# Appunti Dati e Algoritmi 2

May 31, 2018

# Contents

# 1 Teoria dell'NP-Completezza

## 1.1 Introduzione

Le *classi di complessità* sono insiemi di problemi. Le principali classi di complessità note sono:

1. problemi che ammettono algoritmi *efficienti* (ovvero polinomiali, di qualunque grado);
2. problemi che ammettono limite di esecuzione inferiore del tipo $f(n) = c^n$ con $c > 1$, sono un numero ristretto di problemi, perlopiù costruiti appositiamente per avere tale limite;
3. problemi che non ammettono algoritmo di risoluzione, ovvero problemi *indecidibili* come l'*Halting Problem* di Turing [1];
4. problemi per i quali non è stato determinato un algorimo efficiente ma di cui non si ha nemmeno un *lower bound*, chiamati per questo *problemi intrattabili*; si tratta di problemi reali, riscontrabili in moltissimi ambiti scientifici e ingegneristici; possiedono inoltre un'interessante *proprietà di chiusura*: risolto uno di questi problemi in tempo polinomiale è possibile risolvere in modo simile anche tutti gli altri problemi di questa classe.

Per semplificare la teoria ci si occupa soltanto di *problemi decisionali*, quindi problemi nella forma

$$\Pi : I \mapsto \{yes, no\} \tag{1}$$

che differiscono da quelli precedentemente incontrati nella forma $\Pi \subseteq I \times S$.
Sia $\Pi_C(x)$ il problema concreto tale che

$$\forall x \in \{0,1\}^*, \quad \Pi_C(x) = \begin{cases} 1 & se \quad (\exists i \in I \qquad \mathsf{e(i)} = \mathsf{x}) \quad \wedge \quad (\Pi_A(i) = x) \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

si definisce la *complessità* (o il tempo) associato ad un algoritmo che risolva $\Pi_C$ la funzione

$$T_{A_{\Pi_C}}(n) = max\{\# \quad passi \quad eseguiti \quad da \quad A_{\Pi_C}(x) \quad \forall x \qquad \mathsf{x} = n\} (3)$$

dove $\mathsf{x} = \mathsf{e(i)}$ è la taglia di $x$ data dalla funzione di encoding $e$.
D'ora in poi si utilizzeranno encoding *concisi*, più vicini possibile all'encoding di taglia minima così da poter avere problemi ugulmente definiti come $\Pi_C : \{0,1\}^* \mapsto \{0,1\}$.

[1] Il problema originale è in realtà il famoso *Entscheidungsproblem*, ovvero *il problema della decisione*, posto da Hilbert nel 1928 e risolto contemporaneamente nel 1936 da A. Turing (con le macchine omonime) e da A. Church (per mezzo del *lambda calcolo*)

## 1.2 Linguaggi Formali

Sia $\Sigma = 0,1$ un alfabeto, si definisce *linguaggio formale* $L$ su $\Sigma$ un insieme di stringhe $L \subseteq \Sigma^*$, $L \subseteq 0,1^*$. Esempi:

- $\epsilon$, stringa vuota
- $L = \{\emptyset\}$, linguaggio vuoto
- $L^0 = L = \{\epsilon\}$, linguaggio contenente soltanto la stringa vuota
- $L_1 \cup L_2$, $L_1 \cap L_2$, unione e intersezione di linguaggi
- $L^C = \{0,1\}^* - L$, linguaggio complementare
- $L_1 \cdot L_2 = \{z \in 0,1^* \quad \exists x \in L_1, \quad \exists y \in L_2 \quad z = \langle x, y\rangle\}$, concatenazione di linguaggi
- $L^* = \bigcup_{i=0}^{\infty} L^i$, tutte le concatenazioni finite di tutti gli elementi di $L$.

Dato un problema concreto $\Pi_C : \{0,1\}^* \mapsto \{0,1\}$, si definisce

$$L_{\Pi_C} = \{x \in \{0,1\}^* \qquad \Pi_C(x) = 1\} \tag{4}$$

è il linguaggio accettato dal problema concreto $\Pi_C$.
Fino a qui si è sviluppata una teoria riguardante soltamente i problemi decisionali, mentre la maggior parte dei problemi sono problemi di ottimizzazione, ovvero nella forma

$$\Pi_{OPT} \subseteq I \times S, \quad c : S \mapsto \mathbb{R}^+ \cup \{0\} \forall i \in I \quad determina \quad s^* \qquad (\mathrm{i}\Pi s\hat{\ }*) \quad \wedge \quad c(s\hat{\ }*) = inf/sup\{c(s), i\Pi s \tag{5}$$

Tuttavia gli algoritmi che risolvono problemi decisionali possono essere introdotti in procedure che risolvono problemi di ottimizzazione. Sia $A$ un algoritmo con input in $\{0,1\}^*$ e output in $\{0,1\}$, $A$ *accetta* $x \in \{0,1\}^*$ se $A(x) = 1$, $A$ *rigetta* $x$ se $A(x) = 0$ ($A$ termina in un numero finito di passi). Si dice *linguaggio accettatto* da $A$ il linguaggio definito come

$$L_A = \{x \in \{0,1\}^* \qquad \mathsf{A(x) = 1}\} \tag{6}$$

Un linguaggio $L$ è deciso in tempo polinomiale da un algoritmo $A$ se

1. $L = L_A$

2. $\exists k_1 \in \mathbb{N}$ T_A( $x$) = O( $x$^–k_1")$

## 1.3 Classi di complessità

Si definisce la classe $\mathcal{P}$ la classe

$$\mathcal{P} = \{L \subseteq \{0,1\} \qquad \text{L deciso in tempo polinomiale}\} \tag{7}$$

$$= \{\Pi_C : \{0,1\}^* \mapsto \{0,1\} \qquad \text{L deciso in tempo polinomiale}\} \tag{8}$$

poiché esiste un isomorfismo tra i problemi concreti $\Pi_C$ e i linguaggi $L$.

Per introdurre la classe $\mathcal{NP}$ è necessario comprendere la differenza tra i concetti di

- *risoluzione*, dato un problema $\Pi \subseteq I \times S$, l'algoritmo che lo risolve riceve un'istanza $i \in I$ e restituisce una soluzione $s \in S$;
- *verificabilità*, dato un problema $\Pi \subseteq I \times S$, l'algoritmo riceve un'istanza de problema $i \in I$ e una sua presunta soluzione $q$ (chiamata *certificato*) e verifica che $q \in S$ o meno.

Sia $V(x, y)$ un algoritmo verificatore. Si dice che $x$ è verificato da $V$ se $\exists y$ V(x,y) = 1$. Si $dicecheL\grave{e}verificatointempopolinomialese \exists V(x, y)$ tale che:

1. $L = L_V$;
2. $\exists c_1, k_1 \in \mathbb{R}^+ \cup \{0\} \quad \forall x \in L \quad \exists y$ $V(x, y) = 1$ e $\mathsf{y} \leq c_1\, \mathsf{x}^{k_1}$
3. $\exists c_2, k_2 \in \mathbb{R}^+ \cup \{0\} \quad \mathsf{T}(x, y) \leq$ $c_2(\mathsf{x} + \mathsf{y})^{k_2}$

Come si può vedere, al punto 2 viene posta una condizione sulla dimensione del certificato. Se questa non fosse presente si potrebbe pensare di fornire tutte le possibili soluzioni come certificato. Ma questo equvarrebbe a risolvere il problema con la forza bruta.

La classe $\mathcal{NP}$ è quindi definita come

$$\mathcal{NP} = \{L \subseteq \{0,1\}^* \quad \text{L verificabile in tempo polinomiale}\} \tag{9}$$

**Nota bene.** $\mathcal{NP}$ non significa *Non Polynomial time*, ma bensì *Nondeterministic Polynomial time*; questo perché la classe $\mathcal{NP}$ può essere alternativamente definita per mezzo delle macchine di Turing non deterministiche.

Con questo teorema si rende esplicito il rapporto tra le classi $\mathcal{P}$ e $\mathcal{NP}$.
**Teorema.** $\mathcal{P} \subseteq \mathcal{NP}$, ovvero $L \subseteq \{0,1\}^* \in \mathcal{P} \Rightarrow L \in \mathcal{NP}$.

**Dimostrazione.** Intuitivamente, essendo $\mathcal{P} = L$ risolubile itp"$ eNP = L verificabile itp"$, se $L \in -P" \Rightarrow \exists A_L \$che$ decide $L$ in tempo polinomiale. Quindi, se $L \in \mathcal{NP}$ deve esistere $V_L$ algoritmo verificatore che verifica $L$ in tempo polinomiale (*aggiungere dimostrazione formale*).

**Riducibilità.** Sia $\Pi_1 : \{0,1\}^* \mapsto \{0,1\}$ e $\Pi_2 : \{0,1\}^* \mapsto \{0,1\}$ e sia $f$ la funzione $f : \Pi_1 \mapsto \Pi_2$, $f : i_1 \mapsto i_2 = f(i_1)$. In questo modo abbiamo che $\Pi_1(i_1) = \Pi_2(f(i_1)) = \Pi_2(i_2)$. Questa funzione è calcolabile in tempo polinomiale se $\exists A_f : \{0,1\}^* \mapsto 0, 1$ $A_f(x) = f(x)$ e $\exists k > 0$ T_–A_f" ( $x$) = O( $x$^k)$.