Fabrizio Luccio. Appunti di algoritmica

3 Rappresentazione dell'informazione e algoritmi

Sequenze di rappresentazione

La rappresentazione di un oggetto in senso lato (per esempio un'immagine o un insieme di suoni) prende forma in matematica come sequenza di segni da esaminare uno dopo l'altro. Questi segni sono i *caratteri* o *simboli* di un *alfabeto* finito: possono essere segni grafici, lettere, note musicali ecc., e devono essere impiegati in modo che a oggetti distinti corrispondano sequenze distinte.

Per esempio sullo schermo di un computer un'immagine è una matrice di punti colorati organizzati per righe parallele, così piccoli che l'occhio umano non può singolarmente distinguerli ma ovviamente li distingue, li genera e li elabora un insieme di circuiti. Dunque anche l'immagine nel suo complesso è rappresentabile come sequenza di caratteri, ciascuno determinato dalla posizione nella sequenza e dal colore. Il colore a sua volta è rappresentato da una sequenza di caratteri: nel caso dello schermo la sequenza è di ventiquattro bit (caratteri binari) pari a tre byte (gruppi di otto bit): per esempio 11111111 11111111 00000000 è un bel giallo vivo.

Ma quanti colori distinti si rappresentano con ventiquattro bit? Come vedremo $2^{24} > 4.000.000$; un numero enormemente maggiore di quanto sarebbe necessario anche per immagini bellissime, e che trae motivo da altre considerazioni (il byte è un'unità standard nei circuiti commerciali, e ciascuno dei tre byte rappresenta la componente di una tricromia con cui il colore viene formato). È però in genere necessario mantenere basso, o se possibile minimizzare, il numero di simboli richiesti per la rappresentazione di oggetti, e ciò è legato a un principio matematico.

Poniamo che l'alfabeto $\mathcal{A}$ contenga n caratteri, ovvero $|\mathcal{A}| = n$, e indichiamo con N il numero di oggetti da rappresentare. La lunghezza $s(n,N)$ della più lunga sequenza che rappresenta un oggetto dell'insieme (ovvero la "parola" più lunga per dare un nome distinto a tutti gli oggetti) è funzione dei parametri n, N, e il valore di $s(n,N)$ dipende dal metodo di rappresentazione scelto.

Se $n = 1$, cioè $\mathcal{A}$ contiene un solo carattere, i vari oggetti potranno essere rappresentati solo da ripetizioni di quel carattere e almeno una di esse dovrà contenerne N. Se il carattere è 0, i sette nani saranno indicati per esempio con 0,00, . . . , 0000000, secondo una rappresentazione *unaria* che è estremamente sfavorevole come ora vedremo.

Per $n = 2$, convenzionalmente $\mathcal{A} = \{0, 1\}$, la rappresentazione è *binaria*. Per ogni intero $k \geq 1$ (vedi dispensa 2.2) le possibili sequenze binarie di lunghezza k sono 2^k e il numero totale di sequenze binarie lunghe da 0 a k è $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$, o $2^{k+1} - 2$ escludendo la sequenza di lunghezza zero cioè vuota. Per rappresentare N oggetti dovremo dunque avere $2^{k+1} - 2 \geq N$; e applicando il logaritmo in base 2 a entrambi i membri della disuguaglianza avremo $k \geq \log_2(N + 2) - 1$. Nella migliore rappresentazione possibile $s(n,N)$ sarà pari al minimo intero k che soddisfa tale relazione, ovvero $s(n,N) = \lceil \log_2(N + 2) - 1 \rceil$, da cui facilmente si ottiene:

$\lceil \log_2 N \rceil - 1 \leq s(2, N) \leq \lceil \log_2 N \rceil$.

Dunque $\lceil \log_2 N \rceil$ caratteri binari sono sufficienti (e "quasi" necessari, a meno di 1) per costruire N sequenze differenti e quindi per dare un nome a N oggetti. La rappresentazione unaria che richiede almeno una sequenza lunga N mostra dunque uno svantaggio esponenziale rispetto alla binaria che richiede lunghezza logaritmica in N (si ricordi che le funzioni esponenziale e logaritmica sono una l'inverso dell'altra).

Il ragionamento si estende immediatamente a rappresentazioni *n-arie*, convenzionalmente $\mathcal{A} = \{0, 1, ..., n-1\}$ con $n > 2$, per cui si può semplicemente dimostrare che $\lceil \log_n N \rceil - 1 \leq s(n, N) \leq \lceil \log_n N \rceil$, e quindi $\lceil \log_n N \rceil$ caratteri n-ari sono sufficienti per costruire N sequenze differenti. L'esempio più immediato è costituito dai numeri decimali ($n = 10$) per cui con sequenze di k cifre si costruiscono 10^k numeri (per esempio per $k = 3$ si costruiscono i $10^3 = 1000$ numeri da 000 a 999).

In conclusione utilizzando alfabeti di $n \geq 2$ caratteri si possono rappresentare N oggetti con sequenze di $O(\log N)$ caratteri. Tali rappresentazioni sono dette *efficienti* in alternativa alla non efficiente rappresentazione unaria. Si noti che l'influenza della cardinalità n dell'alfabeto per $n \geq 2$ non ha grande impatto sulla lunghezza della rappresentazione perché, come osservato, i logaritmi sono tutti proporzionali tra loro (e infatti la base non è indicata nell'ordine di grandezza). Passando per esempio da base 10 a base 2 il coefficiente di proporzionalità è $\log_2 10 \approx 3.32$, ovvero la rappresentazione binaria dei numeri è circa tre volte più lunga della rappresentazione decimale.

Notiamo infine che quanto fin qui detto non ha relazione con la natura degli oggetti rappresentati ma riguarda solo la possibilità di identificarli con un nome. Ciò non esclude che possa esservi una relazione semantica tra nome e oggetto: l'esempio più evidente è quello dei numeri, ottimizzati al massimo possibile nella numerazione posizionale (o araba) che impieghiamo normalmente.

Studiamo ora un metodo per elencare le sequenze: non è un problema ovvio come potrebbe sembrare. Le sequenze generate con i caratteri di un alfabeto finito sono infinite ma *numerabili*, cioè possono essere poste in corrispondenza biunivoca con i numeri naturali 0, 1, 2 . . . . A tale scopo dobbiamo stabilire un metodo per ordinare le sequenze in modo che ciascuna di esse possa essere raggiunta in un numero finito di passi a partire da l'inizio dell'elenco. Perciò si impiega un *ordinamento canonico*. Consideriamo per esempio le sequenze costruite sull'alfabeto $\mathcal{A} = \{a, b, \ldots, z\}$. L'ordinamento canonico è basato sulle regole seguenti:

1. si stabilisce un ordine tra i caratteri dell'alfabeto (nel nostro esempio si userà l'ordinamento alfabetico standard);

2. si ordinano le sequenze per lunghezza crescente e, a pari lunghezza, si segue l'ordine stabilito all'interno dell'alfabeto come in un dizionario.

Nel nostro esempio avremo l'ordinamento:

$a, b, \ldots, z,$
$aa, ab, \ldots, az,\ ba, bb, \ldots, bz,\ \ldots\ldots, za, zb, \ldots, zz,$
$aaa, aab, \ldots, aaz, \ldots\ldots \qquad\qquad \ldots\ldots, zzz,$
$\ldots\ldots\ldots$

L'elenco inizia con le sequenze costituite da un solo carattere, numerate da 1 per a fino a 26 per z; seguono le aa, ab, ac, ecc. da 27 in poi. Si aggiunge inoltre all'inizio la sequenza vuota associata allo 0. In questo modo si ottiene una corrispondenza biunivoca tra i numeri naturali e le sequenze: dato qualsiasi numero a questo corrisponde esattamente una sequenza e viceversa.

Si noti che l'ordinamento canonico è semplice ma non ovvio: per esempio non potremmo ordinare le sequenze in ordine alfabetico come in un dizionario indipendentemente dalla loro lunghezza, perché l'elenco conterrebbe tutte le infinite sequenze che iniziano con a prima di quelle che iniziano con b: dunque le sequenze che iniziano con b non potrebbero essere a distanza finita dall'inizio dell'elenco e non avrebbero numeri finiti ad esse associati. In matematica si conclude che le sequenze sono *infinite* e *numerabili*: esse costituiscono quindi "il più piccolo" infinito possibile.

Problemi e algoritmi

Da quanto detto fin qui possiamo affermare che tutta la conoscenza trasmessa dall'uomo è rappresentabile con le parole di un codice costruito su un insieme finito di simboli. E queste parole sono potenzialmente infinite ma numerabili.

La teoria degli insiemi infiniti nacque alla fine del 1800 per opera di Georg Cantor che dimostrò che esistono infiniti di diverse *cardinalità*, ovvero che non possono essere posti in corrispondenza biunivoca tra loro. L'esempio più famoso è l'infinito dei numeri reali che sono *di più* degli interi, quindi non sono numerabili. La teoria si sviluppò poi nel corso di pochi decenni con conseguenze fondamentali per l'informatica, o meglio per quella parte di essa che va sotto il nome di *algoritmica*.

Scopo di questa disciplina è infatti quello di affrontare la risoluzione di un problema mediante l'esecuzione di una serie di azioni che tipicamente possono essere descritte a una macchina in senso lato; il che richiede di formulare un *algoritmo* e tradurlo in un *programma* per la macchina che operi su un insieme di *dati*. Algoritmo, programma e dati sono descritti con sequenze di simboli e sono quindi potenzialmente infiniti ma numerabili. Questo implica in particolare che i dati su cui i programmi sono chiamati a operare hanno natura *discreta*: per esempio un calcolatore non può lavorare su numeri reali ciascuno dei quali richiederebbe una sequenza infinita di descrizione, ma lavora su approssimazioni finite (tipicamente su troncamenti a un massimo numero di cifre), quindi equivalentemente su interi.

Possiamo sostanzialmente dire che un algoritmo e il suo programma calcolano una *funzione* intera $y = f(x)$ definita sugli interi, perché hanno la cardinalità degli interi sia il *dominio* (valori della x) che il *codominio* (valori della y). Per esempio un programma per ordinare n nomi in ordine alfabetico, ciascuno rappresentato da un gruppo di caratteri di lunghezza prefissata k (per esempio k byte in binario),

ha come dato di ingresso una sequenza di n pacchetti di k caratteri e come risultato una sequenza riordinata della stessa lunghezza. Le due sequenze rappresentano rispettivamente i valori di x e y nella funzione di ordinamento $y = f(x)$. Riferendoci a quanto detto sugli ordini di grandezza le due sequenze hanno lunghezza kn e quindi sono di ordine $\Theta(n)$ poiché k è una costante. Un algoritmo di ordinamento è chiamato a calcolare il valore di y per ogni possibile valore di x e la sua complessità sarà funzione di n. Vediamo ora perché la teoria degli infiniti ha influenzato profondamente la nascita dell'informatica.

Le sequenze fin qui considerate sono in numero infinito, ma sono di lunghezza finita benché a priori illimitata: cioè per qualsiasi intero n si possono costruire sequenze più lunghe di n. Ogni funzione è invece associata a una sequenza infinita. Infatti $y = f(x)$ stabilisce una corrispondenza tra ogni valore di x e un valore di y. Volendola rappresentare in forma tabellare si dovrebbe scrivere un intero y per ciascun intero x: dunque una sequenza infinita di y. Una tabella di funzioni dovrebbe associare una riga a ogni funzione $f_0, f_1, f_2, \ldots$ e una colonna a ogni intero $x = 0, 1, 2, 3, \ldots$, riportando in ogni casella i, j il valore di $f_i(j)$. La tabella ha infinite colonne per x, e infinite righe perché le corrispondenze tra x e y che si possono immaginare sono ovviamente infinite.

Utilizzando il metodo di ragionamento di Cantor si può dimostrare che le funzioni *non sono numerabili*, ovvero esistono più funzioni che interi. Infatti posto che una qualsiasi numerazione $f_0, f_1, f_2, \ldots$ esista, si consideri la particolare funzione definita come:

$$g(i) = f_i(i) + 1, \quad \text{per ogni } i = 0, 1, 2, \ldots$$

cioè la funzione che si ottiene sommando 1 a tutti i valori contenuti nella diagonale principale della tabella ove sono riportati i valori di $f_i(i)$. La funzione $g(n)$ è chiaramente definita in modo legittimo ma non può coincidere con alcuna delle funzioni della tabella perché differisce da ciascuna nel valore riportato nella diagonale, e questo è vero per qualunque numerazione fosse assegnata alle funzioni. Dobbiamo concludere che le funzioni *sono di più* degli interi, ovvero l'insieme delle funzioni è infinito e non numerabile.[1] Ma poiché gli algoritmi sono sequenze finite e quindi numerabili desumiamo un principio fondamentale:

Poiché esistono più funzioni che algoritmi, devono esistere funzioni per cui non esiste algoritmo di calcolo.

Tali funzioni, e i problemi che esse rappresentano, sono dette *non calcolabili*, e sorprendentemente sono in numero infinitamente maggiore di quelle calcolabili perché il loro infinito è più vasto di quello degli interi, quindi degli algoritmi.

Nei primi decenni del 1900 questi problemi furono intensamente dibattuti dai logici-matematici: se infatti l'esistenza di funzioni non calcolabili era innegabile, non se ne conosceva nemmeno una. Il problema fu chiuso nel 1936 quando Alonzo

[1] Lo stesso *argomento diagonale* fu impiegato da Cantor per dimostrare che i reali sono più degli interi.

Church e Alan Turing posero due definizioni di algoritmo equivalenti tra loro e fino a oggi accettate assieme al primo problema non calcolabile in tali paradigmi, che nel caso di Turing era:

Non può esistere un algoritmo A che decida in tempo finito se un altro algoritmo B termina in tempo finito la sua elaborazione su dati D, per B e D arbitrari.

Più in generale la lezione di Church e Turing va interpretata come l'impossibilità "inter pares" di decidere del comportamento di un altro se non simulando tale comportamento (ma se B non termina il suo calcolo su D in tempo finito, A non potrà conoscere se B termina o meno attraverso la simulazione di B).

D'ora in aventi ci occuperemo di problemi calcolabili e dei loro algoritmi di risoluzione lasciando all'informatica teorica il compito di indagare sulla non calcolabilià dei problemi. Il che, in un'ottica di integrazione tra l'informatica e la biologia, lascia aperto il formidabile problema di stabilire se il comportamento di un sistema biologico sia dimostrativamente imprevedibile.