Ejercicio Extra Tipo Recuperatorio

Un dispositivo que toma imágenes de un fenómeno natural está corriendo sobre una plataforma tecnológica con recursos limitados. Cada imagen (representada por una matriz de NxM) está compuesta por píxeles con valores entre 0 y 255. Se tiene implementado un algoritmo de compresión que comprime aquellas porciones de la imagen distintas del color negro (0 en la escala de valores del pixel). Dicho algoritmo procede de la siguiente manera: por cada una de las filas de la matriz, toma cada secuencia delimitada por uno o mas pixeles de color negro (valor 0) con más de X repeticiones de un valor de píxel (para ser comprimida todos los elementos de la secuencia deben ser iguales), comprime la secuencia poniendo en la primera posición el valor negado de la cantidad de ocurrencias y a continuación el valor del pixel que se repite. Cada fila de la matriz empieza y termina con uno o más pixeles negros.

Se pide implementar el algoritmo de descompresión que restablezca la matriz original. Asumir que cada fila posee suficientes lugares como para realizar la descompresión.

Ejemplo de matriz comprimida con X = 3:

0	-8	67	0	14	0	-4	33	0	5	98	0	0	0	0	0	0	0	0	0
0	0	25	25	0	-5	3	0	25	44	44	0	-4	1	0	0	0	0	0	0
0	44	44	44	0	-7	15	0	-4	9	0	12	0	0	0	0	0	0	0	0

La matriz resultante quedaría de la siguiente forma:

0	67	67	67	67	67	67	67	67	0	14	0	33	33	33	33	0	5	98	0
0	0	25	25	0	3	3	3	3	3	0	25	44	44	0	1	1	1	1	0
0	44	44	44	0	15	15	15	15	15	15	15	0	9	9	9	9	0	12	0

```
public class extraRecuperatorio {
  public final static int N = 3;
  public final static int M = 20;
  public final static int SEP = 0;
  public static void main(String[] args) {
    int[][] matriz = {{0,-8,67, 0,14, 0,-4,33, 0, 5,98, 0, 0,0,0,0,0,0,0,0},
                      {0, 0,25,25, 0,-5, 3, 0,25,44,44, 0,-4,1,0,0,0,0,0,0},
                      {0,44,44,44, 0,-7,15, 0,-4, 9, 0,12, 0,0,0,0,0,0,0,0}};
    for (int i = 0; i < N; i++)
      descomprimirFila(matriz[i]);
    mostrar(matriz);
  }
  public static void descomprimirFila(int[] arr) {
    int ini = 0;
    int fin = -1;
    int cantidad = 0;
    while (ini < M) {
      ini = buscarInicio(arr, fin + 1);
      if (ini < M) {
        fin = buscarFin(arr, ini);
        if (estaComprimida(arr, ini)) {
          cantidad = descomprimir(arr, ini, fin);
          fin = fin + cantidad - 2;
        }
      }
    }
  }
  // Podría no retornar la cantidad
  public static int descomprimir(int[] arr, int ini, int fin) {
    int cantidad = -arr[ini];
    arr[ini] = arr[ini + 1];
    for (int i = 0; i < cantidad - 2; i++) {
      correrADerecha(arr, ini);
      // Ya duplica ini, no hace falta setearlo
    }
    return cantidad;
  }
  public static void correrADerecha(int[] arr, int ini) {
    for (int pos = M - 1; pos > ini; pos--)
      arr[pos] = arr[pos - 1];
  }
  // Podría no modularizarse
  public static boolean estaComprimida(int[] arr, int ini) {
    return arr[ini] < 0;
  }
  public static int buscarFin(int[] arr, int pos) {
    while (pos < M && arr[pos] != SEP)
      pos++;
```

```
        return pos - 1;
    }
    public static int buscarInicio(int[] arr, int pos) {
        while (pos < M && arr[pos] == SEP)
            pos++;
        return pos;
    }
    public static void mostrar(int[][] matriz) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++)
                System.out.print(matriz[i][j] + "|");
            System.out.println();
        }
    }
}
```