Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Auxiliar 2 - "Cotas inferiores: Teoría de la Información y Reducciones"

Profesores: Pablo Barceló
Gonzalo Navarro
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

26 de marzo del 2018

P1. MultiMezcla

Se desea ordenar un arreglo $A[1, \ldots, n]$, formado por la concatenación de r arreglos ordenados de largo $n_1, \ldots, n_r$. Queremos probar que el problema es $\Omega\left(\sum_{i=1}^{r} n_i \log \frac{n}{n_i}\right)$, en términos de número de comparaciones en el peor caso.

a) Usando árbol de decisión pruebe la cota inferior.

b) Muestre un algoritmo que realice $\mathcal{O}(n \log r)$ comparaciones.
¿Contradice este algoritmo la cota inferior?

P2. Tuercas y Tornillos

Nos dan una fila de n tuercas y otra de los n tornillos que les corresponden, pero en cualquier orden. Visualmente no podemos comparar dos tuercas o dos tornillos, pero si podemos probar una tuerca con un tornillo y decidir cuál es mayor que la otra, o si se corresponden.

a) Demuestre que el problema de hacer corresponder a cada tuerca con su tornillo es $\Omega(n \log n)$ en comparaciones tuerca-tornillo incluso en promedio.

b) Muestre un algoritmo que realice $\mathcal{O}(n \log n)$ comparaciones en promedio.

P3. Distinción de Elementos

Utilice un argumento similar a una reducción para dar una cota inferior de $\Omega(n \log n)$ (en el modelo de comparaciones) para el problema de determinar si n enteros son diferentes.

P4. Multiplicación de Matrices

Muestre que el problema de multiplicar dos matrices de $n \times n$ se reduce al problema de calcular el cuadrado de una matriz. Usando su reducción, pruebe que no existe algoritmo $O(n^2)$ para calcular el cuadrado de una matriz.

"Information is not knowledge."
Albert Einstein

P1. MultiMezcla

a) Notamos que el árbol de decisión correspondiente a este problema es el mismo árbol del problema de ordenar, la diferencia está sin embargo en el número de outputs distintos que existen, pues ahora tenemos la restricción que los elementos de un subarreglo deben quedar ordenados en el arreglo final. Por conteo, existen $\frac{n!}{n_1!\cdot\ldots\cdot n_r!}$ outputs distintos y por lo tanto la altura del árbol de decisión es :

$$\begin{aligned}
\log \frac{n!}{n_1! \cdot \ldots \cdot n_r!} &= \log n! - \sum_{i=1}^{r} \log n_i! \\
&\approx n \log n - cn - \sum_{i=1}^{r} (n_i \log n_i - cn_i) \\
&= \sum_{i=1}^{r} n_i \log n - cn - \sum_{i=1}^{r} n_i \log n_i + cn \\
&= \sum_{i=1}^{r} n_i (\log n - \log n_i) \\
&= \sum_{i=1}^{r} n_i \log \frac{n}{n_i}
\end{aligned}$$

Donde usamos la aproximación de Stirling ($\log n! \approx n \log n - cn$).

b) Nuestro algoritmo mantiene una cola de prioridad de r elementos implementada con un heap (entonces las operaciones de inserción y eliminación toman $\mathcal{O}(\log r)$). Los elementos del heap además tienen un puntero que indica de cual subarreglo proviene este elemento. Así el algoritmo comienza insertando los primeros elementos de cada uno de los subarreglos, luego, repetidamente hasta que no queden más elementos en la cola es extrae el un elemento (poniéndolo en el arreglo final) y se inserta el siguiente elemento del mismo subarreglo de donde venía el elemento extraído.
Finalmente, como todos los elementos son insertados y removidos una vez, el algoritmo realiza $\mathcal{O}(n \log r)$ comparaciones.

Esto no contradice la cota inferior, pues :

$$\begin{aligned}
\sum_{i=1}^{r} n_i \log \frac{n}{n_i} &= n \left(\sum_{i=1}^{r} \frac{n_i}{n} \log \frac{n}{n_i} \right) \\
&\leq n \log \sum_{i=1}^{r} \frac{n_i}{n} \frac{n}{n_i} \\
&= n \log r
\end{aligned}$$

"Information is not knowledge."
Albert Einstein

Donde usamos la desigualdad de Jensen (Si $\sum \lambda_i = 1$ y f es cóncava, entonces $f\left(\sum_{i=1}^{r} \lambda_i x_i\right) \geq \sum_{i=1}^{r} \lambda_i f(x_i)$).

P2. Tuercas y Tornillos

a) Podemos interpretar el input de este problema como dos arreglos (uno de tuercas y uno de tornillos), por lo que el número de posibles input es $(n!)^2$. Luego, si consideramos que cada uno de los input se presenta con igual probabilidad y que las comparaciones realizadas por el algoritmo codifican aquel input podemos usar el Teorema de Shannon para mostrar que el algoritmo no puede hacer menos de $\log n!^2 = \mathcal{O}(n \log n)$ comparaciones en promedio.

b) Lo que hacemos es tomar un tornillo y particionar el arreglo de tuercas según ese tornillo. Con esto, tendremos las tuercas menores a ese tornillo a la izquierda, las mayores a la derecha y la tuerca que calza con el tornillo, con esta última tuerca particionamos los tornillos con lo que finalmente tendremos 2 subproblemas (el de la izquierda y el de la derecha) que resolvemos recursivamente.
Para concluir, notamos que la división de problemas en subproblemas es idéntica a la de QuickSort con la diferencia que en nuestra división hacemos 2 "particionar" en vez de 1, lo que aumenta el numero de comparaciones al doble, pero no cambia la complejidad promedio de $\mathcal{O}(n \log n)$ de QuickSort.

P3. Distinción de Elementos

Supongamos por contradicción que existe un algoritmo A que resuelve "Distinción de Elementos" en $o(n \log n)$.
Construiremos otro algoritmo que ordena en $o(n \log n)$ en base a las comparaciones que realiza A, lo que será una contradicción dada la cota inferior conocida de "ordenar".

Antes veamos que si etiquetamos los elementos del input de menor a mayor, $a_1, a_2, \ldots, a_n$, A debe haber comparado a_{i-1} con $a_i \forall i$ (si no, un adversario puede construir un input donde todos los elementos son distintos y otro igual excepto que $a_{i-1} = a_i$).

Consideremos ahora el grafo dirigido de comparaciones (aquel donde los nodos son los elementos del input y hay un arco de u a v si es que ambos se compararon y u resultó ser menor a v). Este grafo no tiene ciclos dirigidos (pues si tuviese uno el input sería inconsistente), por lo que podemos encontrar un orden topológico de los nodos del grafo que de hecho corresponde a $a_1, a_2, \ldots, a_n$, por la propiedad del párrafo anterior.

Con esto nuestro algoritmo, ejecuta A ($o(n \log n)$), con las comparaciones realizadas construye el grafo de comparaciones ($o(n \log n)$) y averigua el orden topológico del grafo ($o(n \log n)$), con lo que ordena el input en $o(n \log n)$, lo que es una contradicción.

"Information is not knowledge."
Albert Einstein

P4. Multiplicación de Matrices

Sean A y B las matrices de $n \times n$ que queremos multiplicar.
Creamos la matriz:

$$\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}$$

Que es una matriz de $2n^2 \times 2n^2$ por lo que lo podemos hacer en $\mathcal{O}(n^2)$.
Elevamos nuestra matriz al cuadrado:

$$\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} A^2 & A \times B \\ 0 & 0 \end{bmatrix}$$

y finalmente extraemos el resultado del cuarto superior derecho de este resultado en $\mathcal{O}(n^2)$.

Supongamos ahora por contradicción que existe un algoritmo para elevar al cuadrado una matriz de $\mathcal{O}(n^2)$, entonces podemos usar la reducción anterior para obtener un algoritmo de $\mathcal{O}(n^2)$ para multiplicación de matrices, lo que es una contradicción pues existe una cota inferior de $\omega(n^2)$ para ese problema.

"Information is not knowledge."
Albert Einstein