Linguaggi e Problemi

Indice

La Teoria della Complessità Computazionale è nata con l'obiettivo di classificare i problemi rispetto al loro grado di trattabilità, ossia, rispetto alla quantità di risorse sufficienti a garantire la loro risoluzione per qualunque insieme possibile di valori assunti dai loro dati. Fino ad ora, nella Dispensa 6, abbiamo studiato questa questione in un caso particolare, ossia, il caso in cui il problema da risolvere fosse la decisione circa l'appartenenza di un oggetto ad un insieme. D'altra parte, il concetto di *problema* sembra essere molto più ampio ... Ma è davvero così?
In questa dispensa cercheremo di chiarirci le idee in proposito, tentando di definire formalmente il concetto un po' etereo di "problema" ed individuando diverse tiplogie di problemi. Metteremo, infine, in relazione i problemi con i concetti, e le classi, studiati nella Dispensa 6.

7.1 Cosa è un problema?

Già nella Dispensa 1 abbiamo definito informalmente il concetto di problema così come abbiamo imparato a conoscerlo nel corso della nostra educazione scolastica: un insieme di dati, che soddisfano qualche insieme di relazioni reciproche e che costituiscono una istanza del problema, ed una domanda. Risolvere il problema significa rispondere alla domanda. Nel seguito, rappresenteremo in un ambito formale questo concetto intuitivo.

Osserviamo subito che, quando osserviamo un fenomeno e rileviamo dall'osservazione un certo insieme di dati, tali dati sono soltanto quelli che *esplicitamente* ricaviamo dall'osservazione, ma che esiste un altro insieme di dati *impliciti*, che non misuriamo esplicitamente, che, comunque, caratterizzano il fenomeno. Ad esempio, se abbiamo a disposizione un certo insieme di contenitori (dei quali ignoriamo la capacità) e una bilancia e abbiamo bisogno di un litro di acqua, possiamo riempire uno dei contenitori con 1 kg. di acqua: conoscendo il peso specifico dell'acqua, sappiamo che nel contenitore è contenuto un litro di acqua. In questo caso, l'informazione *esplicita* è data dal peso specifico dell'acqua e dal peso della quantità di acqua che abbiamo nel contenitore (1 kg.); tale informazione esplicita ci permette di ricavare, mediante una legge a noi nota (il volume è il rapporto fra peso e peso specifico) l'informazione implicita. Alla luce di questa informazione possiamo, dunque, affermare che:

un problema è una richiesta di ricavare una qualche informazione implicitamente descritta da un insieme di dati e da una legge che mette in relazione i dati con l'informazione implicita richiesta.

Naturalmente, i dati impliciti che possiamo voler ricavare dai dati espliciti e dalla relazione che abbiamo a disposizione possono essere di molti tipi diversi, ossia, su uno stesso insieme di dati possiamo avere richieste differenti. Nell'esempio dell'acqua riportato sopra, richieste alternative potrebbe essere:

1) trova un contenitore che può contenere (almeno) un litro d'acqua;
2) esiste un contenitore che può contenere (almeno) un litro di acqua?
3) quanti sono i contenitori che possono contenere almeno un litro d'acqua?
4) trova, fra i contenitori che possono contenere almeno un litro d'acqua, il contenitore che può contenerne meno (ossia, quello che più si avvicina a contenere esattamente un litro d'acqua).

In conclusione, formalmente,

Definizione 7.1: Un problema è una quintupla $\langle I, R, S, \eta, \rho \rangle$ in cui I è l'*insieme delle istanze* (ossia, i dati espliciti), R è l'insieme delle risposte, S è la funzione che specifica, per una data istanza, l'insieme delle *soluzioni possibili* per quella istanza[1], $\eta : I \to 2^{S(I)}$ è la funzione che ricava dall'insieme delle soluzioni possibili di una istanza del problema un insieme di *soluzioni effettive* (i dati impliciti, quelli di cui vogliamo ricavare qualche tipo di informazione), e $\rho : I \times \eta(I) \to R$ è la richiesta del problema (ossia, quella che specifica il tipo di informazione che si vuole ottenere, in relazione all'istanza, dall'insieme delle soluzioni effettive di una istanza).
La *risposta* ad una istanza $x \in I$ di un problema è $\rho(x, \eta(x))$.

Chiariamo questi concetti con qualche esempio.

[1] Si osservi che, al fine di non appesantire ulteriormente la notazione, non viene indicato esplicitamente il codominio di S: semplicemente, $S(I)$ denota l'insieme delle soluzioni possibili associate all'istanza I.

Esempio 7.1: Ancora rispetto alla questione di contenitori di acqua, formalizziamo i quattro problemi descritti in precedenza.
In ogni caso, I è l'insieme dell'insieme dei contenitori, ciascuno rappresentato dal peso dell'acqua che può contenere quando è riempito fino all'orlo: dunque, un'istanza $C \in I$ è un insieme $C = \{c_1, \ldots, c_n\}$ tale che $c_i \in \mathbb{R}^+$ per ogni $i = 1, \ldots, n$. Poiché una soluzione possibile è un contenitore, abbiamo anche che le soluzioni possibili $S(C)$ per una data istanza $C \in I$ coincidono con C: $S(C) = C$. Ancora, in tutti i casi, per ogni $C \in I$, ricordando che il peso specifico dell'acqua è 1 (ossia, il volume occupato da una massa d'acqua è sempre pari al suo peso), $\eta(C) = \{c \in C : c \geq 1\}$.
La funzione ρ e l'insieme R nei quattro casi sono definiti di seguito.

1) Trova un contenitore che può contenere (almeno) un litro d'acqua. In questo caso, $\rho(C, \eta(C))$ è un qualsiasi elemento di $\eta(C)$, se $\eta(C) \neq \emptyset$, altrimenti $\rho(C, \eta(C))$ non è definita, e $R = C$.

2) Esiste un contenitore che può contenere (almeno) un litro di acqua? In questo caso, $R = \{\texttt{vero}, \texttt{falso}\}$ e ρ è un predicato: $\rho(C, \eta(C)) \ = \ [\, |\eta(C)| > 0 \,]$.

3) Quanti sono i contenitori che possono contenere almeno un litro d'acqua? In questo caso, $\rho(C, \eta(C)) = |\eta(C)|$ e $R = \mathbb{N}$.

4) Trova, fra i contenitori che possono contenere almeno un litro d'acqua, il contenitore che può contenerne meno (ossia, quello che più si avvicina a contenere esattamente un litro d'acqua). In questo caso, se $\eta(C) \neq \emptyset$, $\rho(C, \eta(C))$ è un contenitore $\hat{c} \in \eta(C)$ tale che $\hat{c} = \min\{c \in \eta(C)\}$ (di nuovo, $\rho(C, \eta(C))$ non è definita se $\eta(C) = \emptyset$) e $R = C$.

Esempio 7.2: Consideriamo il problema seguente: siano dati tre numeri positivi $x, y, A \in \mathbb{R}^+$; esiste un triangolo avente due lati di lunghezza, rispettivamente x e y, e superficie di area A?
In questo caso, $I = \{(x, y, A) : x, y, A \in \mathbb{R}^+$. Inoltre, se consideriamo x come la lunghezza della base del triangolo, una soluzione possibile è una altezza $h \in \mathbb{R}^+$ per il triangolo e, dunque, $S(x, y, A) = \mathbb{R}^+$. Infine, $R = \{\texttt{vero}, \texttt{falso}\}$,

$$\eta(x, y, A) = \{h \in \mathbb{R}^+ : h \leq y \ \wedge \ y^2 - h^2 \leq x^2 \ \wedge \ \frac{hx}{2} = A\},$$

e $\rho(x, y, A, \eta(x, y, A)) = \ [\, |\eta(x, y, A)| > 0 \,]$.

Esempio 7.3: Consideriamo il problema seguente: siano dati un grafo non orientato $G = (V, E)$ ed un intero $K \in \mathbb{N}$; è vero che ogni sottografo di G che contiene almeno k nodi non è completo?
In questo caso, $I = \{(G = (V, E), k) : G$ è un grafo non orientato $\ \wedge \ k \in \mathbb{N}\}$, $S(V, E, k) = \{V' \subseteq V : |V'| \geq k\}$,

$$\eta(V, E, k) = \{V' \in S : \ \forall u, v \in V' \ [\, (u, v) \in E \,]\ \},$$

e $\rho(V, E, k, \eta(V, E, k)) = [\, |\eta(V, E, k)| = 0 \,]$.

7.2 Tipi di problemi

Come abbiamo visto nel precedente paragrafo, possiamo definire diversi tipi di funzione ρ rispetto allo stesso insieme di istanze e anche rispetto alla stessa definizione di soluzioni effettive. In questo paragrafo cercheremo di individuare una classificazione dei problemi rispetto al tipo di funzione ρ che li caratterizza.

I problemi con i quali abbiamo maggiore familiarità sono quelli in cui viene richiesto di individuare (ossia, di calcolare) un elemento dell'insieme delle soluzioni effettive: trovare la somma di due numeri dati, trovare un cammino fra una coppia di nodi in un grafo che abbia lunghezza limitata da qualche dato intero k, ... Problemi di questo tipo vengono detti *problemi di ricerca* ed il modello di calcolo che si presta alla loro soluzione è la macchina di Turing di tipo trasduttore.

I *problemi di ottimizzazione* possono essere considerati un caso particolare dei problemi di ricerca: in un problema di ottimizzazione, agli elementi dell'insieme delle soluzioni effettive è associata una misura e viene richiesto di individuare una soluzione effettiva la cui misura sia ottima (ossia, minima oppure massima) fra tutte le soluzioni effettive.

Anche i problemi di ottimizzazioni si incontrano con grande frequenza: individuare il *massimo* comune divisore di due interi dati, individuare il cammino di lunghezza *minima* che collega una data coppia di nodi in un grafo dato ...

Un'altra categoria di problemi che fa riferimento al modello di macchina di Turing di tipo trasduttore è quella dei *problemi di enumerazione*, in cui si richiede di calcolare il numero di soluzioni effettive di una data istanza. In alternativa, può anche essere richiesto di calcolare esplicitamente *tutte* le soluzioni effettive di una data istanza: calcolare tutti i divisori di un dato numero intero, tutte le permutazioni degli elementi di un dato insieme, tutti i cammini che collegano una data coppia di nodi in un grafo dato ...

Osserviamo, ora, che sino ad ora, non abbiamo utilizzato il modello di macchina di Turing di tipo riconoscitore. Questo modello viene utilizzato per la soluzione dell'ultima categoria di problemi che consideriamo: i *problemi di decisione* o *problemi decisionali*, quelli in cui la funzione ρ è di tipo booleano. I problemi decisionali sono alla base della Teoria della Complessità Computazionale e saranno trattati diffusamente nel resto della dispensa.

7.3 Problemi decisionali e linguaggi

Come abbiamo premesso al termine del precedente paragrafo, i problemi decisionali sono quei problemi che richiedono una risposta di tipo booleano: vero oppure falso. Conseguentemente, l'insieme R è l'insieme $\{\texttt{vero},\texttt{falso}\}$ ed il modello di calcolo soggiacente è quello della macchina di Turing di tipo riconoscitore.

Poiché la funzione ρ che caratterizza la risposta al problema è sempre di tipo booleano, ossia, è un predicato, in un problema decisionale riassumeremo l'insieme R delle risposte, la funzione η, che caratterizza le soluzioni effettive, e la funzione ρ in un unico predicato π. Pertanto, assumeremo, da ora in avanti, che un problema decisionale sia sempre descritto da una tripla $\langle I,S,\pi\rangle$: I è l'insieme delle istanze, S è l'insieme delle soluzioni possibili, e π è un predicato che, con argomenti $x \in I$ e $S(x)$, testimonia se da S si può dedurre l'esistenza di soluzioni effettive per x.

Esempio 7.4: Nell'esempio 7.2, il predicato che descrive il problema, che chiameremo in breve LT, è

$$\pi_{LT}(x,y,A,\mathbb{R}^+) = \exists h \in \mathbb{R}^+ : h \leq y \ \wedge y^2 - h^2 \leq x^2 \ \wedge \ \frac{hx}{2} = A.$$

Nell'esempio 7.3, il predicato che descrive il problema, che chiameremo in breve NC, è

$$\pi_{NC}(V,E,k,S_{NC}(V,E,k)) = \forall\, V' \in S_{NC}(V,E,k)\ [\exists u,v \in V' : (u,v) \notin E\ ] = \forall\, V' \subseteq V : |V'| \geq k\ [\exists u,v \in V' : (u,v) \notin E\ ].$$

Sia, dunque, $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma\rangle$ un problema decisionale. Possiamo partizionare le istanze di Γ nei due insiemi seguenti:

$$Y_\Gamma = \{x \in I_\Gamma : \pi_\Gamma(x) = \texttt{vero}\} \qquad \text{e} \qquad N_\Gamma = \{x \in I_\Gamma : \pi_\Gamma(x) = \texttt{falso}\}.$$

Allora, risolvere il problema Γ significa discriminare gli elementi di Y_Γ da quelli di N_Γ, ossia *riconoscere* gli elementi di Y_Γ. Nel rsto di questo corso, ci riferiremo agli elementi di Y_Γ come alle *istanze sì* del problema Γ e agli elementi di N_Γ come alle sue *istanze no*.

Sembra, quindi, naturale associare al problema decisionale Γ l'insieme Y_Γ, considerare tale insieme un linguaggio ed applicare ad esso tutti i concetti esposti nella Dispensa 6. C'è, però, un problema: un linguaggio è un insieme di parole costituite di caratteri appartenenti ad un alfabeto finito. Percìo, per poter applicare ai problemi i concetti esposti nella Dispensa 6, dobbiamo rappresentare gli elementi dell'insieme I_Γ mediante parole che li codifichino opportunamente.

7.4 Problemi e codifiche

In questo paragrafo cerchiamo di capire il concetto di codifica di un problema decisionale (o, più in generale, di un problema). Iniziamo con un esempio di problema decisionale.

Esempio 7.5: **il problema SODDISFACIBILITÀ**. Una funzione booleana, o predicato, di n variabili è una funzione

$$f : \{\texttt{vero},\texttt{falso}\}^n \to \{\texttt{vero},\texttt{falso}\}$$

che combina n variabili booleane (ossia, variabili che possono assumere solo il valore `vero` o il valore `falso`) mediante gli operatori $\wedge$, $\vee$ e $\neg$: un esempio di funzione booleana è $f(x_1,x_2,x_3) = \neg x_1 \vee (x_2 \wedge x_3)$.
Se la arietà di f è n, per comodità diremo che f è definita sull'insieme di variabili $X = \{x_1,\ldots,x_n\}$.
Una funzione booleana f si dice *in forma congiuntiva normale* se f è la congiunzione di un certo numero $m \in \mathbb{N}$ di clausole, ossia, $f = c_1 \wedge c_2 \ldots \wedge c_m$ dove ogni clausola c_j è la disgiunzione di variabili (semplici o negate) in X.
La funzione definita sopra non è in forma congiuntiva normale, mentre è in forma congiuntiva normale la seguente funzione: $f(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neq x_2 \vee \neg x_3)$.
Se ogni clausola è la disgiunzione di esattamente k variabili, la funzione booleana si dice in forma *k-congiuntiva normale*.
Il problema SODDISFACIBILITÀ (in breve, SAT) chiede se, data una funzione booleana f in forma congiuntiva normale, esiste una assegnazione di verità alle variabili sulle quali f è definita che soddisfa f, ossia, che rende vera f. Formalmente, quindi, SAT è definito dalla tripla seguente:

$I_{SAT} = \{f : \{\texttt{vero},\texttt{falso}\}^n \to \{\texttt{vero},\texttt{falso}\}$ tale che f è in forma congiuntiva normale $\}$;

$S_{SAT}(f) = \{(b_1,\ldots,b_n) \in \{\texttt{vero},\texttt{falso}\}^n\}$;

$\pi_{SAT}(f,S_{SAT}) = \exists (b_1,\ldots,b_n) \in \{\texttt{vero},\texttt{falso}\}^n : f(b_1,\ldots,b_n) = \texttt{vero}$, ossia, sostituendo in f ogni occorrenza della variabile x_i con il valore b_i (ed ogni occorrenza di $\neg x_i$ con $\neg b_i$), per ogni $i = 1,\ldots,n$, la funzione f assume il valore `vero`.

Nel resto di questo esempio ci concentreremo sul problema 3SAT le cui istanze sono funzioni booleane in forma 3-congiuntiva normale. Per completezza, notiamo che, per ogni $k > 0$, il problema kSAT è definito in maniera analoga.
Mostriamo, ora, due codifiche diverse per l'insieme I_{3SAT}.

Codifica χ_1. Sia $\Sigma = \{0,1\}$. Codifichiamo ciascuna delle variabili in X con n caratteri di Σ: in particolare, la variabile x_i è codificata dalla parola di n caratteri il cui unico '1' è quello in posizione i (e, di conseguenza, tutti gli altri caratteri sono '0'). Codifichiamo, poi, ciascuna clausola c_j con la lista delle codifiche delle variabili che essa contiene ciascuna preceduta da uno '0' o da un '1': se la clausola c_j contiene x_i allora, nella codifica di c_j la codifica di x_i sarà preceduta da '0', se, invece, la clausola c_j contiene $\neg x_i$ allora, nella codifica di c_j la codifica di x_i sarà preceduta da '1'. Le clausole saranno codificate una di seguito all'altra, senza interruzione. Infine, per riuscire a capire quando termina una variabile e inizia la successiva (e, quindi, anche quando termina una clausola e inizia la successiva), abbiamo bisogno di conoscere il valore n: a questo scopo, la codifica di f inizia con una sequenza di n '1' seguita da uno '0'. Ad esempio, se $\hat{f} = f(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, la codifica di f sarà:

$$\chi_1(\hat{f}) = 11100100101010011100101010001.$$

Codifica χ_2. Codifichiamo, in questo caso, le istanze di SAT utilizzando, anche ora, l'alfabeto $\{0,1\}$ ma congiuntamente, in questo caso, alla seguente osservazione: ogni funzione booleana è identificata univocamente dai valori che assume su ogni possibile assegnazione di verità alle sue variabili. Ad esempio, la funzione $\hat{f} = f(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ è rappresentata dalla seguente tabella di valori:

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
vero	vero	vero	vero
vero	vero	falso	falso
vero	falso	vero	vero
vero	falso	falso	vero
falso	vero	vero	falso
falso	vero	falso	vero
falso	falso	vero	vero
falso	falso	falso	vero

Allora, possiamo rappresentare ciascuna riga della tabella utilizzando '1' per rappresentare `vero` e '0' per rappresentare `falso`, e scrivendo le righe una di seguito all'altra. Poiché ciascuna riga à costituita da esattamente $n+1$

caratteri, analogamente a quanto fatto per la codifica χ_1, al fine di distinguere una riga dalla successiva, premettiamo alla sequenza delle righe una parola di n caratteri '1' seguita da un carattere '0':

$$\chi_2(\hat{f}) = 11101111110010111001011001010011 0001$$

Osserviamo che entrambe le codifiche utilizzano l'afabeto binario.

Consideriamo, ora, il seguente algoritmo che verifica, data $f \in I_{SAT}$, se f è soddisfacibile:

1) calcola il numero n di variabili sulle quali f è definita f;

2) per ogni assegnazione di verità a all'insieme delle n variabili in f: verifica se $f(a(X)) =$ `vero` e, in tal caso termina nello stato di accettazione q_A;

3) termina nello stato di rigetto q_R[2].

Vediamo ora il precedente algoritmo in esecuzione su una istanza di SAT codificata con entrambe le codifiche.
Se f è codificata mediante la codifica χ_1, allora: il numero n viene individuato leggendo i primi n caratteri della parola $\chi_1(f)$ (i caratteri iniziali, fermandosi al primo '0' che si incontra); poi, per ognuna delle assegnazioni di verità alle n variabili, in un numero di passi proporzionale alla lunghezza di $\chi_1(f)$, si verifica se f è soddisfatta da tale assegnazione. Allora, poiché il numero di assegnazioni di verità ad n variabili booleane è pari a 2^n, in un numero di passi proporzionale a $n + |\chi_1(f)|2^n$ l'algoritmo verifica se f, codificata mediante la codifica χ_1, è soddisfacibile. Valutiamo, ora, $|\chi_1(f)|$: osserviamo, innanzi tutto che, se n è il numero di variabili su cui f è definita, il numero di clausole di 3 variabili è limitato da $(2n)^3$; allora, qualunque sia f,

$$|\chi_1(f)| \leq n + 1 + 3n(2n)^3 \leq 26n^4,$$

ove n è il numero di variabili su cui f è definita. Questo significa, l'algoritmo indicato richiede tempo proporzionale a

$$n + |\chi_1(f)|\ 2^n \geq n + |\chi_1(f)|\ 2^{\frac{\sqrt[4]{|\chi_1(f)|}}{26}},$$

ossia, tempo esponenziale nella lunghezza dell'input.
Se f è codificata mediante la codifica χ_2, allora: il numero n viene individuato leggendo la parola $\chi_2(f)$ fino a quando viene incontrato il primo carattere '0' (e, quindi, in un numero di passi proporzionale a n) e poi, poiché f è codificata rappresentando tutte le possibili assegnazioni di verità per le variabili sulle quali è definita, viene deciso se la funzione è soddisfacibile semplicemente scandendo $\chi_2(f)$. Allora, l'algoritmo verifica se f, codificata mediante la codifica χ_2, è soddisfacibile in un numero di passi proporzionale a $|\chi_2(f)| + n$: dunque, in un numero di passi *polinomiale* nella lunghezza dell'input.
Ma, allora, l'algoritmo che abbiamo proposto è un algoritmo polinomiale oppure no?

Dall'esempio appena discusso, sembra che la caratteristica *essere un algoritmo polinomiale* dipenda dal modo in cui è codificato l'input. Se però osserviamo con più attenzione le due codifiche possiamo concludere quanto segue.

- La codifica χ_1 rappresenta di f solo l'informazione strettamente necessaria , ossia, la sua struttura. Se consideriamo una funzione booleana come l'insieme delle assegnazioni di verità che la soddisfano (che è esattamente quello che abbiamo fatto quando abbiamo descritto la codifica χ_2), la codifica χ_1 è una sorta di rappresentazione per caratteristica di tale insieme.

- La codifica χ_2 rappresenta, invece, f in forma espansa: si tratta di una rappresentazione tabulare dell'insieme corrispondente ad f.

La ragione per cui l'algoritmo indicato opera in tempo polinomiale in $|\chi_2(f)|$ è il fatto che $\chi_2(f)$ rappresenta *esplicitamente* tutte le assegnazioni di verità per f, contrariamente a χ_1 che le rappresenta in forma *implicita* - in effetti, mentre $|\chi_1(f)| \leq 26n^4$, abbiamo che $|\chi_2(f) = n + 1 + 2^n(n+1)$. Come a dire che la codifica χ_2 presenta una sorta di

[2] Si osservi che il passo 3) viene eseguito solo se la macchina non ha terminato (in q_A) in alcuna iterazione del ciclo al punto 2).

ridondanza nel rappresentare l'informazione.
Possiamo affermare che la codifica χ_2 è *irragionevolmente* lunga.

Alla luce di quanto detto sino ad ora, introduciamo il concetto di *codifica ragionevole*.

Definizione 7.2: Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$ un problema decisionale, e sia χ una codifica per I_Γ. La codifica χ è una *codifica ragionevole* per Γ se, per ogni altra codifica χ' per I_Γ, esiste un intero $k \in \mathbb{N}$ tale che, per ogni $x \in I_\Gamma$,

$$|\chi(x)| \in \mathbf{O}(|\chi'(x)|^k).$$

Equivalentemente, possiamo anche dire che χ *non* è una codifica ragionevole per Γ se esistono un'altra codifica χ' per I_Γ e una funzione super-polinomiale F[3] tali che

$$|\chi(x)| \in \Omega(F(|\chi'(x)|)).$$

Segue dalla definizione che, se χ_1 e χ_2 sono due codifiche ragionevoli per un problema Γ, allora esse sono *polinomialmente correlate*, ossia, esistono due interi $h,k \in \mathbb{N}$ tali che, per ogni $x \in I_\Gamma$, $|\chi_1(x)| \in \mathbf{O}(|\chi_2(x)|^h)$ e $|\chi_2(x)| \in \mathbf{O}(|\chi_1(x)|^k)$

7.5 Problemi decisionali e complessità

Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$ un problema decisionale, e sia $\chi : I_\Gamma \to \Sigma^*$ una codifica (ragionevole) per Γ. La codifica χ partiziona Σ^* in tre sottoinsiemi di parole:

- parole che codificano istanze sì di Γ, ossia, parole di Y_Γ;
- parole che codificano istanze no di Γ, ossia, parole di N_Γ;
- parole che *non* codificano istanze di Γ.

Il *linguaggio associato a* Γ mediante la codifica χ è il sottoinsieme $L_\Gamma(\chi)$ di Σ^* contenente le parole che codificano l'insieme Y_Γ, ossia,

$$L_\Gamma(\chi) = \{x \in \Sigma^* : \exists y \in I_\Gamma \, [\, x = \chi(y) \, \wedge \pi_\Gamma(y, S_\Gamma(y)) \,] \,\}.$$

Definiamo, ora, il *linguaggio delle istanze di* Γ, ossia, il linguaggio

$$\chi(I_\Gamma) = \{x \in \Sigma^* : \exists y \in I_\Gamma \, [\, x = \chi(y) \,] \,\}.$$

Allora, ricordando che χ è una codifica di I_Γ e, quindi, $\chi(y) \neq \chi(z)$ se $y,z \in I_\Gamma$ sono due istanze differenti, possiamo definire il linguaggio $L_\Gamma(\chi)$ anche nella meniera seguente:

$$L_\Gamma(\chi) = \{x \in \Sigma^* : x \in \chi(I_\Gamma) \, \wedge \pi_\Gamma(\chi^{-1}(x), S_\Gamma(\chi^{-1}(x))) \,] \,\}.$$

Quest'ultima notazione risulterà utile a breve.

Esempio 7.6: Ricordiamo il problema 3SAT introdotto nell'esempio 7.5 e la codifica χ_1, che abbiamo visto essere una codifica ragionevole. Allora, una parola $x \in \{0,1\}^*$ è in $L_{3SAT}(\chi_1)$ se sono verificati i due fatti seguenti.

1) Innanzi tutto, x deve essere effettivamente la codifica secondo χ_1 di qualche funzione booleana f in forma 3-congiuntiva normale: ad esempio, è facile verificare che 01011100111 non è la codifica di alcuna funzione. Se x non è una codifica valida, possiamo subito concludere che $x \notin L_{3SAT}$.
2) Se x è la codifica secondo χ_1 di una funzione booleana f in forma 3-congiuntiva normale, affinché $x \in L_{3SAT}(\chi_1)$ occorre che f sia soddisfacibile.

[3] Ossia, F cresce più velocemente di qualunque polinomio. Formalmente, per ogni $k \in \mathbb{N}$, $\lim_{n\to\infty} \frac{F(n)}{n^k} = +\infty$.

A questo punto, possiamo definire la complessità computazionale di un problema decisionale.

Definizione 7.3: Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$ un problema decisionale, sia $\chi : I_\Gamma \to \Sigma^*$ una codifica ragionevole per Γ e sia $\mathscr{C}$ una classe di complessità (spaziale o temporale, deterministica o non deterministica). Diciamo che $\Gamma \in \mathscr{C}$ se $L_\Gamma(\chi) \in \mathscr{C}$.

Dall'Esempio 7.6 e dalla definizione sopra si evince che nel computo della complessità computazionale di un problema decisionale entra in gioco anche il costo della verifica che la parola in esame sia effettivamente la codifica di un'istanza del problema. L'esempio che segue serve a chiarire questo concetto.

Esempio 7.7: Consideriamo il problema decisionale Γ_1 seguente: sia dato grafo non orientato $G = (V,E)$ che contiene un ciclo che passa una ed una sola volta per ciascuno dei suoi nodi, e siano dati due suoi nodi $u, v \in V$; decidere se esiste in G un percorso che collega u a v.
Formalizziamo il problema precedante mediante la tripla $\langle I_{\Gamma_1}, S_{\Gamma_1}, \pi_{\Gamma_1} \rangle$:

- $I_{\Gamma_1} = \{\langle G = (V,E), u, v\rangle : G$ è un grafo non orientato $\wedge \exists$ un ciclo c in G che passa una e una sola volta attraverso ciascun nodo di $G \wedge u, v \in V\}$;
- $S_{\Gamma_1}(G,u,v) = \{p : p$ è un percorso in $G\}$;
- $\pi_{\Gamma_1}(G,u,v,S_{\Gamma_1}(G,u,v)) = \exists p \in S_{\Gamma_1}(G,u,v)$ che connette u a V.

Naturalmente, se sappiamo che un grafo contiene un ciclo che passa (una e una sola volta) attraverso tutti i nodi di G, allora $S_{\Gamma_1}(G,u,v)$ contiene anche la coppia di percorsi contenuti nel ciclo che congiungono u a v e v a u. Questo significa che ogni istanza del problema è una istanza sì. Quindi, indipendentemente dalla codifica utilizzata, decidere se una qualunque istanza del problema soddisfa il predicato del problema richiede costo costante.
D'altra parte, data una qualunque codifica ragionevole (diciamo, binaria) χ per Γ_1, per decidere se una parola $x \in \{0,1\}^n$ è contenuta in $L_{\Gamma_1}(\chi)$, dobbiamo verificare sia se x è la codifica di un grafo *che contiene un ciclo che attraversa tutti i nodi una e una sola volta* e di una coppia di suoi nodi, sia se detto grafo contiene un percorso che connette i due nodi. Come vedremo, la prima di queste due verifiche (ossia la verifica che x sia effettivamente la codifica di un'istanza di Γ_1) è un noto problema **NP**-completo. Pertanto, anche se la verifica del predicato π_{Γ_1} richiede tempo costante, non possiamo certo affermare che Γ_1 sia un problema appartenente a **P**. In effetti, si dimostra che Γ_1 è un problema **NP**-completo.

7.6 Istanze sì, istanze no, non istanze: problemi decisionali complemento

Come abbiamo visto al termine del precedente paragrafo, poiché la complessità dei problemi decisionali viene ricondotta a quella dei linguaggi, la complessità di un problema dipende non soltanto dal suo predicato (e, quindi, da quanto è difficile trovare una soluzione per una sua istanza), ma anche dalla difficoltà di decidere l'insieme delle sue istanze. Per questa ragione, alcuni dei risultati validi nella teoria della complessità dei linguaggi non si traducono direttamente in risultati validi per la teoria della complessità dei problemi decisionali. In particolare, questa mancata corrispondenza si verifica quando si considerano le classi complemento ed è connessa alla questione ambigua delle non istanze.

Osserviamo, in effetti, come si presenti una evidente asimmetria fra la partizione generata in Σ^* (ove Σ è un qualunque alfabeto finito) da un linguaggio e la partizione generata in Σ^* da un problema decisionale: mentre un linguaggio individua una *bi*-partizione di Σ^* in parole del linguaggio e parole del linguaggio complemento, un problema decisionale ed una sua codifica in Σ individuano una *tri*-partizione di Σ^* in parole che codificano istanze sì, parole che codificano istanze no, e parole che codificano non-istanze. Vediamo, in quel che segue, le conseguenze della asimmetria appena evidenziata sulla complessità del *problema complemento* di un dato problema.

Dato un problema decisionale $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$, il suo problema complemento, intuitivamente, è

$$\Gamma^c = \langle I_\Gamma, S_\Gamma, \neg\pi_\Gamma \rangle,$$

ossia, è costituito dalle istanze di Γ per le quali il predicato non è soddisfatto, cioè, dagli $y \in I_\Gamma$ tali che $\pi_\Gamma(y, S_\Gamma(y))$ è falso.

Dunque, fissata una codifica ragionevole $\chi : I_\Gamma \to \Sigma^*$ per il problema decisionale Γ, il linguaggio associato al problema Γ^c, complemento di Γ, è

$$L_{\Gamma^c}(\chi) = \{x \in \Sigma^* : \chi(x) \in I_\gamma \wedge \neg\pi_\Gamma(\chi{-}1(x), S_\Gamma(\chi^{-1}(x)))\}.$$

Come si può osservare, $L_{\Gamma^c}(\chi)$ non coincide con il linguaggio complemento $L^c_\Gamma(\chi)$ di $L_\Gamma(\chi)$; infatti,

$$L^c_\Gamma(\chi) = \{x \in \Sigma^* : \chi(x) \notin I_\Gamma \vee \neg\pi_\Gamma(\chi{-}1(x), S_\Gamma(\chi^{-1}(x)))\}.$$

Questa mancata coincidenza fra $L_{\Gamma^c}(\chi)$ e $L^c_\Gamma(\chi)$ è causa di una importante anomalia della complessità computazionale dei problemi decisionali: essa, infatti, impedisce di classificare il problema decisionale Γ^c come necessariamente appartenente alla classe di complessità co$\mathscr{C}$ quando Γ appartiene a $\mathscr{C}$.

Esempio 7.8: Consideriamo il complemento del problema decisionale Γ_1 descritto nell'Esempio 7.7. Il problema Γ^c_1, allora, è il seguente: sia dato grafo non orientato $G = (V,E)$ che contiene un ciclo che passa una ed una sola volta per ciascuno dei suoi nodi, e siano dati due suoi nodi $u, v \in V$; decidere se non esiste in G alcun percorso che collega u a v. Ribadiamo quanto affermato nell'Esempio 7.7: se sappiamo che un grafo contiene un ciclo che passa (una e una sola volta) attraverso tutti i nodi di G, allora ogni istanza del problema Γ_1 è una istanza sì del problema Γ_1. Equivalentemente, ogni istanza del problema Γ_1 (e, dunque, ogni istanza del problema Γ^c_1) è una istanza no del problema Γ^c_1. Quindi, indipendentemente dalla codifica utilizzata, decidere se una qualunque istanza del problema soddisfa il predicato del problema

$$\Gamma^c_1$$

richiede costo costante.
D'altra parte, data una qualunque codifica ragionevole (diciamo, binaria) χ per Γ_1, esattamente come per il linguaggio $L_{\Gamma_1}(\chi)$, anche per decidere se una parola $x \in \{0,1\}^n$ è contenuta in $L_{\Gamma^c_1}(\chi)$, dobbiamo verificare se x è la codifica di un grafo che contiene un ciclo che attraversa tutti i nodi una e una sola volta e di una coppia di suoi nodi. Dunque, anche il linguaggio $L_{\Gamma^c_1}(\chi)$ è **NP**-completo. Osserviamo, invece, che, in virtù del Teorema 6.21, il linguaggio $L^c_{\Gamma_1}$ è co**NP**-completo.

L'anomalia evidenziata dall'esempio sopra, in cui un problema decisionale e il suo complemento non appartengono, rispettivamente, ad una certa classe di complessità e alla sua classe complemento non si verifica se il linguaggio delle istanze del problema è "facile" da decidere, come precisato dal seguente teorema.

Teorema 7.1: *Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma\rangle$ un problema decisionale e sia $\chi : I_\Gamma \to \Sigma^*$ una sua codifica ragionevole. Se $\chi(I_\Gamma) \in$ **P**, allora*

- *se $L_\Gamma(\chi) \in$ **P** allora $L_{\Gamma^c}(\chi) \in$ co**P**;*
- *se $L_\Gamma(\chi) \in$ **NP** allora $L_{\Gamma^c}(\chi) \in$ co**NP**.*

.

Dimostrazione: Poiché $\chi(I_\Gamma) \in \mathbf{P}$, allora esistono una macchina di Turing deterministica T ed un intero h tali che, per ogni $x \in \Sigma^*$, T decide se $x \in \chi(I_\Gamma)$ e $dtime(T,x) \in \mathbf{O}(|x|^h)$.

Se $L_\Gamma(\chi) \in \mathbf{P}$, allora esistono una macchina di Turing deterministica T_Γ ed un intero k tali che, per ogni $x \in \Sigma^*$, T_Γ decide se $x \in L_\Gamma(\chi)$ e $dtime(T_\Gamma,x) \in \mathbf{O}(|x|^k)$. Combinando T e T_Γ, deriviamo una nuova macchina deterministica T' che, con input $x \in \Sigma^*$, opera in due fasi, come di seguito descritto.

Fase 1. Simula la computazione $T(x)$: se $T(x)$ termina nello stato di rigetto, allora anche T' termina nello stato di rigetto in quanto $x \notin \chi(I_\Gamma)$, altrimenti ha inizio la Fase 2.

Fase 2. Simula la computazione $T_\Gamma(x)$ invertendo, però, gli stati finali: se $T_\Gamma(x)$ accetta allora T' termina nello stato di rigetto, se $T_\Gamma(x)$ rigetta allora T' termina nello stato di accettazione.

Quindi, $T'(x)$ accetta se e soltanto se $x \in \chi(I_\Gamma)$ e $x \notin L_\Gamma(\chi)$, ossia, se e soltanto se $x \in L_{\Gamma^c}(\chi)$. Inoltre, è semplice verificare che $dtime(T',x) \in \mathbf{O}(|x|^{\max\{h,k\}})$. In conclusione, $L_{\Gamma^c}(\chi) \in \mathbf{P}$.

Se $L_\Gamma(\chi) \in \mathbf{NP}$, allora esistono una macchina di Turing non deterministica NT_Γ ed un intero k tali che, per ogni $x \in L_\Gamma(\chi)$, NT_Γ accetta x e $ntime(NT_\Gamma, x) \in \mathbf{O}(|x|^k)$. Combinando T e NT_Γ, deriviamo una nuova macchina non deterministica NT' che, con input $x \in \Sigma^*$, opera in due fasi, come di seguito descritto.

Fase 1. Simula la computazione $T(x)$: se $T(x)$ termina nello stato di rigetto, allora NT' termina nello stato di accettazione, altrimenti ha inizio la Fase 2.

Fase 2. Simula la computazione $NT_\Gamma(x)$.

Quindi, $NT'(x)$ accetta se e soltanto se $x \notin \chi(I_\Gamma)$ oppure $x \in L_\Gamma(\chi)$, ossia, se e soltanto se x appartiene al linguaggio complemento di $L_{\Gamma^c}(\chi)$. Inoltre, è semplice verificare che $ntime(NT', x) \in \mathbf{O}(|x|^{\max\{h,k\}})$. In conclusione, il linguaggio complemento di $L_{\Gamma^c}(\chi)$ è in **NP**, e dunque $L_{\Gamma^c}(\chi) \in$ co**NP**. □

Risultati analoghi possono essere dimostrati per le classi **EXPTIME** e **NEXPTIME**. Osserviamo infine, che, poiché co**PSPACE** = **PSPACE** = **NPSPACE** = co**NPSPACE**, banalmente, $L_\Gamma(\chi) \in$ **PSPACE** se e solo se $L_{\Gamma^c}(\chi) \in$ **PSPACE**.

7.7 Problemi e complessità: le nostre assunzioni di lavoro

Nel problema Γ_1 degli Esempi 7.7 e 7.8 appare chiaro che tutta la difficoltà del decidere circa l'appartenenza di una parola $x \in \Sigma^*$ tanto al linguaggio L_{Γ_1} quanto al linguaggio $L_{\Gamma_1^c}$ va individuata nella difficoltà di decidere se x è effettivamente una istanza del problema (ossia, se $x \in \chi(I_{\Gamma_1})$) mentre la complessità del decidere il predicato π_{Γ_1} è ininfluente: ogni volta che $x \in \chi(I_{\Gamma_1})$, decidere $\pi_{\Gamma_1}(x, S_{\Gamma_1}(x))$ richiede tempo costante.
Con riferimento alla nostra idea intuitiva di problema, questo non è ragionevole: la difficoltà nel risolvere un problema non dovrebbe essere nel riconoscere che i dati che ci vengono forniti siano effettivamente dati del nostro problema, ma nel trovare una soluzione (o nel verificare che una soluzione esiste) ad una data istanza del problema. Per questa ragione, da ora in avanti assumeremo che

dato un problema decisionale $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$ *ed una codifica ragionevole* χ *per esso,*
decidere se $x \in \chi(I_\Gamma)$ *è un problema in* ***P****.*

Questa assunzione ci permetterà di non incorrere nelle discrepanze evidenziate precedentemente fra la teoria della complessità dei linguaggi e la teoria della complessità dei problemi decisionali.

Alla luce di tale assunzione, sorge spontanea la domanda: ma allora il problema Γ_1 di cui agli Esempi 7.7 e 7.8 non è un problema decisionale? Certamente, Γ_1 è un problema decisionale. La questione è che esso non è descritto dalla tripla presentata nell'esempio. Informalmente, data la descrizione di un problema decisionale, considereremo parte dell'istanza solo quei dati che sono *assolutamente necessari* a comprendere la struttura del problema, e sposteremo nel predicato *tutte le proprietà che devono essere soddisfatte dai dati che costituiscono l'istanza*. Così, entrando nel dettaglio negli Esempi 7.7 e 7.8, assumeremo:

- $I_{\Gamma_1} = \{\langle G = (V,E), u, u\rangle : \ G$ è un grafo non orientato $\wedge\, u, v \in V\}$;
- $S_{\Gamma_1}(G,u,v) = \{p : \ p$ è un percorso in $G\}$;
- $\pi_{\Gamma_1}(G,u,v,S_{\Gamma_1}(G,u,v)) = \exists$ un ciclo c in G che passa una e una sola volta attraverso ciascun nodo di $G \wedge \exists p \in S_{\Gamma_1}(G,u,v)$ che connette u a V.

La nostra ultima assunzione di lavoro, sulla quale ci baseremo da ora in avanti, concerne la codifica delle istanze di un problema.
Quando parliamo di codifica di un'istanza, ci riferiamo a due questioni distinte: dobbiamo decidere *cosa* rappresentare dell'istanza (ossia, quali sono i dettagli dei dati del problema che rappresenteremo), e, una volta operata questa scelta, dobbiamo decidere *come rappresentarli*. La scelta di cosa rappresentare dei dati di un problema è, per così dire, una scelta ad alto livello che non tiene conto dei dettagli implementativi: ad esempio, come abbiamo visto, di una funzione booleana possiamo scegliere di rappresentare la struttura (ossia, la sequenza di variabili, parentesi e operatori che la compongono) oppure l'insieme dei valori che essa assume nelle varie assegnazioni di valori alle sue variabili.

Una volta operata questa scelta, dobbiamo decidere in quale modo codificare, effettivamente, le strutture che abbiamo prescelto: quale alfabeto utilizzare, quanti caratteri dell'alfabeto riservare ad ogni componente della struttura, come separare i vari componenti della struttura, e così via. Dunque, la scelta di come rappresentare i dati è una scelta, per così dire, a basso livello che si interessa, cioè, di dettagli puramente implementativi.
Dovrebbe apparire chiaro, da quanto discusso sino ad ora e dall'Esempio 7.5, che due codifiche che rappresentano le stesse strutture dei dati e che differiscono soltanto per quelle che abbiamo chiamato scelte a basso livello sono sempre polinomialmente correlate. Questo a patto, chiaramente, di non usare un numero inutile e spropositato di caratteri per una delle due: se per rappresentare una funzione booleana in forma 3-congiuntiva normale utilizziamo lo stesso schema della codifica χ_1 dell'Esempio 7.5 in cui ciascuna variabile è codificata con n caratteri (per la variabile x_i, essi sono $n-1$ caratteri '0' ed un carattere '1' in posizione i) ma decidiamo di separare una clausola dalla successiva mediante 2^n caratteri '0', otteniamo una codifica che non è polinomialmente correlata con χ_1 soltanto perché dilata inutilmente la codifica delle clausole (intervallarle mediante i 2^n zeri non è, infatti, di alcuna utilità). Pertanto, a patto di evitare inutili dilatazioni delle codifiche a basso livello, potremo, da ora in avanti, concentrarci sulla scelta di cosa rappresentare dei dati di un problema, ossia, da ora in poi,

parlando di codifiche ragionevoli, ci riferiremo sempre alla scelta (ad alto livello) delle strutture dei dati che intenderemo rappresentare

svincolandoci, così, dai dettagli implementativi[4].
Così, come ulteriore esempio, codifica ragionevole di un grafo sarà quella che, del grafo, rappresenta l'insieme dei nodi e l'insieme degli archi. Sarà, invece, irragionevole la codifica che rappresenta tutti i possibili percorsi che connettono ogni coppia di nodi.

In conclusione, da ora in avanti, parleremo sempre di complessità computazionale di un problema, senza far più riferimento alla sua codifica (che assumeremo ragionevole) o al linguaggio che gli corrisponde.
Infine, dato un problema Γ e una sua istanza $x \in I_\Gamma$, coerentemente con quanto discusso in questa dispensa circa la complessità dei problemi decisionali, diremo che $x \in \Gamma$ per intendere che x è una istanza sì di Γ.

[4]Osserviamo che, in realtà esiste per le funzioni booleane in forma 3-congiuntiva normale una codifica binaria molto più "economica" della codifica χ_1, che chiameremo χ_0: nella codifica χ_0 ciascuna variabile booleana viene codificata utilizzando $\lceil \log n \rceil$ caratteri (la codifica della variabile x_i è la codifica binaria di i cui si aggiungono eventuali '0' iniziali per utilizzare tutti i $\lceil \log n \rceil$ caratteri). Tuttavia, è semplice verificare che χ_0 e χ_1 sono polinomialmente correlate.