

Auxiliar 1 - “Árboles de decisión y Adversarios”

Profesor: Pablo Barceló

Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

22 de Marzo del 2017

P1. Buscar en arreglo ordenado

Considere el problema de buscar un elemento x en un arreglo de elementos distintos $A[1 \dots n]$ ordenado (es decir, cumple con la restricción $A[1] < A[2] < \dots < A[n]$). Para simplificar el análisis consideraremos que las consultas que puede hacer al input son del estilo: $\text{¿}A[k] < x\text{?}$

- a) Muestre que el problema es $\mathcal{O}(\log n)$
- b) Muestre, usando árbol de decisión, que el problema es $\Omega(\log n)$

P2. Mezclar 2 listas ordenadas

Considere el problema de mezclar 2 listas ordenadas de igual tamaño, $X[1, \dots, n], Y[1, \dots, n]$ de modo que el resultado final este también ordenado. En este problema estaremos interesados en las preguntas del estilo: $\text{¿}X[i] < Y[j]\text{?}$

- a) Muestre que el problema es $\mathcal{O}(2n - 1)$
- b) Muestre, usando árbol de decisión, que el problema es $\Omega(2n - \frac{1}{2} \log n)$
- c) Muestre, usando adversario, que el problema no puede ser resuelto en menos de $2n - 1$ comparaciones

P3. Conectividad de un grafo

Considere el problema de responder si un determinado grafo no dirigido es conexo. En este problema estaremos interesados en las preguntas del estilo: $\text{¿} \text{Existe una arista entre los nodos } u \text{ y } v\text{?}$

- a) Muestre que el problema es $\mathcal{O}(\binom{n}{2})$
- b) Muestre, usando adversario, que el problema no puede ser resuelto en menos de $\binom{n}{2}$

Soluciones

P1. Buscar en arreglo ordenado

- a) La “búsqueda binaria” es un algoritmo conocido que resuelve el problema en $\mathcal{O}(\log n)$ preguntas. ⚡
- b) Los posibles inputs del algoritmos son, que el elemento buscado se encuentre en alguna de las n posiciones del arreglo. De esta manera existen n inputs al problema. Por otro lado, como las preguntas del algoritmo ($A[k] < x?$) poseen 2 posibles respuestas, su árbol de decisión correspondiente es un árbol binario (donde cada camino a partir de un nodo lleva a los inputs consistentes con una de las respuestas (si o no)).

Con lo anterior, cada árbol de decisión construido de esta manera representa el comportamiento de un algoritmo que resuelve el problema, y la altura del árbol el número de preguntas que debe realizar en el peor caso. Basta entonces mostrar que la altura del árbol está acotada para mostrar que el problema no puede realizarse en menos de cierta cantidad de pasos.

Teorema 1. 💎

Todo árbol binario con n hojas tiene altura $h \geq \log_2 n$

Con lo ya dicho y el teorema 💎, podemos concluir que cualquier árbol de decisión de este problema tendrá altura mayor o igual a $\log_2 n$ y por lo tanto cualquier algoritmo le tomará $\log_2 n$ comparaciones o más, siendo el problema $\Omega(\log n)$.

P2. Mezclar 2 listas ordenadas

- a) El algoritmo de “merge” ocupado en “MergeSort” realiza en el peor caso $2n - 1$ comparaciones (una por cada elemento que pone en la lista de los ordenados). ⚡
- b) En total tenemos $2n$ elementos, pero no todas las $(2n)!$ permutaciones son posibles outputs del algoritmo (pues deben respetar que los órdenes en las listas originales se mantengan).


¿Cuántos outputs posibles hay entonces? 🤔

Para contarlos notemos que basta elegir las posiciones en las cuales se encuentran los elementos de una de las listas (si ya están escogidos el resto tiene su posición determinada) lo que se puede hacer de $\binom{2n}{n}$ formas distintas, siendo por tanto este el número de outputs posibles.

Si ponemos estos outputs en las hojas de un árbol de decisión, tenemos que por el teorema 💎 su altura será $\geq \log \binom{2n}{n}$.

Usando ahora la aproximación de Stirling ($n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$) llegamos a que $\log \binom{2n}{n} \sim 2n - \frac{1}{2} \log n - \frac{1}{2} \log \pi$, luego el problema es $\Omega(2n - \frac{1}{2} \log n)$

c) Por $\Rightarrow \Leftarrow$ existe un algoritmo A que realiza $\geq 2n - 2$ comparaciones.

El adversario  construye el siguiente input:

- $X \rightarrow$ lista cuyos elementos son $x_i = 2i - 1$
- $Y \rightarrow$ lista cuyos elementos son $y_i = 2i$


Si ejecutamos el algoritmo A sobre el input anterior encontraremos un i^* tal que x_{i^*} no fue comparado con ambos y_{i^*-1} e y_{i^*} (si todos los i lo cumplieran se habrían hecho más de $2n - 2$ comparaciones).

Supongamos sin mucha pérdida de generalidad que solo no fue comparado con y_{i^*} , entonces en el siguiente input:

- $X \rightarrow$ lista cuyos elementos son $x_i = 2i - 1$, excepto $x_{i^*} = 2i^*$
- $Y \rightarrow$ lista cuyos elementos son $y_i = 2i$, excepto $y_{i^*} = 2i^* - 1$

El algoritmo recibe exactamente las mismas respuestas a las mismas preguntas que se hizo sobre el input anterior (pues el orden relativo de los elementos no se altera y no se compara x_{i^*} con y_{i^*}). Por lo tanto tengo 2 inputs válidos de listas para los cuales A retorna el mismo resultado lo que es una $\Rightarrow \Leftarrow$.

P3. Conectividad de un grafo

- a) El problema puede ser resuelto aplicando un algoritmo de búsqueda en grafos (como BFS o DFS), los cuales son lineales en el tamaño del grafo, como un grafo puede tener a lo más $\binom{n}{2}$ aristas, el problema es $\mathcal{O}(\binom{n}{2})$ ⚡
- b) 💡! El  irá respondiendo las preguntas del algoritmo de manera de darle la menor cantidad de información posible. Para esto mantendrá dos grafos consistentes con las respuestas que le da al algoritmo, pero siendo uno conexo y el otro no, siendo imposible distinguir entre ambos grafos.

Para responder las preguntas del algoritmo el  sigue las siguientes reglas:

- Inicializa dos grafos C como grafo completo (todas las aristas) e I como el grafo vacío (ninguna arista).
- Si le preguntan si existe una arista que desconecta a C respondo que si está y la agrego a I .
- Si no, respondo que no está y la saco de C .

Observemos primero que I es un subgrafo de C . Además, si C tiene un ciclo, ninguna de las aristas del ciclo están en I (pues sacar una de ellas no desconecta C), y por lo tanto I es acíclico.

Veamos también que si $I \neq C$ entonces I es desconexo, pues si fuese conexo, sería un árbol (por ser acíclico), pero como $I \neq C$ hay una arista e en C que no está en I y que de hecho

es parte de un ciclo en C (cuyas otras aristas están todas en I), pero ninguna de las aristas de ese ciclo puede estar en I lo que es una $\Rightarrow \Leftarrow$.

Finalmente, si un algoritmo termina haciendo menos de $\binom{n}{2}$ preguntas hay una arista que está en C , pero no en I por lo que C es conexo pero I no.