CC4102 - Diseño y Análisis de Algoritmos

Prof. Gonzalo Navarro
Prof. Aux. Bernardo Subercaseaux

- Cotas inferiores
- Memoria externa

C1 —

- Amortizados
- Universo discreto

C2 —

~~Probabilísticos~~

- Probabilísticos/Aleatorizados.

C3 —

- Aproximados

Cotas Inferiores

Def Complejidad de un problema

→ es la complejidad del mejor algoritmo que lo resuelve.

→ cada nuevo algoritmo da una cota superior a la complejidad del problema $O(\dots)$

→ veremos cómo demostrar cotas inferiores a la complejidad de problemas. $\Omega(\dots)$

si tenemos que un problema es $O(T(n))$ y $\Omega(T(n))$, entonces

→ la complejidad del problema es $\Theta(T(n))$

→ los algoritmos de costo $O(T(n))$ son óptimos

→ las cotas inferiores $\Omega(T(n))$ son ajustadas.

cell probe model → cuenta acceso a RAM.

Técnicas para demostrar cotas inferiores

- Estrategia del adversario (peor caso)
- Teoría de la Información (caso promedio)
- Reducciones

Adversario - metáfora del peor caso
- intermediario entre el algoritmo y el input que va definiendo el input a medida que el algoritmo pregunta, buscando producirle el mayor costo
- siempre debe haber un input consistente con sus respuestas

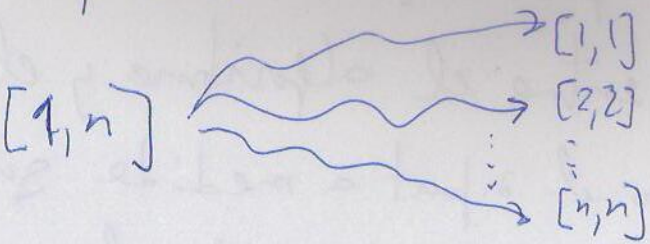
Ej: buscar un elemento x en un arreglo desordenado $A[1, n]$
requiere n accesos a A, pues sino el adversario pondrá x en una celda no mirada por el algoritmo.

Arreglo ordenado: el adversario debe respetar el orden al responder lo que hay en las celdas.

Modelo: el algoritmo sabe que, si x está en el arreglo, entonces está en el intervalo $A[i,j]$.

inicialmente, $[i,j] = [1,n]$

para poder responder correctamente, debemos tener $[i,j] = [k,k]$.



Si el algoritmo accede a $A[k]$, puede pasar que:

(1) $k \notin [i,j]$, y el algoritmo no aprende nada

(2) $A[k] = x$, y el algoritmo termina

(3) $A[k] < x$, y ahora el intervalo es $A[k+1,j]$

(4) $A[k] > x$, y ahora el intervalo es $A[i,k-1]$.

Obs (x inducción): el algoritmo nunca ha mirado una celda en $A[i,j]$.

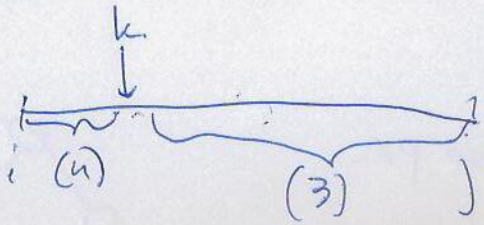
Adversario inteligente:

decide (3) si $k - i \leq j - k$

(4) si no

(3) lleva a $A[k+1, j]$

(4) lleva a $A[i, k-1]$.



⇒ el adversario consigue que el intervalo se reduzca, como mucho, a la mitad.

para llegar de un intervalo de largo n (e. inicial) a uno de largo 1 (e. final), el algoritmo deberá hacer, como mínimo, $\log_2 n$ accesos.

∴ $\log_2 n$ accesos a A es una cota inferior para buscar en un arreglo ordenado.

→ este adversario "nos sugiere" la búsqueda binaria $\left(k = \frac{i+j}{2}\right)$,

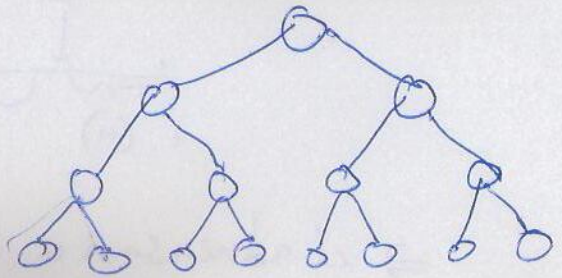
y ese algoritmo hace $\log_2 n$ accesos, lo que entonces es óptimo.

Máximo en un arreglo $A[1,n]$

```
max ← A[1]
for i ← 2 to n
    if A[i] > max
        max ← A[i].
```

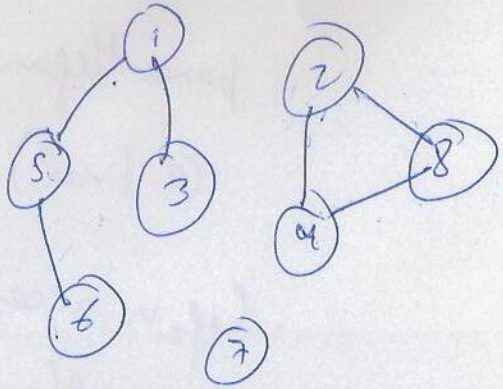
cota sup: $n-1$ comparaciones

cota inferior?



Modelo: un ~~grafo~~ grafo con n nodos, uno por elemento de A.

Conectamos i — j cuando el algoritmo compara $A[i]$ con $A[j]$.



Inicialmente, n nodos desconectados.

→ todo estado final es un grafo conexo.

un grafo conexo tiene al menos $n-1$ aristas,

todo algoritmo (correcto) que encuentre el máximo debe hacer $n-1$ comparaciones al menos.

→ la complejidad de encontrar el máximo en $A[1,n]$ es $n-1$ comparaciones.