

Fondamenti di Informatica

per la Sicurezza

a.a. 2004/05

Teoria della complessità

Stefano Ferrari



Università degli Studi di Milano
Dipartimento di Tecnologie
dell'Informazione

Teoria della complessità

Nella pratica dato un problema computabile:

- bisogna sapere come calcolare la soluzione (algoritmo);
- bisogna sapere scegliere la soluzione meno onerosa.

La teoria della computazione non tiene conto dell'efficienza: ogni computazione è istantanea perché ogni passo di computazione è tale.

La **teoria della complessità** studia le risorse computazionali necessarie per l'esecuzione di un algoritmo.

Complessità

La complessità di un algoritmo è definibile principalmente in due modi:

- come il numero di operazioni che esso deve fare per giungere alla soluzione (**complessità temporale**);
- come il numero di celle di memoria necessarie per il calcolo (**complessità spaziale**).

È evidente che in entrambi i casi la misura della complessità dipenda da:

- l'agente di calcolo impiegato;
- la dimensione del dato di input.

Misura di complessità

Il paradigma di calcolo di riferimento è la MdT.

Tipicamente, l'attenzione è posta sulle misure di complessità temporale:

- la complessità spaziale non può superare quella temporale;
- nella pratica, generalmente lo spazio è una risorsa meno preziosa del tempo;
 - però, per esempio, a volte è utile usare la compressione dei dati!

Misura di complessità

La complessità viene misurata con la notazione $O(\cdot)$, in relazione alla dimensione dei dati iniziali.

Tale notazione indica il comportamento **asintotico** rispetto alla dimensione dell'input.

Per esempio, se gli algoritmi A_1 , A_2 e A_3 sono, rispettivamente, $O(n)$, $O(n^2)$ e $O(2^n)$, sono di complessità, lineare, quadratica ed esponenziale.

Perciò, se per elaborare una lista di 10 nomi impiegano tutti 1 s, dobbiamo aspettarci che per 30 nomi A_1 impieghi 3 s, A_2 9 s e A_3 2^{20} s.

Classificazione dei problemi

Ci sono infinite classi di complessità:

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < \\ < O(n\sqrt{n}) < O(n^2) < O(n^3) < O(2^n) < O(n2^n) < \dots$$

Però, la classificazione principale è in problemi:

- **trattabili**: $O(n^k)$, per qualche k ;
- **intrattabili**: $O(f(n))$, con $f(n) > n^k \forall k, n \rightarrow \infty$.

La suddivisione è arbitraria, ma ben motivata:

- a. l'insensibilità degli algoritmi $O(k^n)$ al progresso;
- b. l'invarianza delle classi rispetto ai differenti paradigmi di calcolo.

Problemi intrattabili

- a.** A fronte di un progresso tecnologico che renda i calcolatori 1000 volte più veloci, le nuove dimensioni dei problemi risolubili in 1 s dagli algoritmi A_1 , A_2 e A_3 precedenti, sarebbero:
- A_1 , $O(n)$: 10 000 nomi;
 - A_2 , $O(n^2)$: 100 nomi;
 - A_3 , $O(2^n)$: 20 nomi.
- b.** La complessità di conversione di una MdT in un paradigma di calcolo equivalente è polinomiale.
- Anche cambiando il paradigma di riferimento la trattabilità di un algoritmo non può cambiare.

Problemi \mathcal{P} e \mathcal{NP}

La trattabilità dei problemi si formalizza con le classi di problemi \mathcal{P} e \mathcal{NP} :

- \mathcal{P} : classe dei problemi risolvibili in tempo **polinomiale** da una MdT **deterministica**;
- \mathcal{NP} : classe dei problemi risolvibili in tempo **polinomiale** da una MdT **non-deterministica**.

P vs NP problem

Uno dei problemi aperti dell'informatica è:

$$\mathcal{P} \neq \mathcal{NP}?$$

È uno dei sette **Millennium Problems** posti dal Clay Mathematics Institute (USA):

- un premio da un milione di dollari a chi lo risolve;
- <http://www.claymath.org/millennium/>

Esistono problemi detti **\mathcal{NP} -completi** a cui possono essere ricondotti gli altri problemi \mathcal{NP} .

Dimostrare che un problema \mathcal{NP} -completo è in \mathcal{P} implicherebbe $\mathcal{P} = \mathcal{NP}$.