes matrices

mat1

1	2	3	4
3	4	5	6
7	8	9	0

mat2

4	5	6	0
7	8	9	3
1	4	7	1

mat3

3	8	6	0
1	2	7	8
2	1	4	5

mat4

1	2	3
3	4	5
6	7	8

Si pasamos **mat1** y **mat2**, devolverá verdadero porque para cada columna la intersección es: {1,7}, {4,8}, {9} y {0}.
Si pasamos **mat1** y **mat3**, devolverá falso porque la columna 3 no tienen intersección.
Si pasamos **mat1** y **mat4**, devolverá falso porque no tienen la misma cantidad de columnas.