# Auxiliar 1 - "Cotas Inferiores"

Profesor: Gonzalo Navarro
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

14 de Agosto del 2017

## Teoría de la Información

**P1.** Decimos que un arreglo $A[1 \ldots n]$ está $k$**-ordenado** si puede ser dividido en $k$ bloques, cada uno de tamaño $\frac{n}{k}$ (asumimos que $\frac{n}{k}$ es un entero), de modo que los elementos de cada bloque son mayores que elementos de bloques anteriores. Los elementos en cada bloque no necesitan estar ordenados.

a) Describa un algoritmo que "$k$-ordene" un arreglo arbitrario en $\mathcal{O}(n \log k)$.

b) Muestre que cualquier algoritmo basado en comparaciones que $k$-ordene, requiere $\Omega(n \log k)$ en el peor caso.

c) Describa un algoritmo que ordene un arreglo $k$-ordenado en $\mathcal{O}\left(n \log\left(\frac{n}{k}\right)\right)$.

d) Muestre que cualquier algoritmo basado en comparaciones que ordene un arreglo $k$-ordenado requiere $\Omega\left(n \log\left(\frac{n}{k}\right)\right)$.

**P2.** Supongamos que tenemos las probabilidades $\{p_1, \ldots, p_n\}$, tales que $\sum_{i=1}^{n} p_i = 1$. El **algoritmo de Huffman** construye un árbol binario que tiene estas probabilidades en sus hojas y minimiza $\sum_{i=1}^{n} p_i n_i$, donde $n_i$ es la profundidad de $p_i$ en el árbol.

a) Suponga que las probabilidades vienen ordenadas de forma creciente. ¿Cómo implementaría el algoritmo para que tuviera costo total $\mathcal{O}(n)$ y costo $\mathcal{O}(1)$ por operación?

b) Demuestre la correctitud del algoritmo de Huffman.

## Adversario

**P3.** Considere una propiedad $P$ de los grafos no dirigidos sin loops; por ejemplo, conectividad, aciclicidad, no-planaridad. Sea $G = (V, E)$ un grafo no completo; es decir, no todo par en $V \times V$ corresponde a un arco en E. Decimos que $G$ es crítico para $P$ si $G$ no tiene la propiedad $P$, pero al agregarle cualquier arco en $(V \times V) \setminus E$ obtenemos un grafo $G'$ que satisface $P$. Definimos $x := |(V \times V) \setminus E|$.
Sea $\mathcal{H}$ el conjunto de los grafos que utilizan el mismo conjunto de vértices $V$ que un grafo crítico $G$ para $P$. Considere un algoritmo que verifica si un grafo $H \in \mathcal{H}$ tiene la propiedad $P$ utilizando solo preguntas de la forma: ¿existe un arco entre los vértices $u$ y $v$?.
Muestre que este algoritmo debe realizar a lo menos $x$ preguntas.

"It was a game called Yes and No, where Scrooge's nephew had to think of something, and the rest must find out what; he only answering to their questions yes or no, as the case was..."
A Christmas Carol, Charles Dickens

**P4.** Muestre que el problema de encontrar el máximo y el segundo máximo, en un modelo basado en comparaciones, toma al menos $n + \lceil \log n \rceil - 2$.

## Reducción

**P5.** El problema $3SUM$ consiste encontrar 3 números diferentes (de un conjunto de $n$ números) cuya suma sea 0.

a) Muestre que el problema de 3 **puntos colineales** (encontrar 3 puntos colineales en un conjunto de puntos) es $3SUM - hard$.

b) Muestre que el problema de 3 **en una línea** (encontrar si hay un punto común a 3 líneas en un conjunto de líneas) es $3SUM - hard$.

"It was a game called Yes and No, where Scrooge's nephew had to think of something, and the rest must find out what; he only answering to their questions yes or no, as the case was..."
A Christmas Carol, Charles Dickens

# Soluciones

## Teoría de la Información

**P1.**

**Aproximación de Stirling.** 👌

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$
$$\log n! = n \log n + \mathcal{O}(n)$$

**Teorema 1.** 💎
*Todo árbol binario con $n$ hojas tiene altura $h \geq \log_2 n$*

a) Supongamos para simplificar que $k = 2^r$. Para resolver el problema utilizaremos una modificación del algoritmo de quicksort, y un algoritmo que encuentra la mediana en tiempo lineal [1]. Para esto utilizaremos:

- `particionar(A, i, j, p)`: *particionar* $A[i:j]$ usando el pivote en p.
- `posicionMediana(A, i, j)` : devuelve el índice de la mediana en $A[i:j]$.

Con esto construimos el algoritmo *kSort(A, i, j, k)* que $k$-ordena el arreglo $A[i,j]$:

```
Funcion kSort (A, i, j, k)
    if k = 1 then
        return
    end
    particionar(A, i, j, posicionMediana(A, i, j))
    kSort(A, i, (i+j)/2, k/2)
    kSort(A, (i+j+2)/2, j, k/2)
```

Finalmente, la profundidad de la recursión generada a partir de `kSort(A, 1, n, k)` es $\log k$ (pues el parámetro $k$ se divide a la mitad cada vez) y en cada nivel se hace trabajo lineal en el tamaño del arreglo, por lo que este algoritmo toma $\mathcal{O}(n \log k)$.

b) Las hojas del árbol de decisión respectivo contienen conjuntos de permutaciones que tienen los mismos elementos en los mismos bloques. Podemos contar esto considerando todas las permutaciones posibles y descontando los órdenes dentro de los grupos de tamaño $\frac{n}{k}$. Así, el árbol de decisión correspondiente tendrá $\frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^k}$ hojas y por 💎su altura debe ser $\geq \log \frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^k} = \log n! - \log \left(\left(\frac{n}{k}\right)!\right)^k$ 👌 $= n \log n - k\frac{n}{k} \log \frac{n}{k} + \mathcal{O}(n) = \Omega(n \log k)$

[1] Un algoritmo de búsqueda de media en tiempo lineal se basa en la elección de pivote de mediana de medianas, una explicación del algoritmo se puede encontrar en https://users.dcc.uchile.cl/~mcaceres/MedianFinding.pdf

"It was a game called Yes and No, where Scrooge's nephew had to think of something, and the rest must find out what; he only answering to their questions yes or no, as the case was..."
A Christmas Carol, Charles Dickens

c) Para esto basta con ordenar, con un algoritmo de ordenación óptimo, cada uno de los $k$ bloques a un costo de $\mathcal{O}\left(\frac{n}{k}\log\frac{n}{k}\right)$ cada uno, teniendo de esta forma un costo total de $\mathcal{O}\left(n\log\frac{n}{k}\right)$ ⚡

d) Las hojas del árbol de decisión respectivo contienen cada uno de los $\left(\left(\frac{n}{k}\right)!\right)^k$ posibles inputs y por lo tanto una cota inferior para el problema será $\log\left(\left(\frac{n}{k}\right)!\right)^k = k\left(\frac{n}{k}\log\frac{n}{k} + \mathcal{O}\left(\frac{n}{k}\right)\right) = \Omega\left(\frac{n}{k}\log\frac{n}{k}\right)$

**P2.** a) Si usamos 2 colas, una para nodos sin hijos $C_1$, en donde ponemos inicialmente los nodos con sus probabilidades ordenadas de forma creciente, y otra para los árboles que tienen hijos $C_2$ formados por el algoritmo, luego lo que hacemos es tomar los dos mínimos desde el frente de esas colas fusionarlos como antes y dejarlos al final de $C_2$ podemos hacer el algoritmo en tiempo $\mathcal{O}(n)$ (notar que el costo del algoritmo es simplemente el número de fusiones de Huffman que son $\leq 2n$ pues lo formado finalmente es un árbol).

b) Para demostrar la correctitud de Huffman demostraremos lo siguiente:

"Dado un conjunto de $i$ probabilidades, el algoritmo de Huffman construye un código óptimo" [2]

Por inducción en $i$. En el caso base $i = 2$ Huffman entrega el único árbol que representa un código libre de prefijos, por lo que este es óptimo. Para el paso inductivo asumiremos el enunciado cierto para valores menores a $i$.
Sea $T_H$ el árbol entregado por el algoritmo de Huffman corrido sobre las probabilidades $p_1, \ldots, p_i$ y definamos el peso del árbol $W(T_H)$ como el largo esperado de la codificación, es decir $\sum_{j=1}^{i} p_j n_j$. Supongamos ahora que $p_l$ y $p_m$ son los primeros 2 símbolos escogidos por Huffman (es decir los dos menores) y sea $T_H^*$ el árbol de aplicar Huffman sobre las probabilidades $\{p_1, \ldots, p_i, p_l + p_m\} \setminus \{p_l, p_m\}$ que por hipótesis inductiva ($i-1$ probabilidades) es óptimo.

Notemos ahora que se cumple $W(T_H) = W(T_H^*) + (p_l + p_m)$, puesto que correr el algoritmo sobre las nuevas probabilidades es lo mismo que correrlo sobre las antiguas a partir de la segunda iteración y además este nuevo nodo $p_l + p_m$ estará un nivel más arriba que los dos nodos $p_l$ y $p_m$.

Ahora consideremos un árbol óptimo $T$ para las probabilidades $p_1, \ldots, p_i$ de altura $h$. Notemos que tanto $p_l$ como $p_m$ deben estar ambos a profundidad $h$, pues:

- Si ambos están a una profundidad $< h$ podría cambiar cualquiera de ellos con un nodo de profundidad $h$ generando un árbol de menor (contradicción con que $T$ es óptimo) o igual (renombro $T$ de modo que el nuevo $T$ también es óptimo) $W$.

[2] Para demostrar que el código construído es óptimo veremos que su largo esperado es $\leq$ al de algún otro código óptimo.

- Si solo uno de ellos está a profundidad $h$ significaría que es el único a profundidad $h$ (si no intercambio el otro con el que no está a profundidad $h$ generando un árbol de menor (contradicción con que $T$ es óptimo) o igual (renombro $T$ de modo que el nuevo $T$ también es óptimo) $W$), pero esto es una contradicción pues podría quitarle un bit a su codificación y obtener un árbol de peso menor.

Asumamos ahora que $p_l$ y $p_m$ son hermanos en $T$ (si no lo fuesen intercambiamos al hermano de $p_l$ por $p_m$ renombrando $T$ y obteniendo un nuevo $T$ que también es óptimo). Consideremos además el árbol $T^*$ obtenido de reemplazar el padre de $p_l$ y $p_m$ por el nodo $p_m+p_l$. Con un razonamiento análogo al anterior obtenemos que $W(T) = W(T^*)+(p_l+p_m)$.

Finalmente como $T_H^*$ es óptimo (por H.I.), se cumple que:

$$W(T_H^*) \leq W(T^*)$$
$$W(T_H^*) + (p_l + p_m) \leq W(T^*) + (p_l + p_m)$$
$$W(T_H) \leq W(T)$$

Y como $T$ es óptimo $T_H$, el árbol generado por Huffman, también lo será.

## Adversario

**P3.** Cada vez que el algoritmo pregunta si existe arista $(u, v)$, el adversario responde sí en caso que $(u, v)$ sea una arista del grafo crítico $G$, y no en caso contrario. Asumamos por contradicción que el algoritmo hizo $y < x$ preguntas. Entonces hay un par $(u, v) \in (V \times V) \setminus E$ para el cual el algoritmo no ha preguntado su existencia. Sea $G'$ el grafo obtenido desde $G$ agregando el arco $(u, v)$. Notemos que $G$ y $G'$ son lógicamente correcto respecto a las respuestas que el adversario le dio al algoritmo. Sin embargo, $G$ no cumple $\mathcal{P}$ pero $G'$ si lo hace (ya que $G$ es crítico para $\mathcal{P}$), lo que es una contradicción.

**P4.** Sea $\mathcal{A}$ un algoritmo que resuelve el problema, $A$ el máximo y $B$ el segundo máximo. Entonces, podemos separar las comparaciones realizadas por $\mathcal{A}$ en:

a) Comparaciones entre $A$ y otro elemento.

b) Comparaciones que no involucran a $A$.

Primero veamos que b) $\geq n - 2$, pues el resto de los elementos (que no son $A$ ni $B$) fueron comparados con $B$ o entre ellos. Esto pues si el elemento no fue comparado con nadie, puedo "postularlo como el verdadero máximo" (en vez de $A$) llegando a una contradicción con la correctitud de $\mathcal{A}$. De manera análoga si solo fue comparado con $A$, puedo "postularlo como el segundo máximo".

"It was a game called Yes and No, where Scrooge's nephew had to think of something, and the rest must find out what; he only answering to their questions yes or no, as the case was..."
A Christmas Carol, Charles Dickens

Veamos ahora que a) $\geq \lceil \log n \rceil$.
Sea adversario 😈, que mantiene una función $\omega(i)$ de pesos de sus elementos.
Inicialmente todos los elementos parten con un peso igual a 1. Luego, si el algoritmo pregunta por la comparación entre dos elementos, se le responde tal como antes (si es que ya se había hecho esta pregunta) y en caso contrario, el 😈responde que es mayor aquel elemento que tenga mayor peso (responde conservando consistencia con sus preguntas anteriores) y actualiza los pesos agregándole todo el peso del perdedor de la comparación al ganador de esta.

Veamos que se cumplen las siguientes propiedades:

- $\omega(i) = 0$, solo si $i$ ha perdido alguna comparación.
- Si $\omega(i) > 0$, $i$ aún es un candidato a ser el máximo.
- La suma de los pesos siempre es $n$.
- Se debe cumplir que $\mathcal{A}$ al finalizar, todos los pesos de los elementos sean 0 salvo el de $A$ (si no tendría dos candidatos factibles a máximo). Y en tal caso se cumple que $\omega(A) = n$.

Finalmente, notamos que lo máximo que puede crecer el peso de un elemento en una iteración es duplicar su peso (cuando los pesos de los elementos enfrentados son iguales). Y por lo tanto, para pasar de 1 a $n$ necesita al menos $\lceil \log n \rceil$ incrementos. Luego necesita al menos $\lceil \log n \rceil$ comparaciones.

## Reducción

**P5.** Para reducir desde $3 - SUM$ a *3 puntos colineales*, transformaremos los números $x_1, \ldots, x_n$ del input de $3 - SUM$ en los puntos $(x_1, x_1^3), \ldots, (x_n, x_n^3)$ y se lo daremos como input a *3 puntos colineales*.
Veamos ahora que si existían 3 números distintos cuya suma fuese 0, digamos $a + b + c = 0$, entonces los puntos $(a, a^3), (b, b^3), (c, c^3) = (a, a^3), (b, b^3), (-a - b, -(a + b)^3)$ son colineales (se puede verificar calculando la pendiente de cualquier par de puntos, observando luego que su pendiente es la misma).
Del mismo modo, si tres puntos $(a, a^3), (b, b^3), (c, c^3)$ son colineales, entonces, las pendientes del primero con el segundo y del primero con el tercero deben serlo, es decir:

$$\frac{b^3 - a^3}{b - a} = \frac{c^3 - a^3}{c - a}$$
$$b^2 + ab + a^2 = c^2 + ac + a^2$$
$$(a + b + c)(b - c) = 0$$

Y como $(b, b^3)$ y $(c, c^3)$ son puntos distintos, entonces $(a + b + c)$.

"It was a game called Yes and No, where Scrooge's nephew had to think of something, and the rest must find out what; he only answering to their questions yes or no, as the case was..."
A Christmas Carol, Charles Dickens

**P6.** Para mostrar que *3 en una línea* es $3SUM - hard$ reduciremos desde *3 puntos colineales*. Sea entonces $(a, b)$ un punto del problema *3 puntos colineales*, para este creamos la recta $ax + by + 1 = 0$. Todas estas rectas se las pasamos como input al problema *3 en una línea*.

De este modo, si el problema *3 en una línea* encuentra un punto $(x_1, y_1)$ coincidente a tres rectas, significa que este punto satisface la ecuación de estas tres rectas, es decir:

$$a_1x_1 + b_1y_1 + 1 = 0$$
$$a_2x_2 + b_2y_2 + 1 = 0$$
$$a_3x_3 + b_3y_3 + 1 = 0$$

y por lo tanto, $(a_1, b_1), (a_2, b_2), (a_3, b_3)$, pertenecen a la recta $x_1x + y_1y + 1 = 0$ siendo una instancia que satisface *3 puntos colineales*. De manera análoga se puede verificar la recíproca, siendo la reducción válida.