ALGORITMI E COMPLESSITÀ (LM)

PAOLO BOLDI

Anno accademico 2024/25

Note di Mauro Tellaroli

Indice

1 Introduzione

1.1 Problemi

Definizione 1 (Problema)**.** Un problema π é una tripla $(I_\pi, O_\pi, \text{Sol})$ dove:

- I_π é insieme dei possibili input
- O_π é insieme dei possibili output
- $\text{Sol}_\pi : I_\pi \to 2^{O_\pi} \setminus \{\varnothing\}$ é la funzione che preso l'input restituisce gli output corretti

Esempio: decidere se un numero é primo:

- $I_\pi = \mathbb{N}$
- $O_\pi = \{\text{sí}, \text{no}\}$

Lemma 1 (Tesi di Church Turing)**.** La tesi di Church Turing afferma che la classe delle funzioni calcolabile coincide con quella delle funzioni calcolabili da una macchina di Turing.

Da questo ne deriva che qualsiasi tipo di macchina potrá calcolare solo le stesse funzioni che calcola una macchina di Turing. Una macchina quantistica, per esempio, a livello di calcolo é equivalente a una di Turing (ma non a livello di tempo dove si ha la quantum supremacy).

Definizione 2 (Algoritmo)**.** Un algoritmo per il problema π é una macchina di Turing che preso un input $x \in I_\pi$ restituisce in output $y \in O_\pi$ tale che $y \in \text{Sol}_\pi(x)$.

Tutti i problemi sono risolvibili? La risposta é no. Una motivazione intuitiva é che un qualsiasi algoritmo puó essere rappresentato da una stringa binaria; l'insieme di tutti i possibili algoritmi avrá quindi la stessa cardinalitá di $2^* \sim \mathbb{N}$.

Tutte le funzioni soluzione dei problemi di decisione hanno la stessa cardinalitá di $2^{2^*} \sim \mathbb{R}$.

É noto che $\mathbb{N} \nsim \mathbb{R}$ e quindi esistono delle funzioni soluzione che non possono essere calcolate da un algoritmo.

1.2 Teoria della complessità

Si considerino, da questo momento in poi, solo i problemi risolvibili. La domanda su cui la teoria della complessità si fonda é: quanto efficaciemente si riesce a risolvere un problema?

Ci sono due branche della teoria della complessità, algoritmica e strutturale.

1.2.1 Teoria della complessità algoritmica e strutturale

La **teoria della complessità algoritmica** si chiede quante risorse servono ad un algoritmo per risolvere un problema. Le risorse possono essere:

- il tempo, contato tipicamente come numero di passi;
- lo spazio di memoria;
- energia dissipata;
- numero di CPU nel punto di carico massimo.

Si userá quasi sempre il tempo. Sia T_A la funzione tempo:

$$T_A : I_\pi \to \mathbb{N}$$

che, per ogni input, indica quanto ci mette l'algoritmo A a terminare. Il calcolo di T_A é molto scomodo e difficile. Si userá infatti la funzione:

$$t_A : \mathbb{N} \to \mathbb{N}$$

definita come segue:

$$t_A(n) = \max\{T_A(x) : x \in I_\pi \wedge |x| = n\}$$

che applica la filosofia worst case per ogni lunghezza n di input. Si prenderá quindi il caso peggiore di tempo impiegato dall'algoritmo A su tutti gli input lunghi n.

Dall'altra parte, fissato un problema π, la **teoria della complessità strutturale** si chiede quale sia la complessità del problema stesso e non di uno specifico algoritmo che lo risolve.

1.2.2 Limiti superiori e inferiori

Per entrambe le teorie si userá la complessità asintotica ovvero si studierá come si comporta l'algoritmo in relazione alla lunghezza n dell'input. Si useranno quindi i simboli di Landau O e Ω per indicare rispettivamente:

- Upper bound: si cerca un algoritmo che risolve il problema; la sua complessità fará da limite superiore del problema, o in altre parole, avendo trovato un algoritmo che funziona in un determinato tempo si potrá al massimo trovarne altri piú efficienti. Per indicare l'upper bound si userá la notazione $O(f(n))$. Si cercherá di abbassare questo limite.
- Lower bound: si dimostra che il problema non puó essere risolto in meno di $f(n)$ risorse e si indica con $\Omega(f(n))$. Si cercherá di alzare questo limite.

Se si riesce a far coincidere l'upper bound e il lower bound vuol dire che si é trovato un algoritmo che risolve il problema nella maniera piú efficiente possibile. Questa situazione si indica con $\Theta(f(n))$. Purtroppo questo caso é raro e tipicamente resta un buon gap tra i due limiti. Un esempio puó essere l'ordinamento di array che si é trovato essere $\Theta(n \log n)$.

I casi piú interessanti si trovano in una zona grigia dove l'upper bound é esponenziale e il lower bound polinomiale. In questo caso non si sa se esistono algoritmi efficienti (quindi polinomiali) che risolvono il problema.

2 Problemi di decisione

Tutti i problemi dove l'output é binario ($O_\pi = \{0, 1\}$) sono problemi di decisione. Si avrá quindi una sola risposta possibile, o sí o no; degli esempi possono essere stabilire se un numero é primo o stabilire se una formula logica é soddisfacibile.

2.1 Classi di complessità

Si prenderanno in esame delle classi di problemi in modo da non dover studiare ogni problema singolarmente. Le classi piú note sono P e NP:

- P é la classe dei problemi che possono essere risolti in tempo polinomiale. Chiedersi se un problema appartiene a P equivale a chiedersi se puó essere risolto in maniere efficiente.
- NP é la classe dei problemi risolvibili in tempo polinomiale da una macchina non deterministica.

Una **macchina non deterministica** è un modello teorico di calcolatore che, in ogni stato di computazione, può scegliere tra molteplici possibili transizioni. In altre parole, può esplorare simultaneamente tutte le possibili scelte a ogni passo computazionale, come se disponesse di infinite risorse di calcolo parallele. Se almeno una delle ramificazioni porta a una soluzione accettante, la macchina accetta l'input, altrimenti lo rifiuta (ovvero da come output sí o no). Questo tipo di macchina é puramente teorico.

Molti problemi noti non si sa se appartengono a P ma sicuramente appartengono a NP. Il piú famoso é il problema CNF-SAT.

2.1.1 CNF-SAT

CNF-SAT prende in input una formula logica φ in forma normale congiunta del tipo:

$$\varphi = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_5) \wedge (\neg x_4 \vee x_3)$$

Scopo del problema é determinare se φ é soddisfacibile ovvero se esiste un'assegnamento delle variabili x_i tale che φ sia vera.

In una macchina non deterministica questo problema risulta banale: creo un albero di computazione per ogni possibile assegnamento di variabile e in ognuno controllo se la formula é soddisfatta o meno. In ogni ramo di calcolo basterá controllare le n variabili. Il tempo é quindi polinomiale in una macchina non deterministica e quindi CNF-SAT appartiene a NP.