Técnicas Básicas: Algoritmos inspirados de Cotas Inferiores

Note Title

Tarea

Min	$A[1,\dots,n]$	$n-1 \in \Theta(n)$
Max	$A[1,\dots,n]$	$n-1 \in \Theta(n)$
Min, Max	$A[1,\dots,n]$	$2n-2 \in \Theta(n)$

mejor?

① **Cota Superior** $\Rightarrow \left\lceil \frac{3n}{2} \right\rceil - 2$ cmps

Algoritmo

① Comparar Pares $\left\lceil \frac{n}{2} \right\rceil$ cmps
- $\hookrightarrow P[1\dots n/2]$
- $\hookrightarrow G[1\dots n/2]$

② Usual Min y Max c $\left\lceil \frac{n}{2} \right\rceil - 2$ cmps

② Cota Inferior para Min Max (en el modelo de comparaciones)

p = # de elementos que perdieron al menos una cmp

g = # ——— ganaron ———

x = # nunca fueron comparados

el = # eliminados (ganaron y perdieron una cmp)

(algoritmos no hacen dos veces la misma comparación)

para cualquier algoritmo

En el inicio

$p = 0$
$g = 0$
$x = n$
$el = 0$

Cada comparación

① $x \leftarrow x-2$, $g \leftarrow g+1$, $p \leftarrow p+1$, $el \leftarrow el$

② $x \leftarrow x$, $g \leftarrow g$, $el \leftarrow el+1$, $p \leftarrow p+1$

③ $x \leftarrow x$, $el \leftarrow el+1$, $g+1$, $p \leftarrow p$

En el final

$p = n-1$
$g = n-1$
$x = 0$
$el = n-2$

Cuantas comparaciones se necesitan para
- reducir x a cero ?
- crecer d a n-2 .

$\frac{n}{2}$ aplicaciones de (1) para reducir x a cero

$n-2$ aplicaciones de (2) o (3) para crecer d a n-2

$\frac{3n}{2}-2$ aplicaciones de (1), (2) o (3) para calcular Min, Max.

QED

Nota: El algoritmo que vimos tiene espacio $O(n)$
La cota inferior sugiere un espacio $O(1)$

Cotos inferiores son utiles para el diseño

Otras tecnicas de Diseño de Algoritmos?

1. Listo
   - Programacion Dinamica
   - Dividir para reinar
   - Induccion
2. Ejemplos
   - Fibonacci → Programacion Dinamica & Induccion?
   - Subsequencias, Suma Maxima
   - Subsequencia Comun Mas Larga

③ Fibonacci

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

ⓐ Algoingenuo

```
Fib(n)
    si n=0 return 0
    si n=1 return 1
    sino return Fib(n-1) + Fib(n-2)
```

Complejidad

$= C(n)$

$C(0) = 1$

$C(1) = 1$

$C(n) = C(n-1) + C(n-2)$

$C(n) = F(n+1)$

$\in O(\phi^n)$

(b)

```
i ← 1
u ← 0      // F_{i-1}
v ← 1      // F_i
while (i < n)
    T ← u + v      // F_{i+1}
    u ← v
    v ← T
    i ← i + 1      // v = F_i
return v
```

Complejidad?

$O(n)$

Es optimal?

No

Tarea:
Vea con
mult de matriz

Si $n = 2^5 + 2^3 + 2^0 = 41$

$n = \overline{101001}_2$

$a^n = a^{2^5} * a^{2^3} * a^{2^0}$

Nota

```
u ∈ a ; i ∈ 1
Repeat
    u ∈ u * u
    i ∈ i + 1
until i ≤ ⌊lg n⌋
return u
```

calcule $a^{2^{\lfloor \lg n \rfloor}}$

Tarea: escribe el algoritmo Final

Complejidad Final

Tiempo $O(\lg n)$

Espacio $O(1)$ en este caso
$O(\text{output})$ en general

Cual de los otros exemplos?

① Subsec suma Maxima

[2, -2, 3, -2, 4, 5, 1, 2, -12, 3, 1]

② $O(n^2)$ subsecuencias ⇝ Solucion en tiempo $O(n^3)$

② Guardar las sumas intermedias tiempo $O(n^2)$, espacio $O(n)$

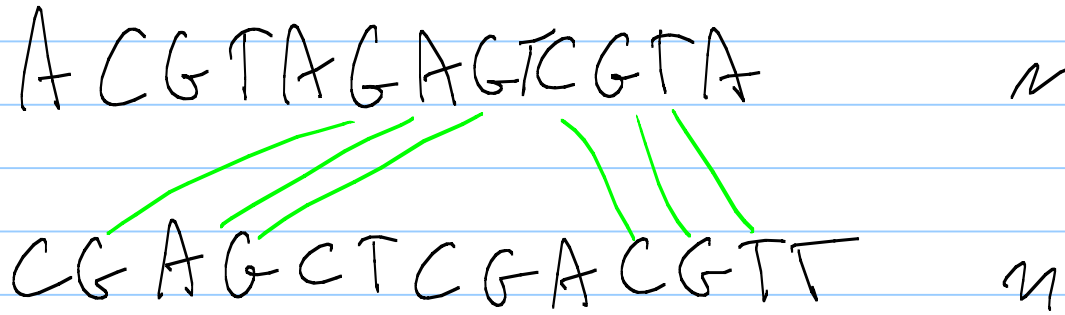
③ Suf

$O(n)$
$\Omega(n)$
$\Theta(n)$

suf[i] ← max{0, suf[i-1] + S[i]}

subs[i] ← max{subs[i-1], suf[i]}

(2) Subsec comun mas larga



Tarea: Cual es la complejidad de este problema? (Con justificación)