Aux 1

20/03/2023

P0] Definiciones y Conceptos

- **Notación Big-O** Si un tiempo de ejecución es $O(f(n))$, entonces para n suficiente% grande, el tiempo de ejecución es a lo más $k \cdot f(n)$ para alguna constante k.
  // $g(x) \in O(f(x))$ ssi $\exists\, k > 0$ t.q. $g(x) \leq k \cdot f(x)$

- **Notación Big-Ω** Si un tiempo de ejecución es $\Omega(f(n))$, entonces para n suficiente% grande, el tiempo de ejecución es por lo menos $k \cdot f(n)$, para alguna constante k.
  / $g(x) \in \Omega(f(x))$ ssi $\exists\, k > 0$ t.q. $g(x) \geq k \cdot f(x)$

- **Notación Big-Θ** Si un tiempo de ejecución es $\Theta(f(n))$, entonces para n suficiente% grande, el tiempo de ejecución es por lo menos $k_1 \cdot f(n)$ y a lo más $k_2 \cdot f(n)$, para algunas constantes k_1, k_2.
  // $g(x) \in \Theta(f(x))$ ssi $\exists\, k_1, k_2 > 0$ t.q. $k_1 \cdot f(x) \leq g(x) \leq k_2 f(x)$

$O(1)$: orden constante

$O(\log(n))$: orden logarítmico

$O(\log \log n)$: orden sublogarítmico

$O(n)$: orden lineal

$O(n \log n)$: orden lineal logarítmico

- Complejidad: costo del mejor algoritmo
- Cota inferior ajustada: la cota inferior más alta posible.

¿Cómo saber si es el mejor algoritmo y si nuestra cota inferior es ajustada?

El algoritmo nos entrega una cota superior $O(T_1(n))$ y nuestra cota inferior es $\Omega(T_2(n))$. Si $T_1 = T_2$, entonces

→ nuestro algoritmo es óptimo ✓

→ cota inferior es ajustada ✓

→ el problema tiene complejidad $\Theta(T(n))$, $T = T_1 = T_2$.

Aux 1: cotas inferiores

P1) $f: \{0,1\}^n \to \{0,1\}$

$f(w) = 1$ ssi w contiene al menos 3 ceros consecutivos.

evasiva: si para determinar $f(w)=1$ necesitamos hacer exactamente n de estas preguntas.

Ej: $n=3$

$$\left.\begin{array}{l} ¿w_2? \to 0 \\ ¿w_2? \to 0 \\ ¿w_1? \to 0 \end{array}\right\} f(w)=1 \checkmark$$

O también pudimos haber preguntado por w_1

$$¿w_1? \begin{cases} 0 \to ¿w_0? ¿w_2? \\ 1 \to f(w)=0 \end{cases}$$

⚠ De todas formas, para $n=3$, nuestro adversario nos obligará a hacer 3 preguntas en el peor de los casos.

Para $n=4$, ¿es evasiva? $\iff$ ¿Necesitamos hacer 4 preguntas a nuestro adversario en el peor caso?

Sea $w = \_\ \_\ \_\ \_$, ¿qué bits debemos preguntar?

Veamos ¿w_1? En el peor caso $w_1 = 0$ (de otra forma, no existen tres 0s consecutivos y podemos afirmar que $f(w)=0$).

¿w_2? $w_2 = 0$

¿w_3? $w_3 = 1$

$\Rightarrow$ ¿w_0? $w_0 = 0$

$\therefore$ en el peor caso debemos hacer 4 preguntas para poder afirmar o no que $f(w) = 1$

$\Rightarrow$ para $n=4$, f es evasiva.

$\square$

Para $n=5$, ¿debemos hacer 5 preguntas?

El algoritmo debe ser astuto al hacer las preguntas al adversario. Por ejemplo, un algoritmo más lento preguntaría en orden: $w_0, w_1, \ldots, w_4$. Un algoritmo más eficiente comenzaría por w_2, ya que si $w_2 = 1$, entonces $f(w) = 0$ necesariam.

Si $w_2 = 0$, ¿cuál bit preguntamos ahora?

$$\underset{w_0}{\_} \quad \underset{w_1}{\_} \quad \underset{w_2}{0} \quad \underset{w_3}{\_} \quad \underset{w_4}{\_}$$

Si preguntamos por w_0 o w_4, vamos a deber preguntar por w_1 y w_3.

En cambio, si preguntamos por w_1 o w_3, luego SOLO tendremos que preguntar por 1 más:

Por ejemplo: ¿w_1? $\rightarrow 0$

¿w_0? $\rightarrow 1$ (peor caso)

$$\underline{1} \quad \underline{0} \quad \underline{0} \quad \_ \quad \_$$

Basta con preguntar por w_3 para poder responder si $f(w) = 1$.

Luego, para $n=5$, f <u>NO</u> es evasiva!

$\square$.

P2) Arreglo desordenado de n elementos

· Obtener los k elementos menores ordenados.

1) Encontrar el k-ésimo elemento $O(n)$

2) Obtener los k elementos menores $O(n)$

↳ por la técnica del adversario sabemos que la cota inferior es $\Omega(n)$, ya que en el peor caso debe recorrer todo el arreglo.
Luego, nuestro algoritmo es simple y recorrer el arreglo completo, lo cual toma tiempo $O(n)$.
∴ conocemos la cota inferior y superior y toman el mismo orden de tiempo $\Rightarrow$ · el algoritmo es óptimo
· cota inferior es ajustada
· problema de complejidad $\Theta(n)$.

3) Ordenar los k elementos $\longrightarrow O(k \log(k))$

Hemos diseñado un algoritmo óptimo que toma tiempo $O(n + k\log k)$

□.

P3) Arreglo desordenado: encontrar mínimo y máximo

1) Encontrar mínimo → $n-1$ comparaciones (visto en clases)

Encontrar máximo en los $n-1$ elementos restantes → $n-2$ comp.

⇒ El algoritmo toma $(n-1) + (n-2)$ comparaciones

⇔ $2n-3$ comparaciones.

2) La técnica del adversario también nos puede ayudar a diseñar un algoritmo óptimo.

Vamos a crear un <u>modelo</u> de lo que el algoritmo va aprendiendo.

Dividimos el conjunto de elementos en 4.

- $a = |A|$, A: elementos nunca comparados
- $b = |B|$, B: elementos que ganan (>) todas sus comparaciones
- $c = |C|$, C: elementos que ganan (<) todas sus comparaciones
- $d = |D|$, D: elementos que ganaron y perdieron alguna vez.

El <u>estado inicial</u> es $(a,b,c,d) = (n,0,0,0)$ y el <u>estado final</u> es $(a,b,c,d) = (0,1,1,n-2)$

	A	B	C	D
A	$\boxed{(a-2, b+1, c+1, d)}$	$(a-1, b, c, d+1)$ $\boxed{(a-1, b, c+1, d)}$	$(a-1, b, c, d+1)$ $\boxed{(a-1, b+1, c, d)}$	$\boxed{(a-1, b+1, c, d)}$ $\boxed{(a-1, b, c+1, d)}$
B		$\boxed{(a, b-1, c, d+1)}$	$\boxed{(a,b,c,d)}$ $(a, b-1, c-1, d+2)$	$\boxed{(a,b,c,d)}$ $(a, b-1, c, d+1)$
C			$\boxed{(a, b, c-1, d+1)}$	$\boxed{(a,b,c,d)}$ $(a, b, c-1, d+1)$
D				$\boxed{(a,b,c,d)}$

¿Qué opciones elegirá el adversario para que el algoritmo aprenda lo menos posible y así generar el mayor costo?

El adversario evita que "d" crezca.

Podemos ver que:

- "a" decrece a lo sumo de a 2. Para pasar de n a 0, necesitamos $\lceil \frac{n}{2} \rceil$ comparaciones
- "d" crece a lo sumo de a 1. Para pasar de 0 a $n-2$, necesitamos $n-2$ comparaciones.
- Nunca "a" decrece al mismo tiempo que "d" crece $\Rightarrow$ cotas son disjuntas $\Rightarrow$ cota inferior $\lceil \frac{3n}{2} \rceil - 2$ comparaciones.

Finalmente, esta técnica nos vislumbra un algoritmo.

1) Comparar los elementos de A de a pares dejando $\frac{n}{2}$ en B y $\frac{n}{2}$ en C. $\longrightarrow \lceil \frac{n}{2} \rceil$ comparaciones

2) Encontrar el máximo en B ($\frac{n}{2} - 1$ comparaciones)

3) " " mínimo en C " " ".

$\Rightarrow$ Total $\lceil \frac{3n}{2} \rceil - 2$ comparaciones

Nuestra cota inferior es $\lceil \frac{3n}{2} \rceil - 2$ comparaciones y nuestro algoritmo toma $\lceil \frac{3n}{2} \rceil - 2$ comparaciones

$\therefore$ nuestra cota inferior es ajustada y nuestro algoritmo es óptimo.

$\square$.