Capitolo 1.1 - 1.2

Lorenzo Vaccarecci

27 febbraio 2024

1 Appunti vari

Relazione P su $I \times S$:

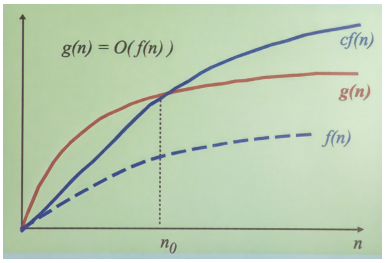
- I insieme degli input
- S insieme delle soluzioni

Problema $\neq$ Algoritmo

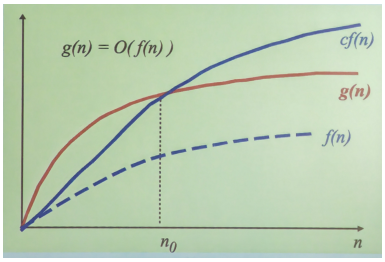
Algoritmo: descrizione precisa e non ambigua di un procedimento di calcolo, che può essere eseguito dall'uomo "meccanicamente" oppure da una macchina.
Correttezza

- A: $I \to S$ parziale (può non terminare)
- $\forall$ input i, $A(i)$ è una soluzione di P(roblema)
- Tecniche:
  - Induzione, per gli algoritmi ricorsivi
  - Invrianti di ciclo, per gli algoritmi iterativi

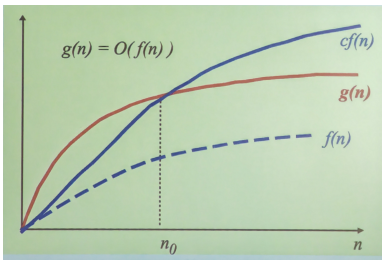
2 Capitolo 1.1



Definizione 1 *Una funzione $g(n)$ appartiene all'insieme $O(f(n))$ (si dice anche $g(n)$ è $O(f(n))$) se esistono due costanti $c > 0$ e $n_0 \geq 0$ tali che $g(n) \leq cf(n)$ per ogni $n \geq n_0$, ossia, da un certo punto in poi, g sta sotto una funzione "multipla" di f ("cresce al più come f").*



Definizione 2 *Una funzione $g(n)$ appartiene all'insieme $\Omega(f(n))$ (si dice anche $g(n)$) è $\Omega(f(n))$ se esistono due costanti $c > 0$ e $n_0 \geq 0$ tali che $g(n) \geq cf(n)$ per ogni $n \geq n_0$, ossia, da un certo punto in poi, f sta sopra una funzione "sottomultipla" di f ("cresce almeno come f")*



Definizione 3 *Una funzione $g(n)$ appartiene all'insieme $\Theta(f(n))$ (si dice anche $g(n)$ è $\Theta(f(n))$) se esistono tre costanti $c_1, c_2 > 0$ e $n_0 \geq 0$ tali che $c_1 f(n) \leq g(n) \leq c_2 f(n)$ per ogni $n \geq n_0$, ossia, da un certo punto in poi, f è compresa tra un "multiplo" di f e un "sottomultiplo" di f ("cresce come f")*

Per comodità si usa anche scrivere $f(n) = O(g(n))$, e analogamente per le altre notazioni. Il simbolo $=$ in questo caso non denota un'uguaglianza, ma l'appartenenza a un insieme.

Proprietà della notazione asintotica

Transitiva

- $f(n)$ è $O(g(n))$ e $g(n)$ è $O(h(n))$ implica $f(n)$ è $O(h(n))$
- $f(n)$ è $\Omega(g(n))$ e $g(n)$ è $\Omega(h(n))$ implica $f(n)$ è $\Omega(h(n))$
- $f(n)$ è $\Theta(g(n))$ e $g(n)$ è $\Theta(h(n))$ implica $f(n)$ è $\Theta(h(n))$

Riflessiva

- $f(n)$ è $O(f(n))$
- $f(n)$ è $\Omega(f(n))$
- $f(n)$ è $\Theta(f(n))$

Simmetrica

$f(n)$ è $\Theta(g(n))$ se e solo se $g(n)$ è $\Theta(f(n))$

Simmetrica Trasposta

$f(n)$ è $O(g(n))$ se e solo se $g(n)$ è $\Omega(f(n))$

Comportamenti notevoli Intrattabili

- $\Theta(2^n)$ esponenziale
- $\Theta(n!)$ fattoriale
- $\Theta(n^n)$ esponenziale in base n

3 Capitolo 1.2

Formalmente, se i è un input, siano $t(i)$ e $s(i)$ il tempo di esecuzione e lo spazio di memoria necessario (in aggiunta allo spazio occupato dall'input stesso) per l'esecuzione dell'algoritmo per l'input i. Le seguenti funzioni risultno allora ben definite:

- **complessità temporale del caso peggiore**: $T_{worst}(n) = \max\{t(i)|i \text{ ha dimensione } n\}$
- **complessità temporale del caso migliore**: $T_{best}(n) = \min\{t(i)|i \text{ ha dimensione } n\}$
- **complessità temporale del caso medio**: $T_{avg}(n) = \text{avg}\{t(i)|i \text{ ha dimensione } n\}$

Per il caso medio, per ogni n si considerano tutti i possibili input di dimensione n, siano $i_1, \ldots, i_N$, e si fa la media aritmetica dei tempi:

$$T_{avg}(n) = \frac{t(i_1) + \cdots + t(i_N)}{N} \tag{1}$$

Usiamo la media pesata se i possibili input non sono equiprobabili.
Vi sono quindi nove possibili affermazioni sulla complessità di un algoritmo, non tutte però indipendenti tra loro nè ugualmente interessanti. In particolare, si ha che se $T_{worst}(n) = O(f(n))$, allora anche $T_{avg}(n) = O(f(n))$ e $T_{best}(n) = O(f(n))$, e simmetricamente se $T_{best}(n) = \Omega(f(n))$, allora anche $T_{avg}(n) = \Omega(f(n))$ e $T_{worst}(n) = \Omega(f(n))$.

- un algoritmo ha complessità $O(f(n))$ se $T_{worst}(n) = O(f(n))$. Ossia $f(n)$ è una delimitazione superiore del tempo di calcolo: al crescere di n il tempo di calcolo non cresce più di $f(n)$, qualunque sia l'input.
- un algoritmo ha complessità $\Omega(f(n))$ se $T_{best}(n) = \Omega(f(n))$. Ossia $f(n)$ è una delimitazione inferiore del tempo di calcolo: al crescere di n il tempo di calcolo cresce almeno come $f(n)$, qualunque sia l'input.
- un algoritmo ha complessità $\Theta(f(n))$ se ha complessità $O(f(n))$ e $\Omega(f(n))$. Ossia, $f(n)$ è una delimitazione sia inferiore che superiore del tempo di calcolo, qualunque sia l'input.

Il caso migliore è di solito scarsamente interessante, poichè riguarda input molto particolari. Il caso medio è significativo solo se la distribuzione delle probabilità riflette la situazione reale di uso dell'algoritmo. Il caso peggiore è la complessità che si studia di solito, poiché è l'unica che fornisce la garanzia che in ogni caso il tempo di esecuzione non sarà maggiore di un certo tempo prevedibile.
Nel caso di un algoritmo randomizzato, possiamo calcolare la media (pesata) dei possibili tempi di computazione per *uno stesso input (expected running time)*:

$$T_{exp}(i) = \mathbb{E}[t(i, c)] \tag{2}$$

worst case expected running time:

$$T_{exp\_worst}(n) = \max\{T_{exp}(i) | i \text{ ha dimensione } n\} \tag{3}$$

Nel confronto precedente abbiamo assunto che l'algoritmo randomizzato fornisca sempre il risultato corretto (algoritmi Las Vegas). Alcuni tipi di algoritmi randomizzati (Monte Carlo) possono anche fornire un risultato errato (forniscono comunque il risultato giusto con probabilità > 0). In questo caso la ripetizione dell'esecuzione permette di limitare la probabilità di sbagliare. Il vantaggio si ha tipicamente in casi in cui l'algoritmo deterministico è molto inefficiente.

Complessità dei problemi

- **delimitazione superiore** *Un problema ha complessità* $O(f(n))$ se esiste un algoritmo di complessità $O(f(n))$ che lo risolve. Ossia, è possibile risolvere il problema in un tempo che cresca non più di $f(n)$.
- **delimitazione inferiore** *Un problema ha complessità* $\Omega(f(n))$ se tutti i possibili algoritmi risolventi hanno complessità $\Omega(f(n))$. Ossia, non è possibile risolvere il problema in un tempo che cresca meno di $f(n)$.

Quindi, per trovare una delimitazione superiore $f(n)$ alla complessità del problema, è sufficiente (e necessario) trovare *un* algoritmo che risolva il problema in un tempo $O(f(n))$, mentre, per trovare una delimitazione inferiore $g(n)$, è necessario dimostrare che *qualunque* possibile algoritmo deve impiegare un tempo $\Omega(g(n))$. Non basta quindi che tutti gli algoritmi conosciuti abbiano complessità $\Omega(g(n))$.
Un problema algoritmico, può essere *aperto (o con gap algoritmico)* o *chiuso*. Un problema aperto può venire chiuso, ma non viceversa. Un problema è chiuso se si conoscono limite superiore e inferiore coincidenti, ossia:

- esiste un algoritmo risolvente di complessità $O(f(n))$
- si è dimostrato che qualunque algoritmo risolvente deve avere complessità $\Omega(f(n))$, ossia non può esistere un algoritmo di complessità inferiore a $\Omega(f(n))$.

Si è quindi dimostrato che l'algoritmo risolvente è **ottimo**. Saranno possibili solo miglioramenti marginali, per esempio per una costante additiva o moltiplicativa.
Un problema è aperto se (tutte le) delimitazioni inferiori e superiori differiscono, ossia:

- il miglior algoritmo risolvente noto ha complessità $O(f(n))$

Un gap algoritmico può essere chiuso in due modi:

- dal di sopra: si trova un algoritmo migliore, abbassando così il limite superiore; se si trova un algoritmo di complessità coincidente con il limite inferiore, tale algoritmo è ottimo e il problema è chiuso.
- dal di sotto: si riesce a dimostrare un limite inferiore più alto; se questo coincide con la complessità dell'algoritmo migliore esistente, si è dimostrato che l'algoritmo è ottimo e il problema è chiuso.

Non tutti i problemi sono risolubili