COMPLESSITÀ COMPUTAZIONALE DEGLI ALGORITMI

Fondamenti di Informatica a.a.2006/07

Prof. V.L. Plantamura

Dott.ssa A. Angelini

Classificazione degli algoritmi

- Tassonomia di costo:
  - algoritmo **costante**: c_0
  - algoritmo **lineare**: $c_1 * n + c_0$
  - algoritmo **quadratico**: $c_2 * n^2 + c_1 * n + c_0$
  - algoritmo **polinomiale**:
    $c_k * n^k + c_{k-1} * n^{k-1} + \dots c_2 * n^2 + c_1 * n + c_0$
  - algoritmo **logaritmico**: $c * \log_2 n$
  - algoritmo **esponenziale**: $c * 2^n$

dove: n è la dimensione dell'input
$c, k, c_0, c_1, \dots, c_k$, sono costanti indipendenti da n

Tirannia del tasso di crescita

Algoritmo	Complessità	Dimensione massima del problema		
	temporale	1 sec	1 min	1 ora
A_1	n	1.000	60.000	3.600.000
A_2	$n \log n$	140	4.893	200.000
A_3	n^2	31	244	1.897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

•algoritmi di complessità n^k sono utilizzabili solo con dimensioni non troppo elevate;

•complessità esponenziali sono invece inutilizzabili anche per input di dimensioni piccole.

Algoritmo	Complessità	Dimensione massima del problema		
	temporale	1 sec	1 min	1 ora
A_1	n	10.000	600.000	360.000.000
A_2	$n \log n$	9.990	599.900	3.599.900
A_3	n^2	97	750	5.700
A_4	n^3	21	80	300
A_5	2^n	12	18	24

Utilizzando un calcolatore 10 volte più potente

- algoritmi di complessità n o nlogn traggono pieno vantaggio dalla evoluzione tecnologica;
- per algoritmi polinomiali (n^2) il vantaggio è ancora evidente;
- negli algoritmi esponenziali i vantaggi sono irrilevanti.

Calcolo al limite

- La funzione *T(n)* è solitamente molto complessa;
- Il confronto tra i diversi algoritmi ha valore solo per dimensioni di input molto grandi;
- I termini che non modificano sostanzialmente l'ordine di grandezza dei valori possono essere eliminati dalla valutazione.

Esempio

- Consideriamo la funzione:

$$T(n) = n^2+100n+\log_{10}n+1000$$

n	*T(n)*	n^2		$100n$		$\log_{10}n$		1000	
	valore	**valore**	**%**	**valore**	**%**	**valore**	**%**	**valore**	**%**
1	1.101	**1**	**0,10**	100	9,100	0	0,0000	1.000	0,820
10	2.101	**100**	**4,76**	1.000	47,600	1	0,0500	1.000	7,620
100	21.002	**10.000**	**47,60**	10.000	47,600	2	0,9910	1.000	4,760
1.000	1.101.003	**1.000.000**	**90,80**	100.000	9,100	3	0,0003	1.000	0,090
10.000	101.001.004	**100.000.000**	**99,00**	1.000.000	0,990	4	0,0000	1.000	0,001
100.000	10.010.001.005	**10.000.000.000**	**99,90**	10.000.000	0,099	5	0,0000	1.000	0,000

Calcolo al limite

- Nell'esempio precedente possiamo asserire che *il tempo di esecuzione è al più proporzionale al quadrato della dimensione dell'input;*
- In genere le costanti di proporzionalità non vengono prese in considerazione perché dipendono da vai fattori (tecnologici): bontà del compilatore, velocità del computer, etc…

Esempio

- Supponiamo di avere, per uno stesso problema, sette diversi algoritmi con diverse complessità.
- Supponiamo che un passo sia eseguito in un μsec (10^{-6} sec) ed osserviamo i tempi di esecuzione per diversi valori di n:

Classe	n=10	n=100	n=1000	$n=10^6$
$\sqrt{n}$	$3*10^{-6}$	10^{-5}	$3*10^{-5}$	10^{-3}
n +5	$15*10^{-6}$	10^{-4}	10^{-3}	1
2n	$2*10^{-5}$	$2*10^{-4}$	$2*10^{-3}$	2
n^2	10^{-4}	10^{-2}	1	10^6 (~12 giorni)
n^2+n	10^{-4}	10^{-2}	1	10^6 (~12 giorni)
n^3	10^{-3}	1	10^5 (~1giorno)	10^{12} (~300 secoli)
2^n	10^{-3}	10^{14} (..secoli)	--------------	--------------

Calcolo al limite

- Per grandi dimensioni dell'input, gli algoritmi possono essere classificati in base ai tempi di risposta:

Algoritmo	*Classe di tempo*	*Proporzionalità*
$\sqrt{n}$	frazioni di sec.	$\sqrt{n}$
n+5, 2n	secondi	n
n^2, n^2+n	giorni	n^2
n^3	secoli	n^3
2^n	?	2^n

Notazioni asintotiche

- Visto che l'incidenza sul tempo di calcolo cresce al crescere di n, l'analisi che si conduce è un analisi al limite, ovvero si studia il comportamento dell'algoritmo per un ***n*** sufficientemente grande;
- Le notazioni che si introducono sono pertanto dette *notazioni asintotiche*.

Motivazioni e Definizioni

- Nelle definizioni delle notazioni asintotiche ritroviamo formalizzato il concetto di analisi al limite;
- Per dire che una funzione $T(n)$ appartiene ad un certo insieme $X(g(n))$ è necessario che l'andamento relativo di $T(n)$ e $g(n)$ sia di un certo tipo *a partire da una prefissata dimensione del dato di ingresso*, ovvero per ogni n maggiore di un certo n_0.

Motivazioni e Definizioni

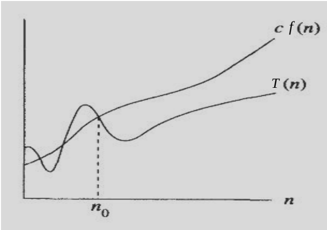
- In particolare, in informatica, sono tre le notazioni comunemente adottate: O, Ω, Θ;
- Una delle ragioni per cui vengono adottate tre notazioni è che le prime due, come sarà evidenziato nel seguito, forniscono un limite "lasco", rispettivamente per i limiti superiore ed inferiore, mentre la terza fornisce un limite "stretto".

O *(grande o)*

- Diciamo che $T(n) = O(f(n))$, *- leggiamo "T(n) ha complessità* grande o di *f(n)"* - se esistono due costanti positive c ed n_0, tali che per ogni $n > n_0$ risulti $T(n) \leq cf(n)$
- Ovvero, a partire da una certa dimensione n_0 del dato di ingresso, la funzione $f(n)$ maggiora la funzione $T(n)$. Possiamo quindi anche dire che la $f(n)$ rappresenta un limite superiore per la $T(n)$ (una delimitazione asintotica superiore).

$\mathcal{O}$ *(grande o)*

- *T(n) =* $\mathcal{O}$*(f(n))*



$\mathcal{O}$ *(grande o) - Esempio*

- Consideriamo la funzione: $T(n) = 2n^2 + 3n + 1 = \mathrm{O}(n^2)$
- Dobbiamo risolvere la disuguaglianza:

$$2n^2 + 3n + 1 \le cn^2, \text{dividendo per } n^2$$

$$2 + \frac{3}{n} + \frac{1}{n^2} \le c, \text{ che ha infinite soluzioni.}$$

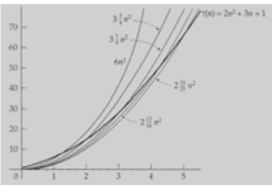
- Diverse coppie di valori di c ed n_0 calcolate usando la definizione di O grande:

c	≥ 6	≥ 3 + (3/4)	≥ 3 + (1/9)	≥ 2 + (13/16)	≥ 2 + (16/25)	…	→	2
n_0	1	2	3	4	5	…	→	∞

Le costanti c ed n_0

Il grafico mostra la funzione f tracciata per i diversi valori di c.

Le funzioni *f(n)* e *T(n)* crescono allo stesso ritmo.



La definizione asserisce che f è "quasi sempre" ≥ T se è moltiplicata per una costante c. Quasi sempre significa per tutti gli n ≥ n_0.

Ma c dipende da n_0 e viceversa. Se si sceglie n_0=2 allora c=3,75, etc…

Inoltre n_0 è sempre il punto in cui *cf(n)* e *T(n)* si intersecano.

Esempio - Dimostrazione

- Sia $T(n) = 2n^2 + 3n + 1$

  Consideriamo $n^2 + (3/2)n + (1/2) = T(n)/2$

  Si ha $(3/2)n + (1/2) \le n^2$ per $n \ge 2$, per cui

  $n^2 + (3/2)n + (1/2) \le n^2 + n^2 = 2\, n^2$ per $n \ge 2$,

  Per cui $T(n) \le 4\, n^2$; $T(n) = O(n^2)$

e le costanti utilizzate sono quindi: $c=4$ e $n_0=2$

Oss: la costante n_0 dipende dalla costante di proporzionalità c prefissata.

O (grande o)

- Tale limite non è però "stretto". In questo esempio $O(n^2)$ è una delimitazione asintotica superiore. È anche vero che n^3 maggiorerà la $T(n)$, e quindi $T(n)$ appartiene anche a $O(n^3)$;
- E' evidente che quest'ultima appartenenza implica un limite sicuramente meno stretto del precedente, ma rimane comunque formalmente ineccepibile.

Esempio

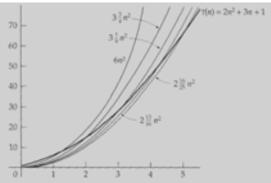
- $2n + 5 = O(n)$ *poiché* $2n+5 \le 7n$ per ogni n,
- $2n + 5 = O(n^2)$ *poiché* $2n+5 \le n^2$ *per* $n \ge 4$,
- $2n + 5 = O(2^n)$ *poiché* $2n+5 \le 2^n$ *per* $n \ge 4$,

- $n^2 = O(n)$? *NO:* $n^2/n = n$ *non limitata*
- $e^n = O(n)$? *NO:* $e^n/n \ge n^2/n = n$ *non limitata*

Termine dominante

$$T(n) = 2n^2 + 3n + 1 = O(n^2)$$

Termine dominante

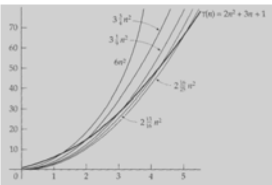


Per scegliere la coppia migliore c, n_0 che risolve la definizione di O, bisognerebbe determinare per quale valore di n_0 un certo termine di T diventa dominante e lo rimane.

Nell'esempio i candidati sono *$2n^2$* e *$3n$*; questi termini possono essere confrontati usando la disuguaglianza $2n^2>3n$, che vale per ogni $n > 1$.

Istruzione dominante

$$T(n) = 2n^2 + 3n + 1 = O(n^2)$$



Intuitivamente un'istruzione dominante è una di quelle che vengono eseguite più volte tra tutte le istruzioni dell'algoritmo (tipicamente, le istruzioni nei cicli più interni) e da cui dipendono i costi dei termini dominanti.

Per cui per valutare il costo di un algoritmo è sufficiente identificare un'istruzione dominante e valutare il costo di esecuzione di tale istruzione.

Alcuni ordini di grandezza tipici

- *Se $T(n) = O(f(n))$* allora i seguenti sono esempi tipici di f(n):
  - 1
  - $(\log n)^k$
  - $\sqrt{(n)}$
  - n
  - $n(\log n)^k$
  - n^2
  - $n^2(\log n)^k$
  - …
  - a^n