

# Tarea 8

Saul Ivan Rivas Vega

Diseño y análisis de algoritmos

Equipo Completo:

Yadira Fleitas Toranzo

Diego de Jesús Isla Lopez

Saul Ivan Rivas Vega

22 de noviembre de 2019

## 1. Ejercicio 1.

Considera el siguiente problema de *formato de textos*. Como entrada tenemos un arreglo  $W[1, \dots, n]$  tal que cada  $W[i]$  es una palabra; supondremos que el último símbolo en  $W[i]$  es un espacio en blanco. También recibimos un entero **long\_linea**  $> 0$  como parte de la entrada, tal que **long\_linea** es mayor o igual a la longitud de cada palabra  $W[i]$ , denotada  $|W[i]|$ . Lo que buscamos es dar *formato* al texto en  $W$  para esto hay que decidir cuales palabras van juntas en la misma línea. Si decidimos poner en una misma línea las palabras  $W[i], \dots, W[i']$ ,  $i \leq i'$ , lo que se denota  $l[i, i']$ , entonces la *penalización* de  $l[i, i']$  es:

- $\infty$  si **long\_linea**  $< |W[i]| + \dots + |W[i']|$  (es decir, las palabras no caben en una sola línea);
- de otra forma,  $(\mathbf{long\_linea} - (|W[i]| + \dots + |W[i']|))^3$

Entonces, la *penalización* de un *formato*  $l_1[1, i_1], l_2[i_1 + 1, i_2], \dots, l_j[i_{j-1} + 1, n]$  de  $W$  es la suma de las penalizaciones de sus líneas.

### 1.1. a) Demuestra que la siguiente estrategia *greedy* produce *formatos* arbitrariamente malos, es decir, con penalizaciones arbitrariamente grandes

Procesamos las palabras de  $W[1]$  a  $W[n]$ ; si aún hay espacio suficiente para meter  $W[i]$  en la línea actual, entonces la metemos; de otra forma iniciamos una nueva línea en la que ponemos a  $W[i]$ .

**Demostración.** Podemos demostrarlo dando un ejemplo donde esto sucede. Sea  $n = 3$ , **long\_linea** = 6 y con las longitudes  $|W[1]| = 3$ ,  $|W[2]| = 3$  y  $|W[3]| = 2$ .

- El algoritmo tomará la línea actual con espacio disponible de 6 y procesará la primera palabra,  $W[1]$ .
- Como el espacio disponible es 6 y  $|W[1]| = 3$  entonces la metemos en la línea actual.
- Ahora el espacio disponible es 3 y como  $|W[2]| = 3$  entonces la metemos en la línea actual.
- Como el espacio disponible es 0 y  $|W[3]| = 2$  creamos una nueva línea y metemos a  $W[3]$  en ella.

Al terminar el algoritmo terminamos con un formato  $f_1$  con las líneas  $l_1[1, 2]$  y la línea  $l_2[3, 3]$ . Veamos sus *penalizaciones*.

En el caso de  $l_1[1, 2]$ ,  $|W[1]| + |W[2]|$  es igual a *long\_linea* entonces entra en el segundo caso de la función de *penalización* entonces la calculamos con:

$$\begin{aligned}
 penalizacion(l_1[1, 2]) &= (long\_linea - (|W_1| + |W_2|))^3 \\
 &= (long\_linea - (3 + 3))^3 \\
 &= (long\_linea - 6)^3 \\
 &= (6 - 6)^3 \\
 &= (0)^3 \\
 &= 0
 \end{aligned} \tag{1}$$

Para  $l_2[3, 3]$ ,  $|W[3]|$  es menor a *long\_linea* entonces entra en el segundo caso de la función de *penalización* entonces la calculamos con:

$$\begin{aligned}
 penalizacion(l_2[3, 3]) &= (long\_linea - (|W_3|))^3 \\
 &= (long\_linea - (2))^3 \\
 &= (long\_linea - 2)^3 \\
 &= (6 - 2)^3 \\
 &= (4)^3 \\
 &= 64
 \end{aligned} \tag{2}$$

Ahora la *penalización* del formato  $f_1$  es la suma de las dos penalizaciones de sus líneas:

$$penalización\ f_1 = 64 + 0 = 64 \tag{3}$$

Sin embargo, tomemos el siguiente formato  $f_2$  con las líneas,  $l_1[1, 1]$  y  $l_2[2, 3]$ . Para  $l_1[1, 1]$  tenemos:

$$\begin{aligned}
 penalizacion(l_1[1, 1]) &= (long\_linea - (|W_1|))^3 \\
 &= (long\_linea - (3))^3 \\
 &= (long\_linea - 3)^3 \\
 &= (6 - 3)^3 \\
 &= (3)^3 \\
 &= 27
 \end{aligned} \tag{4}$$

Y para  $l_2[2, 3]$  tenemos:

$$\begin{aligned}
 penalizacion(l_2[2, 3]) &= (long\_linea - (|W_2| + |W_3|))^3 \\
 &= (long\_linea - (3 + 2))^3 \\
 &= (long\_linea - 5)^3 \\
 &= (6 - 5)^3 \\
 &= (1)^3 \\
 &= 1
 \end{aligned} \tag{5}$$

Por lo tanto la *penalización* total para el formato  $f_2$  es:

$$\text{penalización } f_2 = 27 + 1 = 28 \quad (6)$$

Finalmente como  $f_1$  es mayor que  $f_2$  y además podemos formar arreglos agregando las mismas 3 longitudes del ejemplo en ese orden decimos que el algoritmo produce *formatos* con penalizaciones arbitrariamente grandes.

- 1.2. b) Usa la técnica de programación dinámica para diseñar un algoritmo que encuentre un formato con penalización mínima. Demuestra la correctitud de tu algoritmo y haz un análisis de tiempo y espacio.
- 1.3. c) Modifica tu algoritmo para que funcione con una función de penalización de línea dada; la penalización de un formato sigue siendo la suma de las penalizaciones de sus líneas.

## 2. Ejercicio 2

Considera un arreglo  $A[1, \dots, n]$  con enteros positivos en sus entradas. Decimos que una pareja  $(i, j)$  es un *declive* si  $i \leq j$  y  $A[i] \geq A[j]$ ; la *longitud* de  $(i, j)$  es  $A[i] - A[j]$ . Diseña un algoritmo de tiempo y espacio  $o(n^2)$  que calcule un declive de  $A$  de longitud máxima (es o pequeña, investiga ese concepto). Demuestra que tu algoritmo es correcto y haz el análisis de tiempo y espacio.

**Data:** Arreglo  $A$ .

**Result:** Tupla  $(max\_long, r\_i, r\_j)$  Que representa el valor de la longitud máxima de un declive en  $A$  y los índices  $r\_i, r\_j$  de dicho declive  $(r\_i, r\_j)$ .

```
/* Inicializamos una cola de prioridad donde guardaremos tuplas  $(A[i], i)$  donde
   ordenaremos mayor a menor por  $A[i]$ . */
1  $Q = \{\}$ ;
   /* Inicializamos las respuestas actuales. */
2  $max\_long = -\infty$ ;
3  $r\_i = 0$ ;
4  $r\_j = 0$ ;
   /* Recorremos del final al inicio el arreglo  $A$ , pasando por todas las posiciones
   desde  $n$  hasta 1. */
5 for  $i = n, i \geq 1$  do
   /* Insertamos el elemento actual a la cola de prioridad. */
6    $(A[i], i) \rightarrow Q$ ;
   /* Obtenemos el elemento al frente de  $Q$ , es decir la tupla  $(A[i], i)$  con menor
    $A[i]$ , no lo removemos de la cola de prioridad solo obtenemos sus valores. */
7    $(valor, indice) \leftarrow Q$ ;
   /* Como procesamos de fin a inicio este elemento cumple que  $i \leq indice$ , ahora
   entonces revisamos si  $A[i] \geq valor$  lo cual sería un declive  $(i, indice)$ . */
8   if  $A[i] \geq valor$  then
   /* Calculamos la longitud del declive  $(i, indice)$ . */
9      $current\_long = A[i] - valor$ ;
   /* Actualizamos nuestra respuesta si es mayor a la que llevabamos. */
10    if  $current\_long > max\_long$  then
11       $max\_long = current\_long$ ;
12       $r\_i = i$ ;
13       $r\_j = indice$ ;
14    end
15  end
16 end
17 return  $(max\_long, r\_i, r\_j)$ ;
```

**Algorithm 1:** Algoritmo MAXDECLIVE.

### 3. Ejercicio 3

Considera el problema de selección de centros visto en clase. Demuestra que el siguiente algoritmo devuelve un conjunto  $C$  con a lo más  $k$  centros tal que  $rc(C) \leq 2rc(C^*)$ , donde  $C^*$  es un conjunto con a lo más  $k$  centros óptimo, es decir con radio de cobertura mínimo. Puedes suponer como correctos todos los algoritmos y afirmaciones vistas en clase.

```

Alg. SelecCentros(S, k)
  If |S| <= k then
    return S
  Else
    Inicializa C con cualquier elemento s de S
    While |C| < k do
      Elegir cualquier elemento s en S que maximice dist(s,C)
      Agregar s a C
    endwhile
    return C

```

## 4. Ejercicio 4

Escribe una versión recursiva del algoritmo probabilístico de corte mínimo visto en clase (basado en contracciones de aristas). Demuestra que el conjunto de aristas devuelto por el algoritmo es efectivamente un corte de la gráfica inicial.

## 5. Ejercicio 5

Tenemos  $n$  servidores que buscan coordinarse para ejecutar localmente la misma acción. De forma *abstracta*, los servidores llegan a un *consenso* sobre un bit, y ejecutan localmente la acción asociada. Por ejemplo, si el bit consensuado es 0, cada servidor hace *rollback* en su base de datos local, pero si el bit consensuado es 1, cada servidor hace *commit* localmente. Considera el siguiente algoritmo probabilístico para este problema:

```

Alg. ConsensoBinario
  r = 0
  While TRUE do
    1. r = r+1
    2. Cada servidor obtiene un bit aleatorio b_r con probabilidad uniforme
    3. Cada servidor comunica su bit b_r a todos los servidores
    4. Si todos los bits b_r de los n servidores son iguales, cada servidor
       ejecuta localmente la tarea correspondiente y termina
    5. De otra forma, continua
  endwhile

```

Responde a lo siguiente:

1. Demuestra que el número esperado de iteraciones para ejecutar la acción es  $O(2^n)$ .  
Tip: Modela el problema con una variable aleatoria con distribución geométrica.
2. ¿Cuál es el número esperado de iteraciones si en cada una de ellas los servidores obtienen su  $r$ -ésimo bit,  $b_r$ , llamando una función **shared\_random\_bit**( $r$ ) que devuelve

el mismo  $r$ -ésimo bit a todos los servidores con probabilidad  $p$ , para alguna constante  $0 < p < 1$ ?