Lezione 2: tanti tipi di problemi, anche insolubili

Miriam Di Ianni

5 gennaio 2011

1 Tanti tipi di problemi

Abbiamo iniziato la scorsa lezione definendo, più o meno formalmente, quello che tradizionalmente viene considerato un problema: un insieme di dati a partire dai quali si deve *calcolare* il *valore di una soluzione*. Possiamo tradurre questo concetto in un linguaggio squisitamente matematico, e parlare di problemi che consistono nel *calcolo di una funzione*.

Esempio. Nell'esempio `SommaEDifferenzaSegmenti`, presentato all'inizio della precedente lezione, si richiedeva il calcolo delle lunghezze l_1 ed l_2 di due segmenti dati la loro somma s e la loro differenza d. Possiamo esprimere lo stesso problema affermando che vogliamo calcolare la funzione che associa a due valori numerici, s e d, rappresentanti la somma e la differenza delle lunghezza di due segmanti, le lunghezze dei due segmenti. Come abbiamo visto, tale funzione (a due valori) può essere espressa nella forma

$$f(s,d) = \left(\frac{s+d}{2}, s - \frac{s+d}{2}\right)$$

oppure

$$f(s,d) = \left(\frac{s-d}{2}, d + \frac{s+d}{2}\right).$$

Osserviamo che anche il problema dell'ordinamento di un vettore di interi è un problema di calcolo di una funzione, in cui viene richiesto di calcolare la permutazione di indici del vettore che corrisponde ad un vettore ordinato.

Al termine della lezione precedente, poi, abbiamo incontrato un diverso tipo di problema, in cui veniva richiesto di rispondere "sì" oppure "no". Nel problema che abbiamo incontrato era richiesto di *decidere* se una data parola fornita come input era palindroma. Problemi di questo tipo vengono detti *decisionali* e, sostanzialmente, consistono nella richiesta di verificare se l'insieme dei dati (d'ora in avanti, l'*istanza*) verifica una certa proprietà.

Potrebbe sembrare, ad uno sguardo superficiale, che la definizione di problema decisionale caratterizzi un insieme molto ristretto di problemi - solo quei problemi in cui ci riduciamo a chiederci "è vero che ..."?
In realtà, con l'ausilio di qualche piccolo artificio, possiamo associare un problema decisionale a qualunque problema di calcolo di una funzione: è sufficiente scegliere un elemento e nel codominio della funzione (ossia, nell'insieme dei valori che possono essere assunti dalla funzione) e chiederci se, nella data istanza, il valore della funzione è proprio e (oppure, minore o uguale di e, oppure maggiore o uguale di e).

Esempio. Consideriamo ancora il problema `SommaEDifferenzaSegmenti`; una sua versione decisionale è definita come segue:

dati quattro numeri interi s, d, l_1 ed l_2, è vero che $s = l_1 + l_2$ e che $d = l_1 - l_2$?

In molti casi, possiamo trovare la soluzione di un problema di calcolo di una funzione risolvendo per un opportuno numero di istanze la sua versione decisionale.

Esempio. Consideriamo la seguente funzione: dato un insieme $A = \{a_1, a_2, \ldots, a_n\}$ di n numeri interi, $f(A)$ è il numero di elementi di A maggiori di 0. Una versione decisionale del problema del calcolo di f è la seguente: dati un insieme $A = \{a_1, a_2, \ldots, a_n\}$ di n numeri interi ed un numero intero positivo k, è vero che il numero di elementi positivi di A è maggiore o uguale di k?
Mostriamo ora come calcolare $f(A)$ risolvendo il problema decisionale su al più n istanze. Sia $A = \{a_1, a_2, \ldots, a_n\}$ l'insieme dato; l'algoritmo che calcola $f(A)$ decidendo al più n istanze del problema decisionale è il seguente:

1. impostiamo $k = 1$;
2. se il numero di elementi positivi di A è maggiore o uguale a k allora
   - incrementa k di 1 e ripeti il passo 2.
3. altrimenti $f(A) = k - 1$.

Infine, una importante categoria di problemi è quella dei *problemi di ottimizzazione* in cui viene richiesto di calcolare il valore minimo o il valore massimo di una funzione.

Esempio. Sia $X = \{x_1, x_2, \ldots, x_n\}$ un insieme e sia $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ una famiglia di sottoinsiemi di X (ossia, ciascun S_j è un sottoinsieme di X). Vogliamo scegliere il *minimo* numero di elementi di $\mathcal{S}$ in modo tale che ogni elemento di X compaia in almeno uno degli elementi di $\mathcal{S}$ scelti.

Analogamente ai problemi di calcolo di funzioni, anche ai problemi di ottimizzazione possiamo associare versioni decisionali.

Poiché, dunque, per risolvere qualunque tipo di problema riusciamo sempre a ricondurci a risolvere problemi decisionali, d'ora in avanti ci limiteremo a considerare soltanto questo tipo di problemi.

Attività. Associare un opportuno problema decisionale al problema di ottimizzazione definito nell'esempio precedente e mostrare come il problema di ottimizzazione possa essere risolto risolvendo un opportuno numero di istanze della sua versione decisionale.

2 Quanti sono gli algoritmi?

Certamente, come possiamo immaginare facilmente, esistono un numero infinito di algoritmi. Ma, allora, che senso ha questa domanda? In altri termini, come si fa a contare gli elementi di un insieme infinito? O, ancora, esistono infiniti più grandi di altri infiniti? La risposta alla terza domanda è: sì, esistono infiniti più grandi di altri infiniti. La scoperta di questo fatto, tutt'altro che intuitivo, è dovuta a Georg Cantor, un matematico tedesco vissuto a cavallo del XIX e del XX secolo. In questo paragrafo, inizieremo a rispondere alla seconda domanda, alla luce delle scoperte di Cantor. Invece, nel Paragrafo 3, proprio utilizzando le tecniche individuate da Cantor, mostreremo perché la terza domanda ha risposta affermativa.

Dunque, come si fa a contare gli elementi di un insieme infinito? Ovviamente, non ha senso contare esplicitamente gli elementi di un insieme infinito (sappiamo già che sono infiniti!). Quello che interessa realmente è, piuttosto, mettere in relazione due insiemi infiniti, ossia far corrispondere agli elementi di un insieme gli elementi di un altro insieme in modo tale che:

- ogni elemento del primo insieme è in corrispondenza con uno ed un solo elemento del secondo insieme (in Figura 1.(i) è mostrato un esempio in cui questo non avviene) e
- ogni elemento del secondo insieme è in corrispondenza con uno ed un solo elemento del primo insieme (in Figura 1.(ii) è mostrato un esempio in cui questo non avviene).

Una corrispondenza fra due insiemi che rispetta i due vincoli sopra definiti viene detta *biunivoca*. Un esempio di corrispondenza biunivoca è illustrato in Figura 1.(iii). Intutivamente, è chiaro che due insiemi *finiti* (ossia,

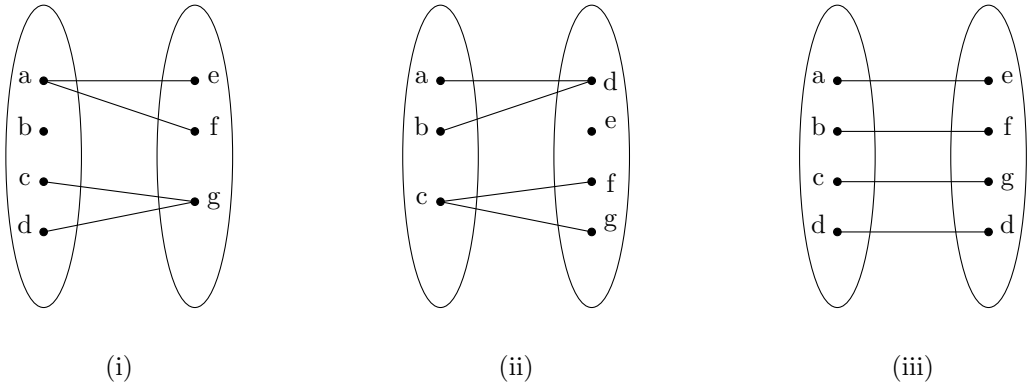

Figura 1: Le corrispondenze in (i) e (ii) sono esempi di corrispondenze non biunivoche. La corrispondenza in (iii) è biunivoca.

contenenti un numero finito di elementi) fra i quali è possibile definire una corrispondenza biunivoca hanno lo stesso *numero di elementi*. Se mettiamo in corrispondenza biunivoca due insiemi *infiniti*, non possiamo più affermare che hanno lo stesso numero di elementi (perché l'infinito non è un numero!). Però, il concetto è sostanzialmente lo stesso: diciamo, in questo caso, che i due insiemi sono *equipotenti* o, anche, che hanno la stessa *cardinalità*.

Cosa ha a che vedere tutto ciò con problemi e algoritmi? Iniziamo a rispondere a questa domanda mostrando che "gli algoritmi sono tanti quanti i numeri naturali". No, diciamolo bene: l'insieme degli algoritmi ha la stessa cardinalità dell'insieme $\mathbf{N}$ dei numeri naturali. Per provare questa affermazione dobbiamo costruire una corrispondenza biunivoca fra l'insieme degli algoritmi e l'insieme dei numeri naturali; in altri termini, questo significa che dobbiamo trovare un modo per *numerare* gli algoritmi, ossia, per poter dire quale è il primo, quale il secondo, e così via. Ma come fare a numerare gli algoritmi se il concetto di algoritmo è così generico come abbiamo visto la precedente lezione?

Innanzi tutto, ricordiamo la tesi di Church-Turing: tutto ciò che è calcolabile lo è mediante una macchina di Turing. Questa tesi ci permette di considerare, anziché algoritmi generici, macchine di Turing. Infatti, la tesi di Church-Turing implica che, dato un qualsiasi algoritmo (descritto in un qualsiasi linguaggio) che calcola una data funzione, esiste una macchina di Turing che calcola la stessa funzione. Allora, possiamo limitarci a numerare le macchine di Turing.

Vediamo, dunque, come numerare le macchine di Turing. Ricordiamo che una macchina di Turing T è univocamente descritta dall'insieme Q delle quintuple che la caratterizzano. Se l'alfabeto $A = \{a_1, a_2, \ldots, a_n\}$ e l'insieme $S = \{s_0, s_1, \ldots, s_h\}$ degli stati sono considerati ordinati (ad esempio, assumendo $a_i < a_j$ e $s_i < s_j$ se $i < j$), allora anche l'insieme Q può essere considerato ordinato secondo l'ordinamento lessicografico: la quintupla $\langle s_{i_1}, a_{j_1}, a_{j_2}, m, s_{i_2}\rangle$ precede la quintupla $\langle s_{i_3}, a_{j_3}, a_{j_4}, m', s_{i_4}\rangle$ se la parola $s_{i_1}a_{j_1}a_{j_2}ms_{i_2}$ precederebbe in un dizionario la parola $s_{i_3}a_{j_3}a_{j_4}m's_{i_4}$.
Sia dunque $\{q_1, q_2, \ldots, q_k\}$ l'ordinamento risultante dell'insieme Q (con $q_1 < q_2 < \ldots < q_k$).

Associamo, ora, a ciascuna quintupla di T un numero intero, calcolato nel modo seguente: siano $b(a_j)$, $b(s_i)$ e $b(m)$, rispettivamente, la codifica in binario di $a_i \in A$, di $s_j \in S$ e di $m \in \{sinistra, destra, ferma\}$; allora, la quintupla $q = \langle s_{i_1}, a_{j_1}, a_{j_2}, m, s_{i_2}\rangle$ viene associata al numero intero $c(q) = b(s_{i_1})2b(a_{j_1})2b(a_{j_2})2b(m)2b(s_{i_2})$ (ottenuto concatenando le varie componenti della quintupla con il numero 2).

Infine, la macchina T è univocamente rappresentata dalla numero $c(q_1)3c(q_2)3\ldots3c(q_k)$, ottenuto concatenando i numeri associati a ciascuna quintupla (ordinatamente) con il numero 2. È possibile dimostrare che la numerazione ottenuta è univoca, ossia, a macchine di Turing diverse sono associati numeri diversi. Dunque, abbiamo messo in corrispondenza biunivoca l'insieme delle macchine di Turing con un sottoinsieme

dei numeri naturali e questo dimostra che la cardinalità dell'insieme delle macchine di Turing non è maggiore di quella dell'insieme dei numeri naturali. D'altra parte, essendo anche l'insieme delle macchine di Turing infinito, la sua cardinalità coincide con quella dell'insieme dei numeri naturali.

Attività. Calcolare il numero associato alla macchina di Turing che decide se una parola è palindroma.

3 Quanti sono i problemi?

Consideriamo, ora, l'insieme dei problemi: quale è la cardinalità di tale insieme? Mostriamo, in questo paragrafo, che esiste un problema che non può essere risolto da alcun algoritmo e che, quindi, la cardinalità dell'insieme dei problemi è maggiore di quella dell'insieme $\mathbf{N}$.

Dimostriamo questo fatto utilizzando la tecnica di *diagonalizzazione*, una particolare tecnica di dimostrazione per assurdo introdotta da Cantor.
Ricordiamo che l'insieme delle macchine di Turing è numerabile. Inoltre, utilizzando un procedimento analogo a quello presentato per dimostrare la numerabilità dell'insieme delle macchine di Turing, è possibile dimostrare che anche l'insieme delle parole di lunghezza finita su alfabeti finiti è numerabile: infatti, possiamo codificare qualunque alfabeto in binario e, a partire da tale codifica, numerare le parole di lunghezza finita analogamente a quanto abbiamo fatto con le macchine di Turing.

Torniamo, ora, alla definizione di macchina di Turing: un *passo* di una macchina di Turing è l'esecuzione di una singola quintupla. Osserviamo che, date una macchina di Turing T ed una parola x, non è detto che la macchina T con input x raggiunga uno stato finale in un numero finito di passi (in qualche modo, il concetto di non terminazione di una macchina di Turing equivale al concetto di loop di un programma).

Introduciamo, infine, il seguente insieme

$$H = \{i : \text{ la macchina di Turing di indice } i \text{ non termina con input la parola di indice } i\}$$

che ci permette di definire il Problema della Fermata: dato un intero positivo x, è vero che $x \in H$?

Supponiamo che esista una macchina di Turing T che risolva il Problema della Fermata: allora, per ogni input x, T con input x risponde sì se e soltanto se $x \in H$, ossia, T con input x risponde sì se e soltanto se la macchina di Turing di indice x con input la parola di indice x non termina. Inoltre, la macchina T avrà un certo indice k nella numerazione dell'insieme delle macchine di Turing. Mostriamo, ora, che per ogni $i \in \mathbf{N}$ $k \neq i$: infatti, la macchina T con input la parola di indice i risponde sì se e soltanto se la macchina di indice i con input la parola di indice i non termina. Dunque, per ogni $i \in \mathbf{N}$, il comportamento della macchina T differisce dal comportamento della macchina di indice i per almeno un input. Questo prova che non esiste un indice k che può essere assegnato alla macchina T. Ma, poiché l'insieme delle macchine di Turing è numerabile, allora la macchina T non esiste, ossia,

il Problema della Fermata non è risolubile.

Riassumendo, nelle prime due lezioni di questo corso abbiamo mostrato i fatti seguenti:

- ogni algoritmo risolve un problema (anche se un problema può essere risolto da più di un algoritmo);
- l'insieme degli algoritmi ha la stessa cardinalità dell'insieme dei numeri naturali;
- esiste un problema che non è risolto da alcun algorimo.

In conclusione, esistono più problemi che algoritmi, ossia, la cardinalità dell'insieme degli algoritmi è maggiore di quella dell'insieme dei problemi.

Attività. Definire una macchina di Turing che non termina per qualche input.

4 Autoreferenzialità e paradossi

Torniamo a considerare il Problema della Fermata e cerchiamo di capire cosa è che lo rende tanto difficile.
L'input di tale problema è costituito da una parola ed una macchina di Turing; quindi, per risolvere una istanza del problema avremmo bisogno di una *macchina di Turing* T che prenda in input una parola ed una *macchina di Turing*. Naturalmente, poiché T è una macchina di Turing, T *potrebbe prendere sé stessa come input*, ossia T dovrebbe operare una decisione su sé stessa.

Questa situazione, in cui una entità è chiamata ad operare su sé stessa, viene chiamata *autoreferenzialità*. L'autoreferenzialità è alla base di molti paradossi logico-matematici.

Il paradosso del mentitore. Io sto mentendo. Potete credermi?
Supponiamo che mi crediate: allora starei dicendo la verità, ossia, sarebbe vero che sto mentendo. Ma se stessi mentendo non potrei certo dire la verità.
Allora, meglio non credermi. Cioè, sto affermando il falso. Dunque, non è vero che sto mentendo. Ma se non sto mentendo, allora sto dicendo la verità, mentre io affermo che sto mentendo ...
Ancora autoreferenzialità: sto affermando qualcosa su me stessa.

Il paradosso di Russel. È, questo, un paradosso che mina le fondamenta stesse della Teoria degli Insiemi (e, quindi, della Logica Matematica).
Sappiamo tutti cosa ve un insieme: una collezione di oggetti (eventualmente, astratti) che hanno tutti una qualche proprietà - la *proprietà caratteristica* o *legge di appartenenza* dell'insieme.
Esistono infinite varietà di insiemi. Fra tutti gli insiemi esistono insiemi che contengono sé stessi come elemento: ad esempio l'insieme dei concetti è esso stesso un concetto e, quindi, è contenuto in sé stesso. Autoreferenzialità, di nuovo.
Possiamo, allora, considerare l'insieme $\mathcal{N}$ degli insiemi che non contengono sé stessi come elemento. Sorge, ora, spontanea la domanda: $\mathcal{N} \in \mathcal{N}$?
Supponiamo intizialmente che $\mathcal{N} \in \mathcal{N}$: in questo caso $\mathcal{N}$ non dovrebbe contenere sé stesso come oggetto, e quindi $\mathcal{N} \notin \mathcal{N}$.
Allora, deve essere $\mathcal{N} \notin \mathcal{N}$: ma allora $\mathcal{N}$ deve contenere sé stesso come oggetto e, quindi, $\mathcal{N} \in \mathcal{N}$ ...
Quale è la soluzione? non c'è soluzione: è un paradosso!