



Analisi della complessità degli algoritmi

Mirko Cesarini, Paolo Costa

12 maggio 2005

Valutazione di un algoritmo

Passi da compiere

1. Esprimere la complessità come una funzione matematica della dimensione dei dati in ingresso
2. Valutare il Growth Rate della funzione

Selection Sort: codice

```
int [] array;
array = ... //inizializ. dell'array

for (int i=0; i<array.length-1; i++){
    int curMax = i;
    for (int j=i+1; j<array.length; j++)
        if (array[curMax] < array[j])
            curMax = j;
    swap (array, curMax, i);
}
```

Selection Sort: calcolo della $f(n)$

$$f(n) = \sum_{i=0}^{n-2} (c_a + c_s + \sum_{j=i+1}^{n-1} c_b) =$$

Tenendo conto che:

⑥ il loop esterno compie $n - 1$ passi ,

⑥ $\sum_{j=i+1}^{n-1} c_b = c_b * (n - (i + 1))$

si può semplificare così la formula:

$$= (n - 1)(c_a + c_s) + c_b \sum_{i=0}^{n-2} (n - (i + 1)) = \dots$$

Selection Sort: calcolo della $f(n)$ 2

$$\begin{aligned} \dots &= (n-1)(c_a + c_s) + c_b(1 + 2 + \dots + n-1) = \\ &= (n-1)(c_a + c_s) + c_b \frac{(1 + (n-1))(n-1)}{2} = \\ &= \frac{c_b}{2}n^2 + (c_a + c_s - \frac{c_b}{2})n - (c_a + c_s) \end{aligned}$$

Selection Sort: Growth Rate

$$f(n) = \frac{c_b}{2}n^2 + (c_a + c_s - \frac{c_b}{2})n - (c_a + c_s)$$

I termini $(c_a + c_s - \frac{c_b}{2})n - (c_a + c_s)$ sono minorabili da 0 e maggiorabili da kn^2 (k costante arbitraria). Quindi per n sufficientemente grande:

$$f(n) \geq \frac{c_b}{2}n^2 \Rightarrow f(n) \text{ is in } \Omega(n^2)$$

$$f(n) \leq (\frac{c_b}{2} + k)n^2 \Rightarrow f(n) \text{ is in } O(n^2)$$

$$\dots \Rightarrow f(n) = \Theta(n^2)$$

Proprietà utili

1. If $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(h(n))$, then $f(n)$ is in $O(h(n))$. [Transit.]
2. If $f(n)$ is in $O(kg(n))$ for any constant $k > 0$, then $f(n)$ is in $O(g(n))$. [No constant]
3. If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$, then $(f_1 + f_2)(n)$ is in $O(\max(g_1(n), g_2(n)))$. [Drop low order terms]
4. If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$ then $f_1(n)f_2(n)$ is in $O(g_1(n)g_2(n))$. [Loops]

Queste regole valgono per O , Ω e Θ .

Proprietà utili: esempio

Proviamo ad utilizzare le proprietà per il calcolo delle seguenti funzioni:

⑥ $c_6 2^n + c_7 n^6$

$$O(c_6 2^n + c_7 n^6) \rightarrow [\text{Drop low order t.}] O(c_6 2^n)$$

$$O(c_6 2^n) \rightarrow [\text{No Constants}] O(2^n)$$

⑥ $n(\log n + n)$

$$f1(n) = n; O(f1(n)) = O(n)$$

$$f2(n) = \log n + n; O(f2(n)) \rightarrow [\text{Drop low o. t.}] O(n)$$

$$O(f1(n)f2(n)) \rightarrow [\text{Loops}] O(nn) = O(n^2)$$

[Loops]: attenzione

Data un'espressione $f(n)$ esprimibile attraverso la moltiplicazione di due sottoespressioni:

$$f(n) = f_1(n) f_2(n)$$

la proprietà [Loops] si può applicare se e solo se f_1 e f_2 sono tra loro indipendenti (e quindi *separabili*).

Prendiamo il seguente esempio:

```
for(i=1; i<=n; i++)  
    for(j=1; j<=i; j++)  
        sum++;
```

[Loops]: attenzione 2

... In questo caso i passi del ciclo interno hanno un costo variabile (e dipendente dal contatore del ciclo esterno). Non posso usare il prodotto di due funzioni indipendenti, ma devo procedere così:

Nel caso dell'inner loop, abbiamo

$$f_2(i) = ic_s$$

Nel caso del loop esterno abbiamo

$$f_1(n) = \sum_{i=1}^n f_2(i) = \sum_{i=1}^n c_s i = c_s \sum_{i=1}^n i = c_s \frac{n(1+n)}{2} = \Theta(n^2)$$

Cicli con incremento esponenziale

Non sempre la complessità di due cicli innestati è $\Theta(n^2)$.
Consideriamo il seguente esempio:

```
sum2=0;  
for (k=1; k<=n; k*=2)  
    for (j=1; j<=k; j++)  
        sum2++;
```

Siccome k assume valori della potenza di due posso sostituirlo con 2^i e far variare i da 0 a $\log n$:

$$f(n) = \sum_{i=0}^{\log n} (c_s 2^i) = c_s \sum_{i=0}^{\log n} 2^i =$$

Cicli con incremento esponenziale 2

$$f(n) = \sum_{i=0}^{\log n} (c_s 2^i) = c_s \sum_{i=0}^{\log n} 2^i =$$

applico $\sum_{k=0}^n x^k = \frac{x^{n+1}-1}{x-1}$ (serie geometrica) ottenendo

$$= c_s \frac{2^{1+\log n} - 1}{2 - 1} = 2c_s 2^{\log n} - c_s = 2c_s n - c_s$$

A questo punto é facile dedurre che $f(n) = \Theta(n)$

Tema d'esame: 6 settembre 2002

Si consideri il seguente frammento di codice in pseudo-C:

```
int a[n], b[n];
j = 1; i = 2;
while (i <= n) {
    b[j] = a[i];
    j++;
    i = i * i;
}
```

Si determini l'ordine di grandezza Θ della complessità temporale di tale frammento in funzione di n .

Soluzione

Si considerino i valori assunti da i nelle prime iterazioni:

$$2, 4, 16, 256, \dots$$

A prima vista non è immediato ricondurli ad una potenza di un qualche numero k . Proviamo però a riscriverli così:

$$2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^3}, \dots$$

A questo punto risulta evidente che i cresce come 2^{2^k}

Soluzione - 2

Se utilizziamo k come indice della nostra sommatoria, esprimiamo k in funzione di i ovvero $k = \log \log i$ e varierà tra 0 (per $i = 2$) e $\log \log n$ (per $i = n$). $f(n)$ diventa:

$$\begin{aligned} f(n) &= \sum_{k=0}^{\log \log n} (c_s) = c_s \sum_{i=0}^{\log \log n} 1 = \\ &= c_s (\log \log n + 1) \\ &= \Theta(\log \log n) \end{aligned}$$