Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación

Ayudantía 10

Análisis de Algoritmos

Fernando Suárez

27 de Junio, 2014

Problema 1

Dado un arreglo $A[1...n]$, un *descenso* es un par de elementos adyacentes en el arreglo, $A[i]$ y A[i+1], tales que $A[i] > A[i+1]$.
Considere el siguiente algoritmo:

```
1: i ← 1
2: s ← 0
3: while i < n do
4:     if A[i] > A[i + 1] then
5:         s ← s + 1
6:     end if
7:     i ← i + 1
8: end while
9: return s
```

Conteste las siguientes preguntas:
a) ¿Qué hace el algoritmo ?
b) Enuncie un invariante del loop adecuado para demostrar que el algoritmo hace lo que propuso en (a)
c) Demuestre que, efectivamente, la expresión dada en la parte (b) es un invariante para el loop dado.
d) Demuestre la corrección parcial del algoritmo(o sea, si éste termina, entonces el valor retornado es el que propuso en la parte (a))
e) Demuestre que el algoritmo efectivamente termina.

Solución:

a) El algoritmo que retorna la variable s, la cual almacena el número de descensos que hay en el arreglo A.

b) $P(k) : 0 \leq k < n \rightarrow (i_k = k + 1 \wedge s_k = |\{j \in \mathbb{N} : 1 \leq j \leq k \wedge A[j] > A[j + 1]\}|)$

c) Por inducción sobre k:
Si $k = 0$, justo antes de ejecutar el loop por primera vez, se tiene
$i_0 = 1 \wedge s_0 = 0 \wedge |\{j \in \mathbb{N} : 1 \leq j \leq k \wedge A[j] > A[j + 1]\}| = 0)$, como se desea. Sea $k \geq 0$ y supongamos (HI) que $P(k)$ se cumple. Debemos demostrar que $P(k + 1)$ se cumple.
Si $k \geq n - 1$, $k + 1 \geq n$ y $P(k + 1)$ es trivialmente verdadero. Si $k < n - 1$, $k + 1 < n$ y entonces el loop se ejecuta por la iteración $k + 1$.
Tras $k + 1$ iteraciones, $i_{k+1} = i_k + 1 = k + 2$, y

$$s_{k+1} = \begin{cases} s_k, & \text{si } A[i_k] \leq A[i_{k+1}] = A[k + 2], \\ s_k + 1, & \text{si } A[i_k] > A[i_{k+1}] = A[k + 2] \end{cases}$$

Por otra parte,
$|\{j \in \mathbb{N} : 1 \leq j \leq k + 1 \wedge A[j] > A[j + 1]\}| =$

$$\begin{cases} |\{j \in \mathbb{N} : 1 \leq j \leq k \wedge A[j] > A[j + 1]\}|, & \text{si } A[i_k] \leq A[i_{k+1}] = A[k + 2], \\ |\{j \in \mathbb{N} : 1 \leq j \leq k \wedge A[j] > A[j + 1]\}| + 1, & \text{si } A[i_k] > A[i_{k+1}] = A[k + 2] \end{cases}$$

Por hipótesis de inducción, $s_k = |\{j \in \mathbb{N} : 1 \leq j \leq k \wedge A[j] > A[j + 1]\}|$, por lo que las cantidades correspondientes a $|\{j \in \mathbb{N} : 1 \leq j \leq k + 1 \wedge A[j] > A[j + 1]\}|$ y a s_{k+1} son iguales.

d) Supongamos que el algoritmo termina tras k iteraciones. Al terminar, debe tenerse $i_k = n$, o sea, $k + 1 = n$, o sea, $k = n - 1$. Por lo tanto, debe cumplirse $P(n - 1)$, por lo que se tiene que $s_{n-1} = |\{j \in \mathbb{N} : 1 \leq j \leq n - 1 \wedge A[j] > A[j + 1]\}|$, que es precisamente lo que se debe probar.

e) Notamos que la expresión $n - i$ es entera (obvio), estrictamente decreciente en el loop (ya que de un paso al siguiente su valor baja en 1, ya que i aumenta) y acotada inferiormente por 1 dentro del loop. Así, el loop es ejecutado una cantidad finita de veces.

Problema 2

a) Demuestre que, dado $n, k \in \mathbb{N}$, se tiene $\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$.

b) Demuestre que, dado $n \in \mathbb{N}$, se tiene $\sum_{i=1}^{n} \log_2 i \in \Theta(n \log_2 n)$

Solución:

a) Claramente $f(n) = \sum_{i=1}^{n} i^k \leq \sum_{i=1}^{n} n^k = n^{k+1}$, por lo que $f(n) \leq 1 \cdot n^{k+1}$ para todo $n \geq 1$.
Así, $f(n) \in O(n^{k+1})$.
Por otra parte,

$$f(n) = \sum_{i=1}^{n} i^k \geq \int_{i=0}^{n} x^k \, \mathrm{d}x = \frac{n^{k+1}}{k+1} = \frac{1}{k+1} \cdot n^{k+1} \tag{1}$$

Como $\frac{1}{k+1}$ es constante, se tiene $f(n) \in \Omega(n^{k+1})$.
Como $f(n) \in O(n^{k+1}) \cap \Omega(n^{k+1})$, tenemos $f(n) \in \Theta(n^{k+1})$.

Nota:

Otra forma de acotar inferiormente $f(n)$ es, en principio, pensar en

$$f(n) = \sum_{i=1}^{n} i^k \geq \sum_{i=\frac{n}{2}}^{n} i^k \geq \sum_{i=\frac{n}{2}}^{n} (\frac{n}{2})^k = \frac{n}{2} \cdot (\frac{n}{2})^k = \frac{1}{2^{k+1}} \cdot n^{k+1}. \tag{2}$$

Esta idea tiene el problema de que $n/2$ no es entero, y al tratar de reemplazarlo por $\lfloor n/2 \rfloor$o $\lceil n/2 \rceil$ se pierde un factor o éstos no son lo suficientemente grandes.
Para arreglar este problema, puede ocuparse:

$$f(n) = \sum_{i=1}^{n} i^k \geq \sum_{i=\lfloor \frac{2n}{3} \rfloor}^{n} i^k \geq \sum_{i=\lfloor \frac{2n}{3} \rfloor}^{n} (\frac{n}{3})^k \geq \frac{n}{3} \cdot (\frac{n}{3})^k = \frac{1}{3^{k+1}} \cdot n^{k+1}. \tag{3}$$

b) Por un lado, acotar a $\sum_{i=1}^{n} \log_2 i$ es sencillo, ya que

$$\sum_{i=1}^{n} \log_2 i \leq \sum_{i=1}^{n} \log_2 n = n \log_2 n, \tag{4}$$

por lo que $\sum_{i=1}^{n} \log_2 i \leq 1 \cdot n \log_2 n$ para todo $n \geq 1$.

Luego, sólo falta acotar a $\sum_{i=1}^{n} \log_2 i$ por abajo:

$$\sum_{i=1}^{n} \log_2 i \geq \sum_{i=\lfloor n/2 \rfloor + 1}^{n} \log_2 i \geq \sum_{i=\lfloor n/2 \rfloor + 1}^{n} \log_2(\lfloor n/2 \rfloor + 1) \tag{5}$$

$$\geq \sum_{i=\lfloor n/2 \rfloor + 1}^{n} \log_2(n/2) = (n - \lfloor n/2 \rfloor) \log_2(n/2) = \lceil n/2 \rceil \log_2(n/2) \tag{6}$$

$$\geq \frac{n}{2} \cdot \log_2(n/2) = \frac{n}{2} \cdot (\log_2(n) - 1) \tag{7}$$

Pero, para $n \geq 4$, $\log_2 n \geq 2$, por lo que $\log_2 n - 1 \geq \frac{1}{2} \cdot \log_2 n$.
Así, si $n \geq 4$, tenemos

$$\sum_{i=1}^{n} \log_2 i \geq \frac{n}{2} \cdot \frac{1}{2} \cdot \log_2 n = \frac{1}{4} \cdot n \log_2 n \tag{8}$$

$$\text{Así, hemos demostrado que si } n \geq 4, \textit{entonces} \tag{9}$$

$$\frac{1}{4} \cdot n \log_2 n \leq \sum_{i=1}^{n} \log_2 i \leq 1 \cdot n \log_2 n \tag{10}$$

$$\text{o sea, hemos probado que} \tag{11}$$

$$\sum_{i=1}^{n} \log_2 i \in \Theta(n \log_2 n). \tag{12}$$

Problema 3

Considere el siguiente algoritmo:

```
1: function MergeSort(A,f,l)
2:     if (n > 1) then
3:         m ← ⌊(f+l)/2⌋
4: MergeSort(A,f,m)
5: MergeSort(A,m+1,l)
6: Merge(A[f...m],A[m+1...l],A[f...l])
7:     end if
8: end function
```

a) Encuentre una función T(n) para la cantidad de comparaciones que realiza el algoritmo *MergeSort* en el peor caso, en función del tamaño del arreglo.
b) Demuestre que T(n) es no-decreciente para todo $n \in \mathbb{Z}^+$.
c) Calcule la complejidad de *MergeSort*. *Hint:* Calcule la compejidad para todo $n \in POTENCIA_2$.

Solución:

a)

$$\mathrm{T}(n) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ \mathrm{T}(\lceil n/2 \rceil) + \mathrm{T}(\lfloor n/2 \rfloor) + \Theta(n) & \text{eoc} \end{cases}$$

b) Sea

$$P(n) : \mathrm{T}(n) \leq \mathrm{T}(n+1) \tag{13}$$

Demostraremos que, para todo $n \geq 1$, se cumple $P(n)$. Para esto usamos inducción fuerte. Supongamos que para todo $k \in \mathbb{N}, 1 \leq k < n$, se cumple $P(k)$. Debemos probar que $P(n)$ se cumple. Hay dos casos:

- n=1.

En este caso, como

$$\mathrm{T}(2) = \mathrm{T}(\lceil \frac{2}{2} \rceil) + \mathrm{T}(\lfloor \frac{2}{2} \rfloor) + 2 = 1 + 1 + 2 = 4 \geq 1 = \mathrm{T}(1) \tag{14}$$

- $n > 1$.
  Como en este caso $n > \lceil \frac{n}{2} \rceil \geq \lfloor \frac{n}{2} \rfloor$, podemos aplicar la H.I. a $\lceil \frac{n}{2} \rceil$ y $\geq \lfloor \frac{n}{2} \rfloor$. Así,

$$\mathrm{T}(n+1) - \mathrm{T}(n) = \mathrm{T}(\lceil \frac{n+1}{2} \rceil) + \mathrm{T}(\lfloor \frac{n+1}{2} \rfloor) + n + 1 - \mathrm{T}(\lceil \frac{n}{2} \rceil) - \mathrm{T}(\lfloor \frac{n}{2} \rfloor) - n \tag{15}$$

$$\mathrm{T}(n+1) - \mathrm{T}(n) = \left(\mathrm{T}(\lceil \frac{n+1}{2} \rceil) - \mathrm{T}(\lceil \frac{n}{2} \rceil)\right) + \left(\mathrm{T}(\lfloor \frac{n+1}{2} \rfloor) - \mathrm{T}(\lfloor \frac{n}{2} \rfloor)\right) + 1 \tag{16}$$

  Por H.I. los últimos dos paréntesis deben ser mayores o iguales a 0. Luego, se cumple $P(n)$.

c) Dado que podemos asumir que el tamaño del arreglo pertenecerá a $POTENCIA_2$, las funciones *techo* y *piso* se simplifican. Nuestra recurrencia simplificada ahora se ve de la siguiente manera:

$$\mathrm{T}(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2\mathrm{T}(n/2) + n & \text{eoc} \end{cases}$$

Para encontrar una solución a la recurrencia debemos ver que pasa en cada iteración de esta, si usamos el reemplazo $n = 2^k$:

$$\begin{aligned} \mathrm{T}(n) &= 2\mathrm{T}(n/2) + n \\ \mathrm{T}(2^k) &= 2\mathrm{T}(2^{k-1}) + 2^k \\ \mathrm{T}(2^k) = 2(2\mathrm{T}(2^{k-2}) + 2^{k-1}) + 2^k &= 4\mathrm{T}(2^{k-2}) + 2 \cdot 2^k \\ \mathrm{T}(2^k) &= 8\mathrm{T}(2^{k-3}) + 3 \cdot 2^k \\ &\vdots \\ \mathrm{T}(2^k) &= 2^i \cdot \mathrm{T}(2^{k-i}) + i \cdot 2^k \end{aligned} \tag{17}$$

Es posible ver que en la iteración i la ecuación de recurrencia será $\mathrm{T}(2^k) = 2^i \cdot \mathrm{T}(2^{k-i}) + i \cdot 2^k$. Sin embargo, debemos demostrar esto por inducción:

$$\textbf{CB: } \mathrm{T}(2^k) = 2^1\mathrm{T}(2^{k-1}) + 1 \cdot 2^k \tag{18}$$

$$\textbf{HI: } \mathrm{T}(2^k) = 2^i\mathrm{T}(2^{k-i}) + i \cdot 2^k \tag{19}$$

$$\textbf{PI: } \mathrm{T}(2^k) = 2^i(2\mathrm{T}(2^{k-i-1}) + 2^{k-i}) + i \cdot 2^k = 2^{i+1}\mathrm{T}(2^{k-(i+1)}) + (i+1) \cdot 2^k \tag{20}$$

$$\tag{21}$$

Vemos que nuestra propuesta era válida. Además, es claro que en la iteración k podemos reemplazar el termino recursivo por el caso base:

$$\begin{aligned} \mathrm{T}(2^k) &= 2^k \cdot \mathrm{T}(1) + k \cdot 2^k \\ \mathrm{T}(2^k) &= 2^k + k \cdot 2^k \end{aligned} \tag{22}$$

Reemplazando $k = \log_2 n$

$$\mathrm{T}(n) = n + n\log_2 n \tag{23}$$

Luego, $\mathrm{T} \in \Theta(n\log_2(n))$.

Problema 4

a) Demuestre que el problema $3 - SAT$ es NP-completo
b) Demuestre que el problema $CLIQUE$ es NP-completo.

Solución:

a) Primero, notemos que el problema $3 - SAT \in NP$. Esto ya que dada una fórmula ψ en FNC de m cláusulas y n variables, es posible chequear si una valuación la satisface en tiempo polinomial($\leq n \cdot m$ en el peor caso).
Luego, dado que sabemos que SAT es NP-completo, nos basta con encontrar una función entre SAT y $3 - SAT$ que sea ejecutable en tiempo polinomial.
Dada una instancia de SAT con un conjunto $C = \{c_1, c_2, ..., c_m\}$ de cláusulas sobre las variables $U = \{u_1, u_2, ..., u_n\}$. Nos proponemos construir un conjunto C' con cláusula sobre las sobre las variables U', que consisten en las variables originales mas un conjunto de variables auxiliares.
Estas nuevas cláusulas se construirán de la siguiente manera:
Se debe reemplazar cada $c_i \in C$ por un conjunto de cláusulas de 3 literales sobre las variables que aparecen en c_i mas algunas variables auxiliares. El procedimiento exacto dependerá del número de literales en c_i. Sea $c_i \in C$ dado por $\{z_1, z_2, ..., z_k\}$ donde z_j es un literal sobre U. Tenemos los siguientes casos:

- $k = 1$.
  Tomemos $c_i = \{z_1\}$. Usaremos dos variables adicionales $\{y_{i,1}, y_{i,2}\}$. Luego formamos el conjunto $C'_i = \{\{z_1, y_{i,1}, y_{i,2}\}, \{z_1, y_{i,1}, \neg y_{i,2}\}, \{z_1, \neg y_{i,1}, \neg y_{i,2}\}\}$.
- k=2.
  Tomemos $c_i = \{z_1, z_2\}$. Usaremos una variable adicional $\{y_{i,1}\}$. Luego formamos el conjunto $C'_i = \{\{z_1, z_2, y_{i,1}\}, \{z_1, z_2, \neg y_{i,1}\}\}$.
- $k = 3$.
  Tomemos $c_i = \{z_1, z_2, z_3\}$. No usaremos variables adicionales. Luego $C'_i = c_i$.
- $k > 3$.
  Tomemos $c_i = \{z_1, z_2, ..., z_k\}$. Usaremos una variables adicionales $\{y_{i,1}, y_{i,2}, ..., y_{i,k-3}\}$.
  Luego formamos el conjunto:
  $C'_i = \{\{z_1, z_2, y_{i,1}\}, \{\neg y_{i,1}, z_3, y_{i,2}\}, \{\neg y_{i,2}, z_4, y_{i,3}\}, \{\neg y_{i,3}, z_5, \neg y_{i,4}\}, .., \{\neg y_{i,k-3}, z_{k-1}, z_k\}\}$.

Es sencillo darse cuenta que la cláusula original es satisfecha si y sólo si este nuevo conjunto es satisfecho. Nuevamente, estas transformaciones están acotadas por un polinomio en el largo de la cláusula(nm).

b) Primero, es fácil ver que $CLIQUE$ esta en NP. Esto ya que dada una instancia $< G, k >$ y un k-clique, podemos verificar en $O(n^2)$ que existe un k-clique en G.
Ahora, queremos encontrar una reducción de $3 - SAT$ a $CLIQUE$. Sea ψ una fórmula booleana en FNC. Por cada literal en ψ vamos a agregar un vértice en nuestro grafo.
Sea k el número de cláusulas en ψ. Vamos a conectar vada vértice a todos los vértices restantes que sean lógicamente compatibles EXCEPTO por los que están en la misma cláusula. Ahora debemos demostrar que G tiene un subgrafo completo de tamaño k ssi ψ es satisfactible.
($\Rightarrow$). Como todos los miembros del subgrafo pertenecen a cláusulas distintas, cualquier valuación σ que hace verdadero a todos literales en el subgrafo hace verdadera a la fórmula. (recordemos que dos literales complementarios no pueden estar en un subgrafo completo)
($\Leftarrow$). Si la fórmula ψ es satisfecha, debe existir una valuación σ que haga verdaderos a al menos un literal en cada cláusula. Sean $l_1 \in C_1, l_2 \in C_2, ..., l_k \in C_k$ estos literales. Notemos que no es posible que existan dos literales complementarios l_i y $l_j (1 \leq i < j \leq k)$. Necesariamente, entonces, podemos construir arcos entre cada para de nodos en donde aparecen dichos literales siguiendo las reglas de construcción del grafo.