# Descripción de solución Problema A
# Sección: 01

Sebastián Valencia Calderón 201111578
Laura Ávila Bermudez 201212736
Grupo 08

25 de mayo de 2015

1. **Algoritmo de solución**. El algoritmo de solución, se divide en dos partes, el cómputo del cuadrado de área máxima dentro de la entrada, y el cómputo del rectángulo de área máxima dentro del mismo. Para el primero, se usó programación dinámica, el algoritmo, usa espacio adicional para llevar cuanta de los lados máximos encontrados hasta ahora. Por ejemplo, si se considera el arreglo rectángular de bits mostrado, el espacio necesario, con su configuración, semuestra a constinuación, además, la explicación de cada item de la nueva matriz:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 2 & 0 \end{bmatrix}$$

   En ésta nueva matriz, los bordes superior e izquierdo, así como su intersección, tienen el valor de 0. Mientras tanto, la submatriz restante, tiene un cero 0 si en ésa misma posición ajustada, la matriz original lo tiene, de lo contrario (la posición ajustada de la matriz original es un uno), el elemento de ésa posición, se obtiene a partir de la fórmula:

$$\text{mín}\left(A_{i-1,j}, A_{i-1,j-1}, A_{i,j-1}\right) + 1$$

   En la fórmula, $A$ es la nueva matriz. La entrada máxima de ésta, corresponde al lado del cuadrado más grande dentro de la matriz original. La igualdad de cada elemento de la matriz se muestra a continuación ($M$ es la matriz original):

$$A_{i,j} = \begin{cases} 0 & \text{if } i = 0 \vee j = 0 \vee B_{i-1,j-1} = 0 \\ \text{mín}\left(A_{i-1,j}, A_{i-1,j-1}, A_{i,j-1}\right) + 1 & \text{otherwise} \end{cases}$$

   Para hallar el rectángulo, se cuentan los unos arriba de cada fila de la matriz, si es un uno, sino, se deja cero, a partir de éste arreglo, se halla el rectángulo más grande dentro de éste histograma. Asimismo, se acumula el máximo. Es decir, para cada fila de la matriz original, se genera un arreglo que cuente para cada posición de la fila, los unos por encima de ésta entrada en caso de que la misma sea un uno. De lo contrario,

se coloca cero. éste nuevo arreglo, se pasa para calcular el rectángulo de área máxima dentro de éste histograma.

2. **Análisis temporal y espacial de la solución**. El tamaño del problema, es la tupla $M, N$. La complejidad espacial de la solución ingenua, es $O(1)$. La complejidad temporal, se deduce de manera anáilitica, contando el número de ejecución de cada inStrucción. Para ésto, se cuenta con la siguiente expresión, donde como operaciones de costo $\Theta(1)$, se consideraron, la suma, la resta, la multiplicación, la división, obtener la longitud de un arreglo, indexar un arreglo, asignar, y comparar átomos de lenguaje.

$$T_t(M,N) = \sum_{i=0}^{M}\left(\sum_{j=0}^{N}\left[\Theta(1) + \sum_{p1=0}^{i+1}\sum_{p2=p1}^{i+1}\sum_{q1=0}^{j+1}\sum_{q2=q1}^{j+1}\left[\Theta(1) + \sum_{r=p1}^{p2+1}\Theta(1) + \Theta(1)\right] + \Theta(1)\right]\right)$$

$$T_t(M,N) = \sum_{i=0}^{M}\sum_{j=0}^{N}\left(\left(\sum_{p_1=0}^{i+1}\sum_{p_2=p_1}^{i+1}(p_2 - p_1 + 4)\right)\sum_{q_1=0}^{j+1}(-q_1 + j + 2) + 2\right)$$

$$T_t(M,N) = \frac{M^4\,N^3 + 2\,M^3\,N^3 + M^2\,N^3}{144} + \frac{2\,M^3\,N^3 + 3\,M^2\,N^3 + M\,N^3}{12} + \ldots$$

$$T_t(M,N) = O(M^4\,N^3)$$

Para la solución propuesta, se tiene, la expresión:

$$T_t(M,N) = 2\sum_{i=0}^{M}\left[\sum_{j=0}^{N}\Theta(1)\right] + \Theta(1) + 2\sum_{i=0}^{N}\Theta(1) + \Theta(1)$$

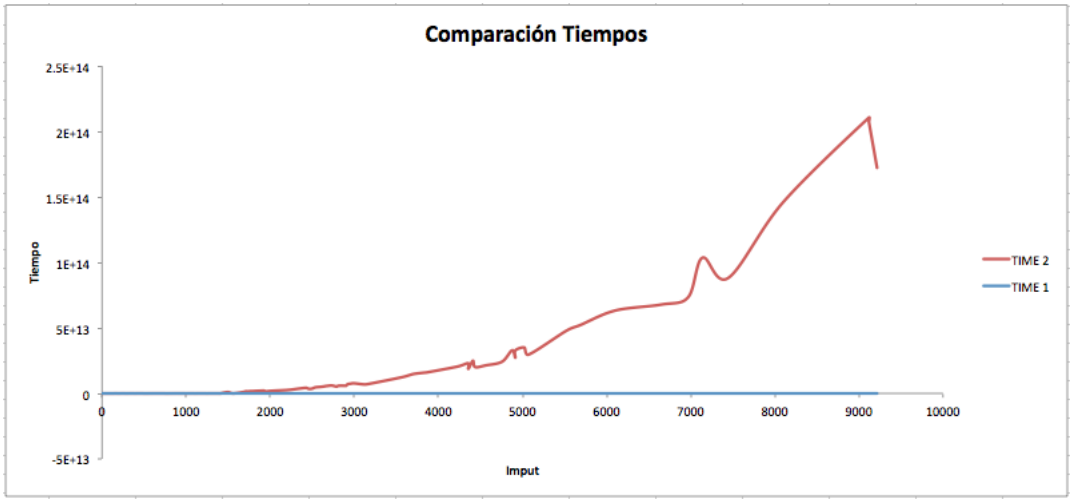
$$T_t(M,N) = 2\ (M+1)\ (N+1) + 2\ (N+1) + 2 = O(MN)$$

Para una comprabación empírica de ésto, se generó un conjunto de entradas aleatorias para todos los tamaños (small, medium, big). El programa de generación, se muestra a continuación:

```
import random

SMALL = 10
MEDIUM = 100
BIG = 1000

def write_set(file_object, lst, bound):
    i = 0
    while i < bound:
        M, N = random.choice(lst), random.choice(lst)

        file_object.write(str(M) + ' ' + str(N) + '\n')
```

```
13
14          j = 0
15          print M, N
16          while j < M:
17              string = ''.join(random.SystemRandom().choice(['0', '1']) for
    _ in range(N))
18              file_object.write(string + '\n')
19              j += 1
20
21          i += 1
22
23  def generate_file(bound):
24      file_name = '../data/ProblemaAGeneral.in'
25      file_object = open(file_name, 'w')
26
27      small_list = [_ for _ in range(1, SMALL + 1)]
28      medium_list = [_ for _ in range(1, MEDIUM + 1)]
29      big_list = [_ for _ in range(1, BIG + 1)]
30
31      write_set(file_object, small_list, bound)
32      write_set(file_object, medium_list, bound)
33      write_set(file_object, big_list, bound)
34
35      file_object.write(str(0) + ' ' + str(0) + '\n')
36
37
38  generate_file(100)
```

A partir de las entradas generadas, se corren los programas y se mide su tiempo según la metodología de Bob Sedgewick. A continuación, se grafican los tiempos, si se asume que la entrada es el producto $MN$, ésto para evitar usar gráficas en tercera dimensión. En la gráfica, la linea roja es el tiempo de la solución ingenua, mientras la azul, la solución planteada.Además, puede verse que la solución planteada, es $M^4N^2/MN = M^3N^2$ veces mejor que la ingenua.




3. **Comentarios finales** En la solución se observa un desempeño dependiente linealmente del producto de las dimensiones de las entradas. Además, se percibe una simplificación

muy severa en el uso de la notación $O$, pues la complejidad de la solución ingenua, desprecia varios términos del resultado de la sumatoria. Si $M = N$, la solucion ingenua es $O(M^7)$, mientras la propuesta es $O(M^2)$, ambas soluciones perteneces a la clase P de complejidad. Sin embargo, al hacer la prueba sobre 100 datos de cada tipo de entrada, se percibe que la solución propuesta no demora casi nada, mientras la otra, demora mucho más de una hora y media.