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

Auxiliar 1 - "Cotas inferiores"

Profesores: Pablo Barceló
Gonzalo Navarro
Auxiliares: Matilde Rivas
Bernardo Subercaseaux

24 de Septiembre del 2018

Si un problema no se puede hacer en menos de $g(n)$, esto es una cota inferior al problema y decimos que el problema es $\Omega(g(n))$

La técnica del Adversario consiste en construir dinámicamente el input de un algoritmo arbitrario, intentando llevarlo al peor caso.

P1. Grado 3

Muestre que determinar si un grafo no dirigido posee o no un vértice de grado 3 toma, en el peor caso, $\binom{n}{2} - n$ consultas del tipo "¿existe un arco entre los vértices u y v?", si la cantidad total de vértices es n.

P2. Búsqueda Aproximada en Texto

El problema de búsqueda aproximada en texto consiste en, dado un string $T[1, n]$, otro string más corto $P[1, m]$, y un entero $0 \leq k < m$, determinar si P ocurre en T permitiendo a lo más k errores (inserciones, borrados o reemplazos de caracteres en P (o T)). Ambos T y P son secuencias de caracteres de un alfabeto $[1, \sigma]$.

(a) Demuestre utilizando la técnica de adversario que no se puede resolver el problema general inspeccionando menos de $k + 1$ caracteres en cualquier ventana posible de T de largo m.

(b) Deduzca de lo anterior una cota inferior de la forma $\Omega(kn/m)$ para el problema, incluso en el caso promedio.

P3. Propiedades de un Grafo

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 sólo preguntas de la forma: ¿existe un arco entre los vértices u y v?.
Muestre que este algoritmo debe realizar a lo menos x preguntas.

P4. K-ordenar Decimos que un arreglo $A[1 \dots n]$ 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 $\Omega\left(n \log\left(\frac{n}{k}\right)\right)$.

P1. Grado 3

En este caso nuestro adversario mantendrá un grafo G que contiene las aristas que aún no han sido preguntadas por el algoritmo.

Inicialmente el adversario tiene un grafo completo $G = V^2$. Cada vez que el algoritmo le pregunta por la existencia de una arista $\{u, v\}$ el adversario setea $G \leftarrow G - \{u, v\}$, en caso que G no tenga un vértice de grado al menos 3 el adversario responde que la arista consultada es parte del grafo; en cambio, si G tiene un vértice de grado al menos 3 el adversario responde que la arista consultada NO es parte del grafo.

Veamos primero que el algoritmo necesita que el adversario le responda afirmativamente acerca de la existencia de una arista. Pues si nunca le responde afirmativamente significa que en G existe un vértice que tiene 3 aristas incidentes, es decir, 3 aristas que el algoritmo no ha consultado si pertenecen o no al grafo. Por lo tanto existen dos inputs posibles, consistentes con las respuestas del adversario, uno que tiene un vértice de grado 3 y otro que no lo tiene.

Veamos ahora que para que el adversario responda afirmativo a una pregunta se necesitan al menos $\binom{n}{2} - n$ consultas, con lo que finalizaremos esta demostración.

En el momento en que el adversario responde por primera vez afirmativamente, todos los vértices de G tienen grado ≤ 2 y por lo tanto G tiene a lo más n aristas. Finalmente, como cada arista que no está en G fue removida por una consulta anterior que se le hizo al adversario y como G comenzó con $\binom{n}{2}$ aristas, el algoritmo hizo al menos $\binom{n}{2} - n$ consultas.

P2. Búsqueda Aproximada en Texto

(a) La siguiente estrategia hace fallar a cualquier algoritmo que inspeccione menos de $k + 1$ caracteres en una ventana de largo m. Dada una ventana V y un patrón P, ambos de tamaño m:

- El adversario comienza con su ventana/buffer vacía.
- Cada vez que el algoritmo pregunta por un caracter V_i, el adversario contesta con algún elemento cualquiera del alfabeto distinto de P_i.

Supongamos que el algoritmo realiza menos de $k + 1$ consultas y responde si existe o no una ocurrencia aproximada.

Si el algoritmo responde que sí existe, entonces el adversario setea todos los caracteres que no han sido inspeccionados con valores distintos a los correspondientes en el patrón, lo que implica que el algoritmo se equivocó pues P no ocurre en la ventana con a lo más k errores.

En caso contrario, si el algoritmo responde que no existe una ocurrencia aproximada del patrón en la ventana, el adversario setea todos los caracteres no inspeccionados de V como los correspondientes en P. De este modo, sí existe una ocurrencia aproximada de P en V ya que a lo más k caracteres quedaron con errores.

De esta forma, se muestra que cualquier algoritmo que inspeccione menos de $k+1$ caracteres de la ventana no es capaz de determinar si existe o no una ocurrencia con a lo más k errores del patrón en ella.

(b) Dado un texto T de largo n, podemos dividirlo en n/m ventanas de largo m. El adversario anterior fuerza a cualquier algoritmo a realizar $k+1$ inspecciones en cada ventana (vamos repitiendo el procedimiento anterior), para declarar si la ventana corresponde a un calce aproximado o no. Luego el algoritmo debe realizar al menos $\Omega(kn/m)$ inspecciones.

P3. Propiedades de un Grafo

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. K-ordenar

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 $A[i, j]$:

```
1 Funcion kSort (A, i, j, k)
2 |  if k = 1 then
3 |  |  return
4 |  end
5 |  particionar(A, i, j, posicionMediana(A, i, j))
6 |  kSort(A, i, (i+j)/2, k/2)
7 |  kSort(A, (i+j+2)/2, j, k/2)
```

[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

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 su altura debe ser $\geq \log \frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^k} = \log n! - \log\left(\left(\frac{n}{k}\right)!\right)^k = n \log n - k\frac{n}{k} \log \frac{n}{k} + \mathcal{O}(n) = \Omega(n \log k)$

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 $\log\left(\left(\frac{n}{k}\right)!\right)^k = k\left(\frac{n}{k} \log \frac{n}{k} + \mathcal{O}\left(\frac{n}{k}\right)\right) = \Omega\left(\frac{n}{k} \log \frac{n}{k}\right)$

Agradecimientos a ~~Manuel~~ Ariel Cáceres :D