

Analisi della complessità degli algoritmi

Matteo Migliavacca, Mirko Cesarini, Paolo Costa

18 maggio 2005

Valutazione di un algoritmo

Passi da compiere

- 1. Esprimere la complessità come una funzione matematica della dimensione dei dati in ingresso
- 2. Risolvere la serie in forma chiusa
- 3. Valutare il Growth Rate della funzione

Alcuni esempi banali

a = b;
$$f(n) = c$$
 for (i=1; i <=n; i++) copia[i]=orig[i];
$$f(n) = c_0 + c_1 + \sum_{i=1}^n (c_2 + c_3 + c_1) =$$

$$= c_0' + \sum_{i=1}^n (c_1') = c_0' + nc_1'$$

Attenzione a inidice iniziale e finale (i=n sono ancora nel ciclo) c_0, c_1, c_3 sono legati ai tre parametri della riga "for"

Serie

Linearità

$$\sum_{k=1}^{n} (ca_k + b_k) = c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

Ad esempio

$$\sum_{k=a}^{b} c = (b-a-1)c$$

Alcuni esempi banali(2)

$$\begin{array}{l} \text{sum} = 0; \\ \text{for (i=1; i <=n; i++)} \\ \text{for (j=1; j<=n; j++)} \\ \text{sum} + \\ f(n) = c_0 + c_1 + c_2 + \sum_{i=1}^n \left(c_3 + c_4 + \sum_{j=1}^n \left(c_5 + c_6 + c_4\right) + c_7 + c_2\right) \\ \\ f(n) = c_0 + \sum_{i=1}^n \left(c_2 + \sum_{j=1}^n c_3\right) = \\ \\ = c_0 + \sum_{i=1}^n c_2 + \sum_{i=1}^n \sum_{j=1}^n c_3 = c_0 + nc_2 + n^2c_3 \end{array}$$

Analisi asintotica del growth rate

Proprietà utili

- 1. If f(n) is in O(kg(n)) for any constant k > 0, then f(n) is in O(g(n)). [No constant]
- 2. 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]

Queste regole valgono per O, Ω e Θ .

Alcuni esempi banali(3)

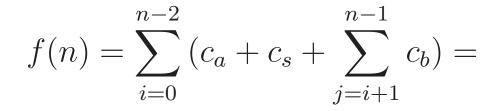
```
static int largest(int[] array) { int largest = 0; for(int i=0; i<array.lenght; i++) if (array[i] > currLargest) currLargest = array[i]; return currLargest; } f(n) = c_0 + \sum_{i=0}^{n-1} (c_{\text{test}} + p_{\text{true}}c_1) = c_0 + \sum_{i=0}^{n-1} c_2
```

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);
}</pre>
```

Selection Sort: calcolo della f(n)



Tenendo conto che:

- $oldsymbol{6}$ il loop esterno compie n-1 passi,

si può semplificare così la formula:

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

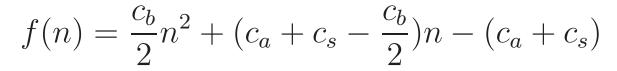
Selection Sort: calcolo della f(n) 2

$$\dots = (n-1)(c_a + c_s) + c_b(1 + 2 + \dots n - 1) =$$

$$= (n-1)(c_a + c_s) + c_b \frac{(1 + (n-1))(n-1)}{2} =$$

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

Selection Sort: Growth Rate



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). Quind per n sufficientemente grande:

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

$$f(n) \le (\frac{c_b}{2} + k)n^2 \Rightarrow f(n) \text{ is in } 0(n^2)$$

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

Proprietà utili

- 1. If f(n) is in O(kg(n)) for any constant k > 0, then f(n) is in O(g(n)). [No constant]
- 2. 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]
- 3. 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]
- 4. 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.]

Queste regole valgono per O, Ω e Θ .

Proprietà utili: esempio

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

- $\begin{array}{ll} & c_6 2^n + c_7 n^6 \\ & O(c_6 2^n + c_7 n^6) \rightarrow_{[\mathsf{Drop\ low\ order\ t.}]} O(c_6 2^n) \\ & O(c_6 2^n) \rightarrow_{[\mathsf{No\ Constants}]} O(2^n) \end{array}$
- $f_1(n) = n; O(f_1(n)) = O(n)$ $f_2(n) = \log n + n; O(f_2(n)) \rightarrow_{[Drop low o. t.]} O(n)$ $O(f_1(n)f_2(n)) \rightarrow_{[Loops]} O(nn) = O(n^2)$

[Loops]: attenzione

Data un'espressione f(n) esprimibile attraverso la motiplicazione 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++;</pre>
```

[Loops]: attenzione 2

... In questo caso i passi del ciclo interno hano 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

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

[Serie 2]Serie utili



$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Serie Geometrica Per $x \in R, x \neq 1$:

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

Somma di quadrati

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Attenzione al valore iniziale (geometrica)

Cicli con incremento esponenziale

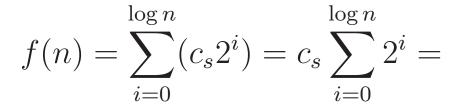
Non sempre la complessità di due cicli innestati è $\Theta(n^2)$. Consideriamo il seguente esempio:

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

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



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;
}</pre>
```

Si determini l'ordine di grandezza ⊖ 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:

$$f(n) = \sum_{k=0}^{\log \log n} (c_s) = c_s \sum_{i=0}^{\log \log n} 1 =$$

$$= c_s \log \log n$$

$$= \Theta(\log \log n)$$

Algoritmi ricorsivi

```
Con la ricorsione il calcolo di f(n) si complica:
int findMax(int[] array, int low, int up){
   if (low == up)
     return array[low];
   med = (low + up)/2;
   maxsx = findMax(array, low, med);
   maxdx = findMax(array, med+1, up);
   if (maxdx > maxsx)
      return maxdx;
   else
      return masx;
```

Equazioni alle ricorrenze

Per il calcolo di f(n) bisogna utilizzare le equazioni alle ricorrenze.

$$f(n) = \begin{cases} 2f(\frac{n}{2}) + k & \text{se } n > 1\\ k_1 & \text{se } n = 1 \end{cases}$$

Espandiamo le ricorrenze nell'equazione:

$$f(n) = \dots$$

Equazioni alle ricorrenze 2

$$= 2(2f(\frac{n}{4}) + k) + k$$

$$= 2(2(2f(\frac{n}{8}) + k) + k) + k$$

$$= 2^{\log n} f(1) + k \sum_{i=0}^{\log n-1} 2^i =$$

$$=2^{\log n}k_1+k\sum_{i=0}^{\log n-1}2^i$$