

## Sección 1

Grupo: PF 03

Meili Vanegas Hernández - 201225567

Jairo Iván Bernal Acosta -

### 1. Algoritmo de solución

El algoritmo propuesto calcula el área del cuadrado y el rectángulo más grande compuesto por unos dada una matriz  $M_{m \times n}$  de unos y ceros. Ambos cálculos se llevan a cabo por separado; para el primero, se construye una matriz  $MC_{m \times n}$  en la que en la posición  $MC_{ij}$  representa la dimensión del cuadrado, siendo esa posición, el vértice superior izquierdo. Este proceso se obtiene al proponer los siguientes 3 condicionales:

- Si la posición  $MC_{ij}$  está ubicada en un borde inferior o derecho de la matriz, el valor es igual al de la misma posición en la matriz original  $M_{ij}$ .
- Si el valor en la posición de la matriz original  $M_{ij}$  es 0, el valor en la misma posición de la matriz alterada  $MC_{ij}$  es el mismo.
- Si no cumple con ninguna de las anteriores condiciones, el valor en la matriz  $MC_{ij}$  corresponde al mínimo valor entre las posiciones diagonal anterior, izquierda o debajo de la posición actual en la matriz alterada.

A la vez que se va calculando la matriz  $MC$ , se va guardando el máximo en una variable entera. Esta es la que al final, al multiplicarse por sí misma, se va a retornar como el área del cuadrado máximo en la matriz.

Por otro lado, para calcular el área del rectángulo máximo, se calcula una matriz adicional  $MR_{m \times n}$ , la cual calcula la distancia vertical, iniciando desde debajo del primer cero en la matriz. En seguida, esta matriz se recorre por filas, y por cada una de éstas, se calcula el área del rectángulo más grande, tomando los arreglos como un histograma.

### 2. Análisis de complejidades espaciales y temporales

Teniendo en cuenta la descripción hecha anteriormente del algoritmo propuesto, se puede establecer que la complejidad temporal, dadas las variables  $m$  y  $n$  (tamaño de la matriz inicial) es la siguiente:

$$T(m, n) = \text{Recorrer Toda la Matriz} + \text{Recorrer Por Filas } MR \cdot P(CMR)$$

Donde en *Recorrer Toda la Matriz* se encarga de construir las matrices adicionales (una para el procedimiento de cuadrados y otra para el procedimiento de rectángulos) y de calcular a su vez el área del cuadrado máximo. Por otro lado, el *Recorrer Por Filas MR* se encarga de recorrer las filas de  $MR$ , las cuales son  $m$  en total. Por último,  $P(CMC)$  se encarga de calcular el área del rectángulo máximo dado un histograma (cada fila de la matriz  $MR$ ), este procedimiento busca agregar los elementos del “histograma” en forma ascendente, calculando los límites que tienen

cada uno de los rectángulos correspondientes a la altura de esa barra, siendo así, la complejidad de este procedimiento es de  $O(n)$ . Por lo anterior, la complejidad del algoritmo propuesto es la siguiente:

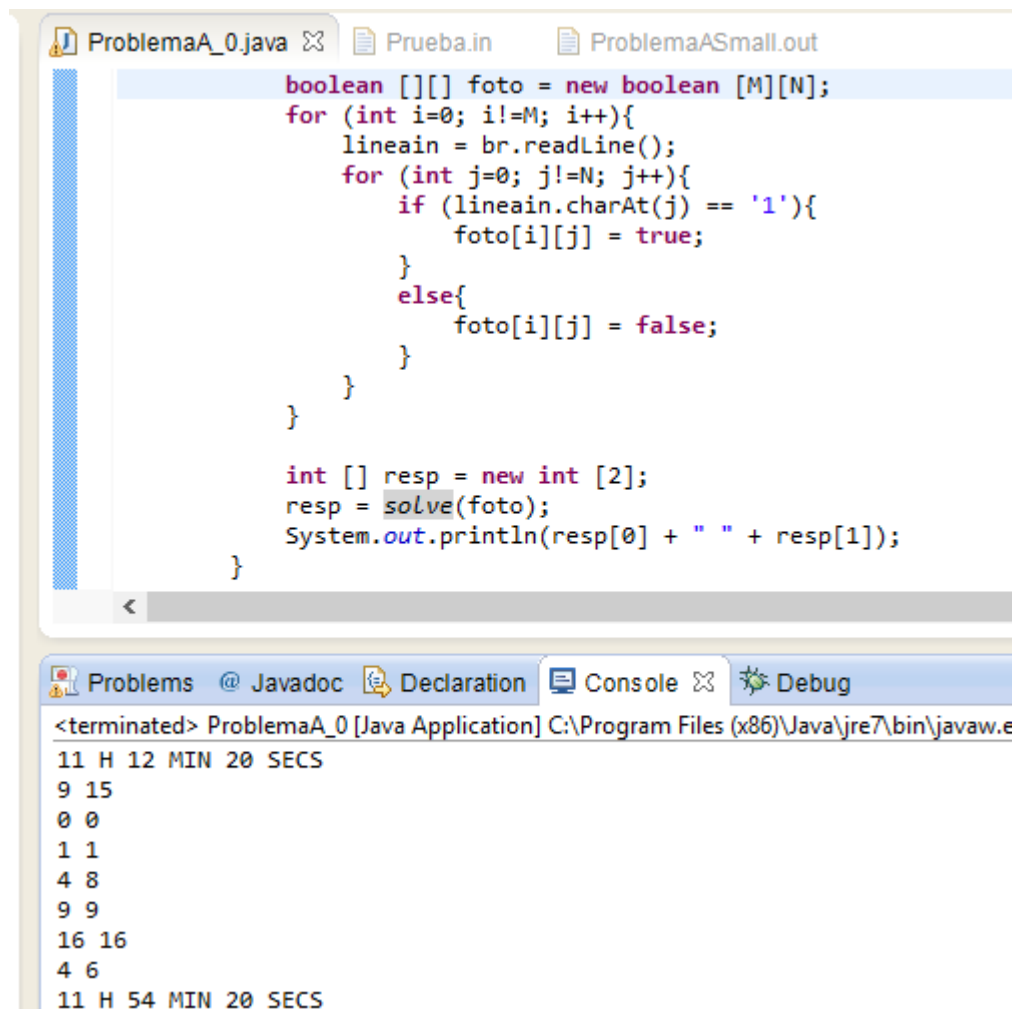
$$T(m, n) = O(m \cdot n) + m \cdot O(n) = O(m \cdot n)$$

Por otro lado, se sabe que la complejidad espacial es de  $S(m, n) = 2 \cdot O(m \cdot n)$ , dado que se deben construir dos matrices adicionales, con el mismo tamaño de la original.

Al comparar la solución ingenua con la propuesta, se puede notar que las diferencias son sustanciales. Un ejemplo claro de lo anterior es el siguiente:

Se lleva a cabo una prueba big, en la que  $m = 1000$  y  $n = 1000$ . Para hacer notar la complejidad temporal que tiene cada uno de los algoritmos (ingenuo y propuesto) se lleva a cabo un `System.out.printf` de la hora, minuto y segundo al empezar el algoritmo y después de ejecutarlo. Los resultados son los siguientes:

#### Solución INGENUA Problema A: Prueba BIG



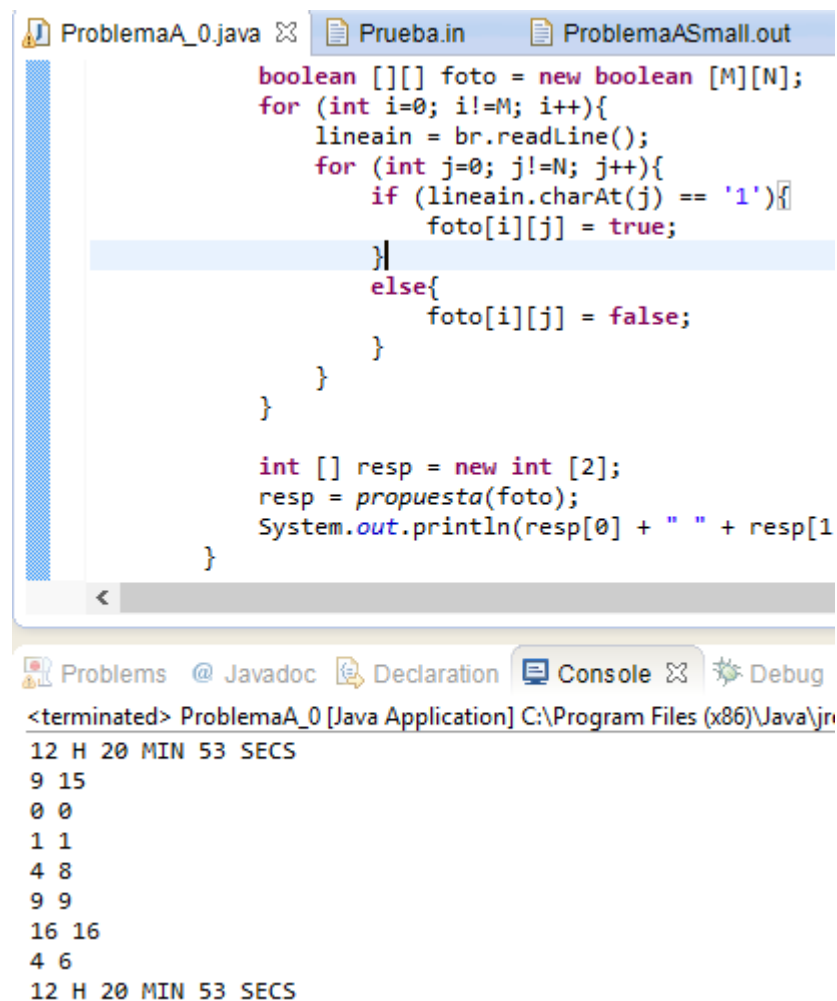
```
ProblemaA_0.java Prueba.in ProblemaASmall.out
boolean [][] foto = new boolean [M][N];
for (int i=0; i!=M; i++){
    lineain = br.readLine();
    for (int j=0; j!=N; j++){
        if (lineain.charAt(j) == '1'){
            foto[i][j] = true;
        }
        else{
            foto[i][j] = false;
        }
    }
}

int [] resp = new int [2];
resp = solve(foto);
System.out.println(resp[0] + " " + resp[1]);
}
```

```
Problems @ Javadoc Declaration Console Debug
<terminated> ProblemaA_0 [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.e
11 H 12 MIN 20 SECS
9 15
0 0
1 1
4 8
9 9
16 16
4 6
11 H 54 MIN 20 SECS
```

Se puede observar que el algoritmo tomó 42 minutos para terminar satisfactoriamente.

### Solución PROPUESTA Problema A: Prueba BIG



```
ProblemaA_0.java Prueba.in ProblemaASmall.out
boolean [][] foto = new boolean [M][N];
for (int i=0; i!=M; i++){
    lineain = br.readLine();
    for (int j=0; j!=N; j++){
        if (lineain.charAt(j) == '1'){
            foto[i][j] = true;
        }
        else{
            foto[i][j] = false;
        }
    }
}

int [] resp = new int [2];
resp = propuesta(foto);
System.out.println(resp[0] + " " + resp[1])
}
```

Problems @ Javadoc Declaration Console Debug

<terminated> ProblemaA\_0 [Java Application] C:\Program Files (x86)\Java\jre

12 H 20 MIN 53 SECS

9 15

0 0

1 1

4 8

9 9

16 16

4 6

12 H 20 MIN 53 SECS

Se puede observar que el algoritmo tomó menos de 1 segundo para terminar satisfactoriamente.

### 3. Comentarios Finales

Se puede concluir que el algoritmo propuesto tiene una complejidad temporal polinomial.