Algoritmi - Lezione 2

Ionut Zbirciog

10 October 2023

1 Modelli di Calcolo

1.1 Macchina di Turing

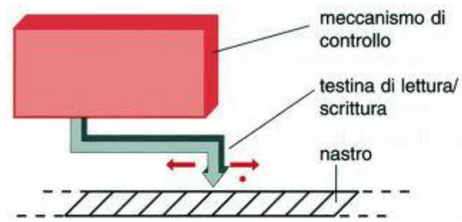


Figure 1: Macchina di Turing.

- Troppo di basso livello.
- Utile per calcoli ma poco efficiente.

1.2 Modello RAM (Random Access Machine)

- Un programma finito.
- Un nastro di input e uno di output.
- Una memoria strutturata come un array.
- Una CPU per eseguire istruzioni.

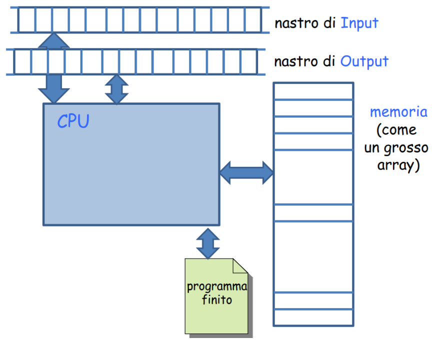


Figure 2: Modello di Von Neumann.

1.3 Analisi della Complessità di Algoritmi

L'analisi della complessità di un algoritmo si basa sul concetto di passo elementare. I passi elementari su una RAM includono:

- Istruzione I/O.
- Operazione aritmetica/logica.
- Accesso/modifica della memoria.

2 Criteri di Costo

2.1 Criterio di Costo Uniforme

È generalmente usato e assume che tutte le operazioni abbiano lo stesso costo. La complessità temporale è misurata come il numero di passi elementari eseguiti.

2.2 Criterio di Costo Logaritmico

Il costo di un'operazione dipende dagli operandi. Un'operazione su un operando x ha costo $\log(x)$. Questo criterio modella meglio la complessità di algoritmi numerici.

3 Caso Peggiore

Si definisce il tempo di esecuzione $T(I)$ di un algoritmo sull'istanza I. $T_{worst}(n)$ rappresenta il tempo di esecuzione sulle istanze di ingresso che comportano più lavoro per l'algoritmo. Questo fornisce una garanzia sul tempo di esecuzione per ogni istanza.
$T_{worst}(n) = \max_{Idimn}\{tempo(I)\}$

4 Caso Medio

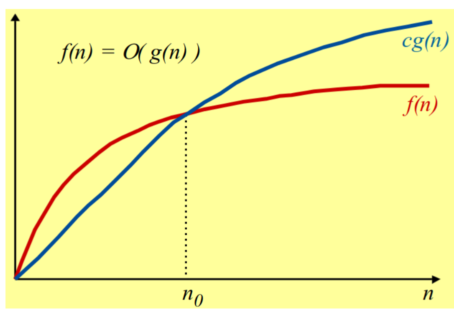
Sia $P(I)$ la probabilità di occorrenza dell'istanza I. $T_{avg}(n)$ rappresenta il tempo di esecuzione nel caso medio, cioè sulle istanze di ingresso tipiche per il problema. Poiché non è sempre possibile conoscere la distribuzione di probabilità sulle istanze, si fanno delle assunzioni.
$T_{avg}(n) = \sum_{I dim n} \{P(I) \cdot tempo(I)\}$

5 Notazione Asintotica

$T(n)$ = numero di passi elementari eseguiti su una RAM nel caso peggire su un'istanza di dimensione n. Si utilizza la notazione asintotica per descrivere la complessità di un algoritmo in modo qualitativo, ignorando costanti moltiplicative e termini di ordine inferiore. L'assunzione implicita è che stiamo considerando istanze di grandi dimensioni.

5.1 Notazione O Grande

Definiamo $f(n) = O(g(n))$ se esistono due costanti $c > 0$ e $n_0 > 0$ tali che $0 \leq f(n) \leq c \cdot g(n)$ per ogni $n \geq n_0$.

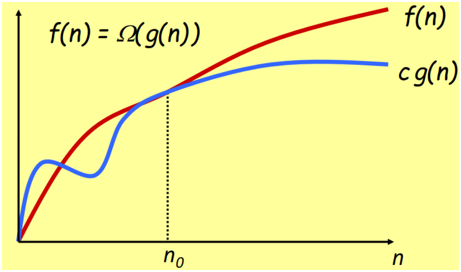


Esempi: f(n) = $2n^2 + 3n$
$f(n) = O(n^3)$
$f(n) = O(n^2)$
$f(n) \neq O(n)$

Nota:
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = O(g(n))$
$f(n) = O(g(n)) \not\Rightarrow \lim_{n\to 0} \frac{f(n)}{g(n)} = 0$
$f(n) = O(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)}$ (se esiste) $< \infty$

5.2 Notazione Omega Grande

Definiamo $f(n) = \Omega(g(n))$ se esistono due costanti $c > 0$ e $n_0 > 0$ tali che $f(n) \geq c \cdot g(n) \geq 0$ per ogni $n \geq n_0$.

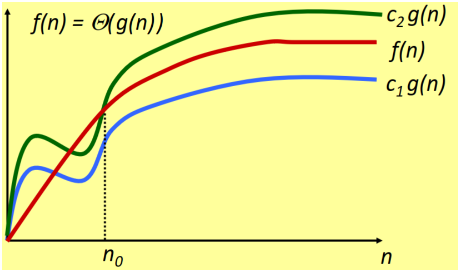


Esempi: $\mathrm{f(n)} = 2\mathrm{n}^2 - 3n$
$f(n) = \Omega(n)$
$f(n) = \Omega(n^2)$
$f(n) \neq \Omega(n^3)$

Nota:
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \Omega(g(n))$
$f(n) = \Omega(g(n)) \not\Rightarrow \lim_{n\to 0} \frac{f(n)}{g(n)} = \infty$
$f(n) = O(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)}$ (se esiste) > 0

5.3 Notazione Theta

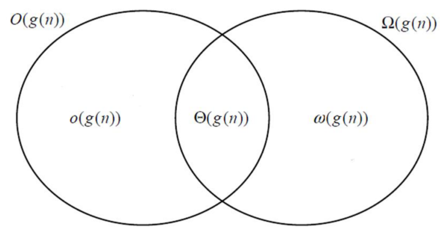
Definiamo $f(n) = \Theta(g(n))$ se esistono tre costanti $c_1 > 0$, $c_2 > 0$, e $n_0 > 0$ tali che $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ per ogni $n \geq n_0$.



Esempi: $\mathrm{f(n)} = 2\mathrm{n}^2 - 3n$
$f(n) = \Theta(n^2)$
$f(n) \neq \Theta(n)$
$f(n) \neq \Theta(n^3)$

Nota:
$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$

$f(n) = O(g(n)) \not\Rightarrow f(n) = \Theta(g(n))$
$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$
$f(n) = \Omega(g(n)) \not\Rightarrow f(n) = \Theta(g(n))$
$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = \Omega(g(n)) \wedge f(n) = O(g(n))$



5.4 Notazione *o* Piccolo

Definiamo $f(n) = o(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
Nota: $o(g(n)) \subset O(g(n))$

5.5 Notazione Omega Piccolo

Definiamo $f(n) = \omega(g(n))$ se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.
Nota: $\omega(g(n)) \subset \Omega(g(n))$

5.6 Alcune Proprietà

- $f(\Omega) = O(g(n))$ se e solo se $g(n) = \Omega(f(n))$.
- $f(n) = o(g(n))$ se e solo se $g(n) = \omega(f(n))$.

6 Alcuni Limiti Notevoli

6.1 Polinomi

Se $P(n) = a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \ldots + a_0$:
P(n) $= \Theta(n^d)$
$P(n) = O(n^d)$
$P(n) = \Omega(n^d)$

6.2 Esponenziali

Se $f(n) = a^n$:
a$^n = \omega(n^d)$
$a^n = \Omega(n^d)$

6.3 Logaritmi

Se $f(n) = \log_b(n)$:
$\log_b(n)^c = o(n^d)$
$\log_b(n)^c = O(n^d)$

6.4 Fattoriali

Se $f(n) = n!$:
n! = o(n^n)
$n! = \omega(a^n)$

7 Velocità delle Funzioni Composte

- Date $f(n)$ e $g(n)$, la velocità della funzione $f(n) + g(n)$ è la velocità della più veloce tra $f(n)$ e $g(n)$.
- Date $f(n)$ e $g(n)$, la velocità della funzione $f(n) \cdot g(n)$ è la velocità di $f(n)$ "più" la velocità di $g(n)$.
- Date $f(n)$ e $g(n)$, la velocità della funzione $f(n)/g(n)$ è la velocità di $f(n)$ "meno" la velocità di $g(n)$.

8 Usare la Notazione Asintotica nelle Analisi

8.1 Analisi della Complessità di fibonacci3

Algorithm 1 fibonacci3

```
1: function FIBONACCI3(intero n) → intero
2:     Sia Fib un array di n interi
3:     Fib[1] = 1; Fib[2] = 1
4:     for i = 3 to n do
5:         Fib[i] = Fib[i − 1] + Fib[i − 2]
6:     end for
7:     return Fib[n]
8: end function
```

$T(n)$ rappresenta la complessità computazionale nel caso peggiore con input n.
c_j rappresenta il numero di passi elementari eseguiti su una RAM quando è eseguita la linea di codice j.

8.2 Upper Bound

- Linee 1, 3, 5 eseguite una sola volta.
- Linee 3 e 4 eseguite al più n volte.

$T(n) \leq c_1 + c_3 + c_5 + (c_3 + c_4) \cdot n = \Theta(n)$
$\Rightarrow T(n) = O(n)$

8.3 Lower Bound

- La linea 4 è eseguita almeno $n - 3$ volte.

$T(n) \geq c_4 \cdot (n - 3) = \Theta(n)$
$\Rightarrow T(n) = \Omega(n)$
$\Rightarrow T(n) = O(n)$ e $T(n) = \Omega(n) \Rightarrow T(n) = \Theta(n)$

9 Perché è una grande idea la notazione asintotica:

- Misura indipendente dall'implementazione dell'algoritmo e dalla macchina reale.
- I dettagli nascosti sono poco rilevanti per n molto grandi.
- Un'analisi dettagliata del numero di passi realmente eseguiti sarebbe difficile, noiosa e non direbbe molto di più.
- Descrive bene in pratica la velocità degli algoritmi.