

Università “Ca’Foscari” di Venezia

Dipartimento di Matematica Applicata

Giovanni Fasano[†]

Cenni sulla Complessità Computazionale

[†]Università Ca’Foscari di Venezia, Dipartimento di Matematica Applicata, Ca’Dolfin Dorsoduro 3825/E, 30123 Venezia, ITALY. E-mail: fasano@unive.it ; URL: www.dma.unive.it/~fasano - A.A. 2009-2010.

1 Cenni di teoria della Complessità

Per quanto riguarda l'aspetto bibliografico, i contenuti di quanto segue sono in buona parte presenti in [GaJo79] e [St92]; riferimenti più specifici verranno citati nel prosieguo. Come di frequente accade, dato un generico problema, per definire in senso classico le basi della teoria della complessità, vengono introdotti alcuni “modelli logici” di funzionamento per un solutore di tale problema. Sostanzialmente si definiscono le operazioni “elementari” che tale solutore è in grado di compiere, si codifica l'istanza del problema in oggetto secondo “un” linguaggio che il solutore è in grado di comprendere, dopodiché si definisce una misura dello “sforzo computazionale” (numero operazioni elementari, lunghezza operandi prodotti, tempo di calcolo, etc.) che il miglior solutore sopporta, per trovare almeno una soluzione (ammesso che esista) del problema in oggetto. Successivamente si suddividono i problemi, risolubili come sopra, in opportune classi, sulla base di un criterio di “omogeneità” al quale i problemi di ciascuna classe devono rispondere.

Anche noi seguiamo tale schema, limitandoci a considerare il modello di complessità basato sulla **Macchina di Turing (MdT)** ed il **Modello Aritmetico (MA)**.

In breve il primo modello permette di formalizzare l'istanza di un problema, mediante una codifica arbitraria di quest'ultima, sotto forma di stringa di caratteri. La computazione dell'istanza, avviene analizzando in sequenza ciascuno dei caratteri di tale stringa, manipolandolo poi con operazioni elementari come la cancellazione, la sostituzione con altro carattere, etc.. La MdT ha un comportamento *sequenziale* nel senso che è in grado di “memorizzare” uno stato e quindi simulare, almeno in senso astratto, anche la dinamica di un flip-flop (cella elementare di memoria in un elaboratore).

Proprio l'elementarità delle operazioni che la MdT è in grado di svolgere, costituisce però un limite per le proprie prestazioni. Visto che le attuali macchine di calcolo possono eseguire operazioni decisamente più “evolute”, si è pensato che tale modello potesse essere inadeguato per l'analisi, la descrizione e la soluzione di molti problemi.

Nasce anche da questo la necessità di adottare il MA. È questo un modello che, come indice di valutazione del grado di difficoltà, per risolvere l'istanza di un problema, considera una stima del numero di operazioni aritmetiche (essenzialmente moltiplicazioni e divisioni) necessarie per trovare la soluzione al problema dato. Più in dettaglio, anche qui si considera una stima, relativa all'algoritmo più “efficiente” (minor numero di operazioni) che è in grado di risolvere il problema.

Anche nell'ambito di questo modello vengono definite alcune “classi di complessità”, così che in ciascuna di queste, i problemi contenuti vengono considerati *equivalentemente* onerosi. Purtroppo, le classi identificate rispettivamente dal modello basato sulla MdT e dal MA, non coincidono ed anzi, in generale possono differire anche sostanzialmente: si darà tra breve un esempio, introducendo alcune tra le categorie della complessità più note ed utilizzate in letteratura.

1.1 La DTM a nastro singolo, la NDTM e la RAM

Come noto dalla letteratura, la **DTM** (Deterministic Turing Machine) è una MdT che è in grado di riconoscere un *linguaggio* (il sottoinsieme di stringhe che codificano le istanze), associato ad un particolare problema. Il riconoscimento avviene producendo un appropriato output; inoltre se in aggiunta a tale riconoscimento, la DTM compie un *numero limitato di operazioni* per ogni stringa in ingresso, allora si dice che essa fornisce un **algoritmo** per la risoluzione del problema.

Macchina di Turing, Modello Aritmetico

Deterministic Turing Machine

Una **NDTM** (NonDeterministic Turing Machine) in linea di principio contiene al proprio interno una DTM, inoltre riceve anch'essa in ingresso stringhe appartenenti ad un linguaggio, ma presenta una novità saliente: ogni stringa σ in ingresso, deve essere corredata di un **certificato**, ovvero di una stringa aggiuntiva **c**. La NDTM sarà in grado di riconoscere (nel senso specificato per una DTM) la stringa σ solo se in ingresso sarà presente la coppia (σ, \mathbf{c}) , essendo **c** “un” opportuno certificato associato a σ . Per esempio, se sul nastro fosse scritta una sequenza di 16 numeri compresi tra 0 e 9, la NDTM non sarebbe senz'altro in grado di comprendere ed elaborare la sequenza. Se invece alla sequenza di 16 numeri viene anche associato il certificato

*NonDeterministic
Turing Machine*

$$\mathbf{c} = \text{“carta di credito”},$$

la NDTM sarebbe in grado di capire il significato della sequenza e di conseguenza sarebbe in grado di processarla.

Nel caso in cui la NDTM riconosca un linguaggio \mathcal{L} , si dirà *comunque* che essa fornisce un algoritmo per la risoluzione del problema di cui \mathcal{L} è il linguaggio associato.

Passando ora al MA, uno strumento comunemente adottato per descrivere l'approccio mediante modello aritmetico, è la **RAM** (Random Access Machine). In sintesi è un dispositivo logico che, come la DTM e la NDTM, in input legge stringhe di caratteri (costruite sulla base di una codifica di un'istanza). Dopo averle memorizzate, le manipola mediante operazioni di tipo aritmetico, assegnazione, trasferimento da/verso registri, etc. ed infine scrive l'output in una cella di memoria. È evidente quindi che, rispetto alle DTM e NDTM, la RAM svolge operazioni più evolute: naturalmente uno studio della complessità dovrà tener conto che “comunque” *per svolgere ciascuna di tali operazioni più articolate, si dovrà usare una sequenza di operazioni più elementari*, vicine a quelle proprie di una DTM o una NDTM.

RAM

1.2 Le classi P e strongly-P

Finora si è accennato a due modelli descrittivi per risolvere (riconoscere) problemi (linguaggi associati a problemi). Cerchiamo ora di quantificare la *mole di calcolo* richiesta, in entrambi i modelli, per produrre in output una soluzione per un'istanza. Come parametro di valutazione, nel caso di una DTM si considera il numero di spostamenti che un elemento fisico interno alla macchina stessa (la **testina**) compie per completare la computazione della stringa σ (Figura 1). Nel caso invece di una RAM, l'indice di valutazione è il numero di operazioni (più spesso le sole **operazioni aritmetiche**) svolte dalla macchina per terminare l'elaborazione di σ . Più comunemente tali indici di valutazione vengono denominati **tempo di computazione (TdC(\cdot))** di σ e come è ragionevole ritenere, in generale sono una funzione non decrescente delle dimensioni del problema, cioè, più esattamente, della lunghezza della stringa di input. Ora, detta n la lunghezza della stringa σ , se il tempo di computazione verifica la:

*Tempo di
Computazione*

$$\text{TdC}(\sigma) \leq k \cdot p(n) \quad k \in \mathbb{R}^+$$

con $p(n) \in P_m[n]$ (anello dei polinomi di grado $\leq m$, ad elementi reali, nella variabile n), i.e. $\text{TdC}(\sigma) = \mathcal{O}(p(n))$, si dice che la DTM (RAM) implementa un algoritmo con **complessità polinomiale**. Di conseguenza, se il miglior algoritmo conosciuto “risolve” il problema Π con complessità polinomiale, il problema Π stesso si dirà a *complessità polinomiale*. L'insieme di tutti gli algoritmi che verificano quest'ultima proprietà, viene

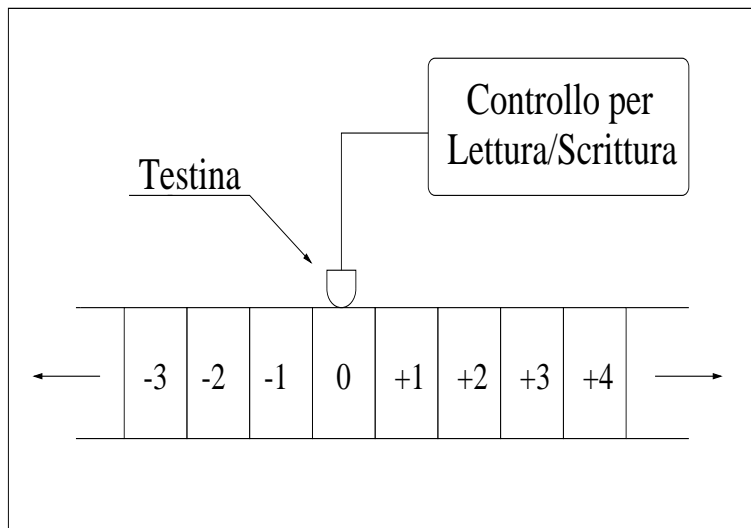


Figura 1: La DTM a nastro singolo.

indicato in letteratura come classe **P**.

Come è ragionevole attendersi, sebbene la complessità computazionale delle operazioni aritmetiche, su una DTM, sia polinomiale, può accadere che il problema Π sia a complessità polinomiale rispetto al MA, ma non rispetto al modello basato sulla MdT e viceversa. Ciò lo si deve sostanzialmente alla possibilità che la *lunghezza degli operandi*, cresca tra un'operazione e la successiva: per esempio il calcolo di a^n richiede $n - 1$ operazioni aritmetiche per il MA, mentre per il modello con MdT ne richiede un numero che cresce esponenzialmente con la variabile n . Per un problema la cui complessità computazionale è polinomiale per entrambi i modelli, si dice che appartiene alla classe dei problemi **strongly-P**.

La classe P

La classe Strongly - P

1.3 Riducibilità polinomiale, le classi NP

Come si è accennato nel Paragrafo 1.1, per il funzionamento di una NDTM si richiede una stringa aggiuntiva data dal certificato, cosa invece non richiesta ad una DTM. Questo fatto consente di poter generalizzare le conclusioni della sezione precedente, nel seguente modo: introduciamo la classe **NP** dei problemi che, con complessità computazionale *polinomiale*, possono essere risolti da un algoritmo implementato con una NDTM. In buona sostanza, nella classe **NP** sono presenti non solo tutti i problemi $\Pi \in \mathbf{P}$ (basta ignorare il certificato), ma anche molti altri problemi con complessità polinomiale “rispetto ad un criterio più ampio”. Come risulta intuitivo in base alla stessa definizione, se $\Pi \notin \mathbf{P}$ ma $\Pi \in \mathbf{NP}$, la necessità di dover calcolare un certificato, comporta uno sforzo di calcolo aggiuntivo. La cosa suggerisce la seguente deduzione: se $\Pi' \in \mathbf{P}$, risolvere Π *non è più facile* che risolvere Π' *. Si è usato il termine ‘suggerire’ in quanto, in base alle rispettive definizioni, la distinzione tra le classi **P** ed **NP** è un fatto qualitativo; se in alternativa si fosse richiesta la verifica di un criterio quantitativo, si sarebbe potuta verificare l'incongruenza di dover classificare: alcune istanze di un problema in una classe ed alcune in un'altra, seppur riferite al medesimo problema. La classificazione data ha quindi il duplice merito di voler:

La classe NP

- (i) identificare la difficoltà intrinseca di un problema

*Ciò non implica che a priori verranno comunque impiegate maggiori risorse di calcolo nel risolvere Π , quanto piuttosto che c'è una ragionevole aspettativa che ciò possa accadere.

(ii) valutare tale difficoltà relativamente agli altri problemi.

Per valutare in maniera più completa la relazione tra un problema e l'altro, si introduce la definizione di **riducibilità polinomiale** come segue: il problema Π è polinomialmente riducibile al problema Π' ($\Pi \propto \Pi'$) se e solo se ogni stringa σ , del linguaggio associato a Π , può essere trasformata in tempo polinomiale [†], in una stringa del linguaggio associato a Π' . Pur rimanendo in un ambito molto generale, sottolineiamo che di criteri di riducibilità ve n'è più di uno in letteratura, inoltre sono spesso legati al modello di riferimento, scelto per descrivere la teoria della complessità stessa. Per esempio, considerando una RAM anziché una DTM, si parla di **NC-riducibilità** del problema Π' dal problema Π , se esistono due costanti $a, b \in \mathbb{R}^+$ t.c.: qualunque stringa σ di lunghezza n del linguaggio associato a Π , può essere trasformata da un vettore di $\mathcal{O}(n^b)$ macchine RAM, in un tempo $\mathcal{O}(\log^a(n))$, in una stringa del linguaggio associato a Π' [‡].

riducibilità polinomiale

Ritornando al modello della MdT, la riducibilità polinomiale offre uno strumento per discriminare, all'interno di classi di complessità, alcune sottoclassi. Naturalmente non permetterebbe, per come è stata definita, di “distinguere” due problemi appartenenti alla classe **P**; tuttavia consente di introdurre la classe dei problemi **NP-complete** come segue. Se:

(1) $\Pi' \in \mathbf{NP}$

(2) $\Pi \propto \Pi' \quad \forall \Pi \in \mathbf{NP}$

allora si dice che $\Pi' \in \mathbf{NP-complete}$. Naturalmente le classi **P** ed **NP-complete** sono disgiunte (in quanto si richiede che la proprietà (2) valga $\forall \Pi \in \mathbf{NP}$) pur essendo entrambe contenute in **NP** (Figura 2).

La classe NP – complete

Molti sono i problemi che appartengono alla classe appena definita, tanto che anche al suo interno si cerca di operare ulteriori distinzioni tra i diversi elementi; comunque una trattazione completa dell'argomento, va oltre i propositi del presente lavoro di ‘accenno’, alla teoria della complessità [§]. Ma per descrivere la complessità associata ai problemi concavi, è necessario introdurre due ulteriori classi assai note in letteratura: le classi dei problemi **NP-hard** ed **NP-easy**.

Per definire la prima di queste classi, in modo completo, sarebbe opportuno analizzare preventivamente un ulteriore strumento della teoria della complessità: l'*oracolo* [GaJo79]; come detto pocanzi, questo ci allontanerebbe dal proposito iniziale di studiare i problemi concavi, daremo quindi per nota quest'ultima nozione. Aggiungiamo però che è possibile “esemplificare” il concetto di ‘oracolo’ con quello più immediato di ‘subroutine’ e, mediante questa equivalenza, definire una macchina di Turing con oracolo (OTM). Si tratta di un dispositivo del tutto simile alla DTM, da cui si differenzia per il solo fatto di includere, tra le operazioni elementari che è in grado di svolgere, anche “chiamate a subroutine”; quindi detta x una generica stringa ed $f(\cdot)$ la funzione implementata dalla subroutine (oracolo), in un solo passo la OTM è in grado di calcolare $f(x)$. Questo generalizza il concetto di *polinomiale riducibilità* dato nella Sezione 1.3, estendendolo a quello di **Turing riducibilità** per i problemi Π e Π' [¶]. Si dirà quindi che Π è Turing riducibile a Π' ($\Pi \propto_T \Pi'$) se, esiste una OTM il cui oracolo è in grado di risolvere Π' in un tempo di

Le classi NP – hard ed NP – easy

[†] Ovvero mediante una DTM che implementa un algoritmo appartenente alla classe **P**.

[‡] In maniera del tutto analoga alla riducibilità polinomiale, si usa per la **NC-riducibilità** il simbolismo $\Pi \propto_{NC} \Pi'$.

[§] Si veda anche [Ma92].

[¶] Si sottolinea che questa estensione si riferisce in genere alla categoria dei *problemi di decisione*.

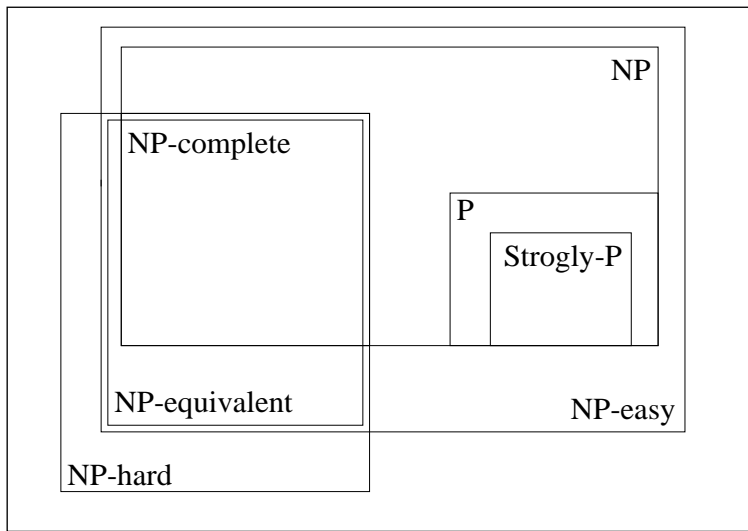


Figura 2: Le principali classi di complessità.

computazione polinomiale. La generalizzazione può essere spiegata nei seguenti termini: la relazione $\Pi \propto_T \Pi'$ suggerisce che esiste un algoritmo \mathcal{A} in grado di risolvere Π in tempo polinomiale se, all'interno di \mathcal{A} vi è una chiamata alla subroutine \mathcal{S} la quale risolve Π' in tempo polinomiale. La non minore complessità di Π' rispetto a Π , risiede nel fatto che la polinomialità di \mathcal{A} dipende “solo” dalla polinomialità di \mathcal{S} , ovvero del livello di difficoltà del problema Π' . Diremo poi che il problema Π' appartiene alla classe **NP-hard** se:

$$\exists \Pi \in \mathbf{NP} - \mathbf{complete} \text{ t.c. } \Pi \propto_T \Pi'$$

Dalla definizione stessa si comprende la denominazione dei problemi afferenti a questa classe; un problema **NP-hard** è in generale *non più facile* di almeno un problema **NP-complete**, inoltre può appartenere o meno alla classe **NP**. È comunque fondamentale sottolineare come la classe dei problemi **NP-hard** *generalizzi* la classe **NP-complete**, in quanto appunto:

$$\begin{aligned} \Pi \in \mathbf{NP} - \mathbf{complete} &\implies \Pi \in \mathbf{NP} - \mathbf{hard} \\ \Pi \in \mathbf{NP} - \mathbf{hard} &\not\Rightarrow \Pi \in \mathbf{NP} - \mathbf{complete} \end{aligned}$$

Per quanto riguarda invece la classe **NP-easy**, essa può definirsi in modo “trasversale”, nel senso che un problema afferente ad una qualsiasi delle classi di complessità fin qui introdotte, può appartenere alla classe dei problemi **NP-easy**. Difatti il problema Π è **NP-easy** se:

$$\exists \Pi' \in \mathbf{NP} \text{ t.c. } \Pi \propto_T \Pi'$$

quindi in questo caso il problema Π è in generale *non più difficile* di almeno un problema **NP** e come per i problemi **NP-hard** può non appartenere alla classe **NP**. I problemi che al contempo appartengono alla classe **NP-hard** ed alla classe **NP-easy** formano la classe dei problemi **NP-equivalent**: sottolineiamo per inciso che quelli tra questi ultimi, che appartengono ad **NP**, si trovano necessariamente nella classe **NP-complete**. La Figura 2 dovrebbe riassumere, per quanto possibile, le principali considerazioni ed i risultati fin qui esposti.

La classe NP – equivalent

References

- [GaJo79] M.R.GAREY,D.S.JOHNSON (1979) “*Computers and Intractability - A Guide to the Theory of NP-completeness*”, W.H.Freeman and Company, New York.
- [St92] L.J.STOCKMEYER (1992) “*Handbook in Operations Research*”, Elsevier Science Publishers, vol. XX, pp. 455-517.
- [Ma92] F.MAFFIOLI (1992) “*Problemi più o meno Trattabili: un’Introduzione alla complessità Computazionale*”, in ‘Metodi di ottimizzazione per le decisioni’, atti della Scuola CIRO, pp. 191-214.