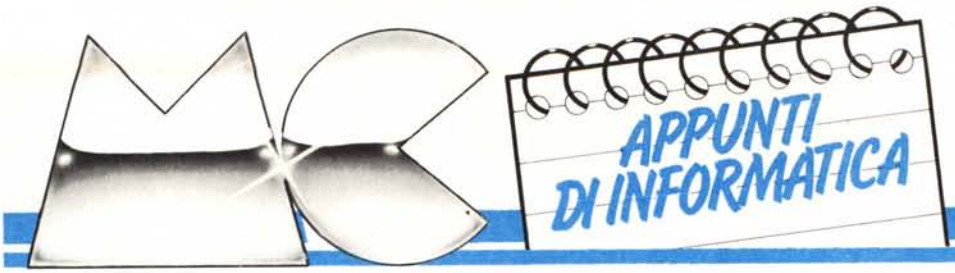

di Andrea de Prisco

Teoria della computabilità

prima puntata

Appunti di informatica, a partire da questo mese, metterà un pò il naso nella teoria dell'informatica. Teoria nel vero senso della parola: assiomi, teoremi, dimostrazioni, asserti, tesi e congetture, naturalmente senza cadere troppo nel difficile o nel noioso (almeno si spera) come di solito accade in queste cose. Si vuole solo mostrare qualche scorcio di «dietro le quinte» che generalmente chi non studia informatica allo stato puro (non necessariamente all'università) non dovrebbe aver già visto. Per citare subito qualcosa, i limiti di calcolo di un computer, il formalismo della Macchina di Turing, gli automi a stati finiti, le grammatiche generative, la semantica denotazionale ed altro. Nel nostro viaggio cercheremo di mostrare quanto di più interessante per tutti troveremo scartabellando tra tutte queste argomentazioni che fanno parte delle basi teoriche dell'informatica. Se a questo aggiungiamo che molti giovani lettori ci scrivono per saperne di più sul corso di laurea in Informatica, possiamo subito svelarvi che quanto state per leggere, è stato estratto dal programma di Metodi per il Trattamento dell'Informazione, materia del terzo anno di studi, nonché una delle più importanti.

L'agente di Calcolo

Nello studio teorico della computabilità, ovvero di ciò che che è calcolabile o non calcolabile con una macchina, la prima cosa da tenere ben chiara è il concetto di Algoritmo. Intuitivamente tutti sanno che cosa esso sia: di solito un procedimento abbastanza «eseguibile» che dovrebbe descrivere ciò che vogliamo fare. Ad esempio, un programma scritto in un qualsiasi linguaggio di programmazione è un algoritmo in quanto (per l'appunto) descrive un procedimento: ciò che il computer dovrà fare. Questo intuitivamente.

Per sviluppare però una teoria riguardo agli algoritmi è d'obbligo una definizione più precisa, specialmente per quel che concerne ciò che circonda l'algoritmo in sé. Tanto per dirne una, l'algoritmo ha ragione d'esistere solo se c'è qualcosa in grado di eseguirlo: un affare pieno zeppo di 0 e 1 è un algorimo se questi 0 e 1 costituiscono un programma per una macchina che accetti tale linguaggio di programmazione. Per agente di calcolo, si intende proprio questo: chi è in grado di eseguire le istruzioni di cui è composto l'algoritmo. La *definizione* di algoritmo continua indicando che qualsiasi programma deve essere di lunghezza finita, che l'agente di calcolo ha a disposizione una memoria e soprattutto che non bisogna fare ipotesi limitative circa la lunghezza dei dati in ingresso, sul numero di passi per effettuare la computazione, sulla quantità di memoria di cui l'agente di calcolo dispone. Tra le limitazioni imposte il fatto che il calcolo deve avvenire in maniera deterministica e che le singole istruzioni devono avere una complessità non infinita.

Approfondiremo meglio quanto appena detto in seguito.

Numeri naturali e funzioni

Più volte, tra le pagine di appunti di informatica, è stato ribadito il concetto che i calcolatori si chiamano così perché calcolano e se possono fare questo nel modo più bello, più facile e più veloce possibile è meglio per tutti. Nella *teoria* informatica, come (qualitativamente) un calcolo avvenga è roba di poco conto, anzi nullo. Ciò che importa è: preso l'algoritmo, l'agente di calcolo e i dati di ingresso, quali saranno i risultati. La prima cosa che faremo, sarà di ridurre la tripla «dati-algoritmo-risultati» a una qualsiasi funzione dai numeri naturali ai numeri naturali. A tal proposito. apriamo una brevissima parentesi.

I numeri naturali, come tutti sanno, sono l'insieme dei numeri che generalmente siamo abituati a considerare: 0, 1, 2, 3, 4, ...

I puntini sospensivi stanno ad indicare il fatto che tali numeri sono infiniti ovvero sappiamo che il più piccolo è lo zero ma non è possibile indicare un estremo superiore di tale insieme. Qualsiasi numero naturale prendiamo, ce n'è sicuramente un altro più grande di questo, che possiamo ottenere banalmente sommando al primo il numero 1.

Una funzione da un naturale a un naturale, è un procedimento che prende come input un numero naturale e ne restituisce un altro (non necessariamente diverso). Ad esempio, la funzione «successore» prende in input un naturale e ne restituisce un altro ottenuto sommando 1 al primo. Dunque la funzione successore é definita (può essere calcolata) su tutti i naturali in virtù del fatto che qualsiasi naturale prendiamo possiamo ottenere il suo successore.

Come controesempio potremmo citare il caso della funzione predecessore che è definita su tutti i naturali tranne che per lo 0 in quanto come è noto lo zero non segue nessun altro naturale (meno uno non vale perché non appartiene ai naturali ma ai numeri relativi: in tale ipotesi si potrebbe dire che la funzione predecessore è definita su tutti numeri relativi). Essere definita su tutti i naturali è sinonimo di funzione totale sui naturali, definizione che useremo spesso in seguito.

Da dati a dato

Per assimilare un oggetto composto da un numero qualsivoglia grande di dati in ingresso, da un algoritmo e da un insieme di risultati ad una funzione che preso un naturale restituisce un altro naturale, il grosso del lavoro sarà quello di trasformare più numeri in un solo numero e viceversa. Ovvero partendo dai dati del nostro programma, otterremo in modo univoco un solo «numerone» che passeremo alla nostra funzione naturale. Questa, come qualsiasi funzione che si rispetti, restituirà un valore, che noi ritrasformeremo in un insieme di risultati. Si badi bene che tutto ciò lo stiamo facendo per ridurre lo studio degli algoritmi generici a quello delle funzioni dai naturali ai naturali, che sono manipolabili molto più semplicemente.

Per passare in modo univoco da un insieme di numeri a un solo numero il quale a sua volta identifichi univocamente l'insieme di numeri dal quale siamo partiti, esistono diversi metodi. Per semplicità noi vedremo quello delle potenze di numeri primi o della scomposizione in fattori primi (a seconda del verso in cui l'adoperiamo) che tra l'altro è anche il più veloce da spiegare.

Siano $A_1, A_2, ..., B_N$ i primi N numeri primi. Il numero naturale che identifica univocamente i dati in ingresso è:

$$B_1^{A_1} \cdot B_2^{A_2} \cdot \ldots \cdot B_N^{A_N}$$

ovvero B_1 elevato a A_1 per B_2 elevato a A_2 fino a B_N elevato alla A_N. Facciamo un esempio: troviamo il numero naturale che identifica univocamente la quadrupla di dati (4,3,1,2). I primi 4 numeri primi sono 2,3,5,7 quindi il risultato sarà:

$$2^4 \cdot 3^3 \cdot 5^1 \cdot 7^2 = 16 \cdot 27 \cdot 5 \cdot 49 = 105840$$

per l'appunto un numerone. Per tornare da un naturale all'insieme di numeri che questo identifica è sufficiente effettuare la scomposizione in fattori primi: ad esempio il numero 126000 identifica la quadrupla (4,2,3,1) in quanto effettuando la scomposizione otteniamo:

$$126000 = 2^4 \cdot 3^2 \cdot 5^3 \cdot 7^1$$

Tipi di Dato

Nella nostra trasformazione da insiemi di dati a un unico numero, abbiamo dato per scontato il fatto che i dati iniziali fossero degli interi. Sicuramente qualcuno potrebbe obiettare che i computer manipolano anche stringhe, numeri in virgola mobile, numeri negativi, record, quindi il restringerci ai soli numeri interi potrebbe essere troppo limitato. Non è così: basta infatti pensare al fatto che qualsiasi struttura dati prendiamo, essa è sempre composta da una quantità più o meno grande di celle di memoria. Conoscendo il fatto che in ogni byte (ad esempio) ci può stare solo un numero tra 0 e 255, ecco che le nostre stringhe, numeri in virgola mobile, record e qualsiasi altra struttura computereccia, può sempre essere vista come insieme di numeri, tutti compresi tra lo zero e la quantità massima memorizzabile in una singola cella di memoria.

Per fare un esempio che si potrebbe considerare limite, il word processor che in questo momento è usato per la redazione dell'articolo che state leggendo, è una funzione da insiemi di caratteri in insiemi di caratteri: i dati in ingresso sono le parole immesse via tastiera e i risultati sono i caratteri che compongo l'articolo una volta formattati per linea e per pagina. Tramite la trasformazione vista sopra, dato che ogni carattere ha un proprio codice ASCII, possiamo ricavarci il numerone che identifica univocamente il testo in ingresso (e vi assicuro che questa volta sarà composto da qualche miliardo di cifre, ndr). Il word processor a questo punto diventa (è un altro programma, ben inteso) una funzione da «un» naturale a «un» naturale e per stampare il nostro articolo sarà sufficiente scomporre in fattori primi il numerone in uscita e prelevare tutti gli esponenti che saranno il codice ASCII di tutti i caratteri di cui il testo formattato si compone.

Figura 1

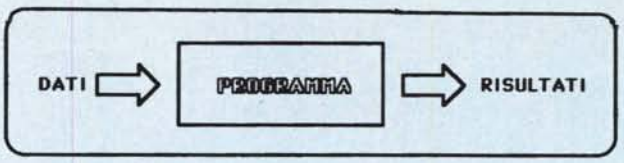

Figura 2

C - Codifica i dati in un numero naturale
F - È la funzione da N→N corrispondente al programma
D - Decodifica l'output di **F** nei risultati.

Goedelizzazione

Assodato che qualsiasi programma che accetti i più disparati input e che restituisca output qualsiasi può essere ricondotto a una funzione da un naturale a un naturale (anche se spesso non sappiamo come questa sia fatta), il problema che intendiamo porci è se esiste o non esiste un formalismo (agente di calcolo più specifiche per scrivere gli algoritmi) capace di calcolare tutte le funzioni possibili e immaginabili dai naturali ai naturali.

Tanto per anticiparvi il risultato che ci accingiamo a dimostrare, un tale formalismo non può esistere: ciò si traduce nel fatto che esistono effettivamente dei problemi che un calcolatore non potrà mai risolvere. Non certo per mancanza di tempo o di memoria: abbiamo detto che per un algoritmo sono ammesse computazioni infinite e che la memoria disponibile per l'agente di calcolo è anch'essa infinita. E' proprio che alcuni problemi si dimostrano essere non calcolabili. Di altri è dimostrata la calcolabilità, di altri ancora fino ad ora nessuno sa se siano calcolabili o meno.

Il primo passo verso la dimostrazione di tale asserto consiste nell'osservare che preso un formalismo S, un programma è rappresentato da un insieme finito e ordinato (una stringa) di simboli. Ad ogni programma scritto in tale formalismo, come detto, possiamo associare una funzione dai naturali ai naturali che, una volta trasformati i dati in ingresso nel naturale da passare alla funzione e preso il risultato ricostruiti da questo i risultati del programma, corrisponde al programma di partenza (vedi figura 1 e 2).

Una simile trasformazione da programmi possibili a funzioni corrispondenti, sicuramente non sarà iniettiva. Nel senso che dato un formalismo è sicuramente possibile scrivere lo stesso algoritmo (nel senso comune del termine) in maniere diverse. Per l'esattezza lo stesso algoritmo lo posso scrivere in infiniti modi diversi. Basta ad esempio cambiare nomi alle variabili oppure aggiungere delle linee che non fanno nulla o qualcosa che non cambia i risultati. Ovvero più programmi corrispondono di fatto alla stessa funzione calcolata.

Ciò si traduce nella prima delle disequazioni utili per la dimostrazione: il numero delle funzioni calcolate dal formalismo S è minore o uguale al numero di programmi che posso scrivere con questo. E si indica così:

$$\#F_s \leq \#P_s$$

figura 3:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13....
0	1	1	1	1	0	0	0	0	0	0	0	0	0	0...
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0...
2	1	0	0	1	1	0	0	0	1	1	1	0	1	0...
3	0	0	1	1	0	0	1	1	0	0	1	1	0	0....
4	0	0	0	0	1	1	1	1	0	0	0	0	1	1...
5	1	0	1	1	1	0	0	1	0	0	1	0	.....	
6	0	1	0	1	0	1	0	1	0	1	0	1	0	1...
7	0	1	1	0	1	0	0	0	0	0	0	1	1	1...
8	0	0	0	0	0	1	0	0	0	0	0	0	0	0....
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1.....
10	1	0	0	0	0	0	0	0	0	0	0	0	0	1....
11	1	0	1	1	1	1	1	1	0	1	0	1	.....	
12	0	1	1	0	0	1	1	1	1	1	1	....	.....	
:														
:														

Tornando al fatto che un programma scritto per il formalismo S è comunque una stringa di simboli, possiamo enumerare tutti i programmi possibili nel seguente modo: li metto prima in ordine di lunghezza (prima i programmi composti da un solo simbolo [se esistono], poi quelli da due ecc.) e nell'ambito di ogni possibile lunghezza opero un ordinamento lessicografico (alfabetico). Dato che enumerare vuol dire associare (nella costruzione di questa tabella) un naturale ad ognuno di questi programmi, la cardinalità dell'insieme dei programmi scritti nel formalismo S è pari a quella dei naturali: é possibile dato un numero conoscere qual è il programma corrispondente nell'ordinamento che facevamo sopra e, viceversa, dato un programma possiamo conoscere la sua posizione in tabella. Come sia possibile il passaggio da numeri a programmi e viceversa, è assai semplice. Se abbiamo detto che sappiamo enumerare, per conoscere ad esempio il programma 147 basterà costruire la nostra tabella fino a tale posizione e prelevare l'ultimo programma. Analogamente per conoscere quale numero nella tabella dei programmi occupa uno di questi, inizieremo a costruire la nostra tabella e continueremo fino a quando non troviamo. Tale procedimento non può continuare all'infinito dato che del programma che stiamo cercando sappiamo di quanti caratteri esso è composto. E dato che ogni gruppo di programmi di pari lunghezza può essere composto solo da un numero finito di programmi (al massimo pari alle combinazioni dei simboli possibili in classe «numero di simboli di cui è composto il programma»), la nostra ricerca avrà sicuramente fine prima di arrivare alla classe di programmi un solo simbolo più lungo del nostro.

La nostra disuguaglianza si arricchisce allora con:

$$\#F_S \leq \#P_S = \#N$$

dove # N rappresenta la cardinalità dell'insieme dei numeri naturali. Per maggiori ragguagli sulle cardinalità finite e transfinite vi rimandiamo al riquadro della pagina a fronte.

A noi, l'unica cosa che interessa per la dimostrazione è che la cardinalità dell'insieme di tutte le funzioni calcolabili dal formalismo S è minore o uguale della cardinalità dei naturali. Dunque:

$$\#F_S \leq \#N$$

L'operazione di mettere in ordine i programmi scritti in un formalismo S è detta Goedelizzazione in onore del matematico Goedel.

Parti di N

Il passo successivo è dimostrare che l'insieme dei naturali non può essere messo in corrispondenza biunivoca con i suoi sottoinsiemi, dove per sottoinsieme si intende qualsiasi «raccolta» di elementi presi dall'insieme di partenza. Ovvero che la cardinalità dei naturali è strettamente minore di quella delle sue parti, ovvero ancora che non esiste un funzione biunivoca che preso un qualsiasi sottinsieme dei naturali restituisca un naturale e viceversa. Si badi bene che quanto stiamo dicendo non è la stessa cosa di prima. Ora è possibile anche che i sottosistemi siano infiniti: ad esempio l'insieme dei numeri pari è un sottinsieme infinito dei naturali così come quello dei dispari o dei numeri primi. Quando al paragrafo precedente operavamo la trasformazione tra insiemi di numeri e naturali, i primi potevano contenere un numero qualsiasi di elementi ma non infiniti. In questo caso, ciò che non fa funzionare più il procedimento visto prima è proprio il fatto che abbiamo a che fare con sottoinsiemi anche infiniti dei naturali e dunque non potremmo ottenere il «numerone» elevando a potenza i primi «infiniti» numeri primi.

Dimostreremo che non è possibile mettere in corrispondenza biunivoca i naturali e i suoi sottoinsiemi. Per fare questo dovremo idealmente manipolare oggetti infiniti: del resto è con insiemi di tale «portata» che ci accingiamo ad operare. Quindi la prima cosa che faremo è costruire una bella tabella infinita in cui sia le righe che le colonne sono etichettate coi naturali. Riempiremo questa tabella con i sottoinsiemi dei naturali nel seguente modo: per ogni riga e colonna metteremo un uno o uno zero a seconda se nel sottoinsie-

Non funziona!

Sì, il procedimento descritto nell'articolo per trasformare un insieme di numeri in un naturale (e viceversa) non funziona in tutti-tutti i casi. Nasconde un piccolo bug che invitiamo i lettori a scoprire, prima di proseguire nella lettura di questo riquadro.

Infatti, per sequenze di numeri che terminano per qualche zero dà risultati non attendibili. Ad esempio, la sequenza (4, 2, 3, 1) che come visto nell'articolo è generata dal numero naturale 126000, non è l'unica a generarlo. Ad esempio anche (4, 2, 3, 1, 0) genera 126000 come qualsiasi altra sequenza che inizi così e termini con un numero qualsiasi di zeri. Per attuare una codifica a tutti gli effetti biunivoca è possibile procedere in due modi. Si può aggiungere come primo elemento della sequenza la cardinalità di questa. Ovvero la sequenza (4,2,3,1) prima della codifica viene trasformata in (4,4,2,3,1) dove il primo 4 indica di quanti elementi è fatto l'insieme da cui siamo partiti. Così, l'insieme (4,2,3,1,0) diventa (5,4,2,3,1,0), (4,2,3,1,0,0,) diventa (6,4,2,3,1,0,0) e così via. Al momento della decodifica il primo valore trovato indicherà di quanti numeri è composta la sequenza in modo da aggiungere eventualmente degli zeri in fondo fino a raggiungimento del numero di elementi dovuto. Resta inteso che il tutto continua a funzionare anche con sequenze non contenenti alcuno zero.

Un altro metodo per aggirare l'ostacolo è quello di aggiungere un uno a tutti gli elementi dell'insieme da cui siamo partendo. In questo modo siamo sicuri di non trovarci alcuno zero tra i piedi quando facciamo la trasformazione. La sequenza (4,2,3,1,0) diventa allora (5,3,4,2,1) che non creerà nessun problema col procedimento descritto nell'articolo. Analogamente, quando otterremo una sequenza scomponendo in fattori primi un naturale, dovremmo ricordarci di sottrarre uno ad ogni elemento per ottenere la sequenza per noi significativa.

me corrispondente alla riga in considerazione è contenuto o no l'elemento corrispondente nella colonna (vedi figura 3). Ad esempio, se il primo sottoinsieme contiene i numeri 0, 1, 2, 3 nella tabella, alla prima riga metteremo la sequenza binaria:

1 1 1 1 0 0 0 0 0 0 .....

Se nella seconda riga vogliamo mettere i numeri pari, la sequenza binaria corrispondente sarà:

1 0 1 0 1 0 1 0 1 ....

E così via per tutti i sottoinsiemi dei naturali. Se riuscissimo a fare una cosa simile, avremmo effettuato una enumerazione dei sottoinsiemi dei naturali: avendo la tabella potremmo sapere qual è il sottoinsieme corrispondente alla posizione X oppure preso un sottoinsieme potremmo stabilire in quale posizione appare in tabella, semplicemente cercandolo. Non importa il fatto che la tabella è di dimensione infinita: se ci assicurassero che qualsiasi sottoinsieme è contenuto in questa la ricerca avrebbe prima o poi termine. Il fatto è che si dimostra che una tabella siffatta non può contenere tutti i sottoinsiemi dei naturali: ce ne sarebbero sempre un certo numero esclusi. Dimostrazione. Abbiamo detto che ogni sottoinsieme riportato in tabella è codificato come una sequenza infinita di uno e di zero. Per dimostrare che tale tabella non contiene tutte le possibili sequenze di uno e di zero (e conseguentemente tutti i sottoinsiemi dei naturali) basta trovare una sequenza che certamente non è contenuta nella tabella. Detto fatto: basta prendere tutti gli elementi di cui è composta la diagonale principale della tabella e complementarla (ovvero tutti gli zero diventano uno e viceversa). Dato che la tabella è infinita, la sequenza così ottenuta sarà anch'essa infinita; quindi se nella tabella fossero elencate tutte, dovremmo aspettarci di trovare anche la nostra, da qualche parte. Ma così non è: infatti, avendo complementato gli elementi, la nostra sequenza non potrà comparire nella tabella dato che ovunque proviamo a confrontarla, perlomeno differirà per l'elemento che incrocierà la diagonale.

Dunque non tutte le sequenze sono presenti in tabella, ovvero non abbiamo enumerato tutti i sottoinsiemi di N, ovvero ancora i sottoinsiemi di N non possono essere messi in corrispondenza biunivoca con i naturali. Quanto detto si traduce nella disequazione:

$$\#N < \#P(N)$$

e si legge: la cardinalità dei naturali è minore di quella delle due parti (P (N) = parti di N). Ricordando che dalla disequazione precedente si leggeva che la cardinalità delle funzioni calcolate dal formalismo S è minore o uguale della cardinalità dei naturali, possiamo concludere che:

$$\#F_S < \#P(N)$$

Funzioni binarie

Prima di scrivere la nostra prossima disequazione occorre notare che manipolare sottoinsiemi dei naturali è la stessa cosa che avere a che fare con le funzioni binarie. Una funzione binaria è una funzione che accetta in input un numero naturale e restitusce uno zero o uno, a seconda di come è fatta la funzione stessa. Un sottoinsieme dei naturali può essere visto come la funzione binaria che preso un qualsiasi naturale restituisce 1 se il naturale in ingresso appartiene al sottoinsieme,0 altrimenti. Ad esempio la funzione binaria corrispondente al sottoinsieme dei numeri pari sarà fatta in modo da valere 1 su tutti i numeri divisibili per due, 0 sugli altri. E dato che ogni sottoinsieme può essere visto come una funzione binaria e che ogni funzione binaria può essere vista come un sotto insieme di N, la cardinalità di parti di N sarà pari a quella delle funzioni binarie:

$$\#P(N) = \#F_B$$

Arrivati a questo punto, non resta altro da dire che le funzioni binarie sono certamente «meno» di tutte le funzioni dai naturali ai naturali in quanto quest'ultime contengono sicuramente anche le binarie e le binarie possono essere viste come funzioni dai naturali ai naturali che «casualmente» restituiscono solo zero o uno (che sono numeri naturali). Scriveremo allora:

$$\#F_B \leq \#F$$

Ricollegandoci alle disequazioni precedenti, dato che queste godono della proprietà transitiva, essendo dello stesso verso, possiamo concludere:

$$\#F_S < \#F$$

che si legge: la cardinalità delle funzioni calcolate dal formalismo S è minore strettamente della cardinalità di tutte le funzioni dai naturali ai naturali.

Come volevasi dimostrare. MC

Cardinalità finite e transfinite

Più volte nell'articolo di queste pagine è menzionato il termine «cardinalità» come proprietà dell'insieme che stiamo trattando. In questo riquadro metteremo un po' d'ordine a tutta la faccenda, specialmente per quel che riguarda gli insiemi infiniti. Ma procediamo con ordine. Un insieme finito è un insieme composto da un numero non infinito di elementi. In particolare un insieme si dice finito quando i suoi elementi possono essere messi in corrispondenza biunivoca con un segmento iniziale dell'insieme dei naturali. Se tale corrispondenza avviene con i numeri 0, 1, 2, ..., P si dice che la cardinalità dell'insieme è P – 1. In parole povere la cardinalità di un insieme finito è pari al numero dei suoi elementi. Quando si passa ad insiemi infiniti, non è possibile quantificare la loro cardinalità ma solo confrontarla con quella di altri insiemi finiti o infiniti. In generale, due insiemi hanno la stessa cardinalità se e solo se gli elementi del primo possono essere messi in corrispondenza biunivoca (1 a 1) con gli elementi del secondo e, naturalmente viceversa. Questa relazione vale sia per gli insiemi finiti che per quelli infiniti: la differenza tra i due è che mentre per i primi è impossibile mettere in corrispondenza biunivoca un insieme con un suo sottoinsieme, per quelli infiniti è possibile. Facciamo un paio di esempi chiarificatori. Se prendo i primi 100 numeri e provo e metterli in corrispondenza biunivoca con i primi 50, mi accorgerò, una volta terminati i 50 del secondo insieme, che non è possibile: mi resteranno ben 50 elementi del primo che non saprò con chi accoppiare. Nel campo dell'infinito ciò non accade. Ad esempio posso mettere in corrispondenza biunivoca i naturali con i soli numeri pari che sono un sottoinsieme proprio dei primi. Per farlo dovrò solo escogitare un trucchetto per far corrispondere ad ogni naturale un numero pari e viceversa: basta prendere le due funzioni 2x e x/2. Per passare da un naturale qualsiasi univocamente a un numero pari basterà moltiplicarlo per due; la corrispondenza inversa si ha prendendo un pari e dividendolo per due. Sembra l'arte dei pazzi, ma abbiamo appena dimostrato che la cardinalità dei numeri pari è uguale a quella di tutti i numeri. E non la metà come si sarebbe tentati a pensare. Tra insiemi infiniti, per decidere se un insieme è o non è della stessa cardinalità dei naturali occorre dimostrarlo: se lo è basta fornire la regola di corrispondenza biunivoca tra i due, se non lo è bisogna (di solito) dimostrare che se lo fosse si perverrebbe a un assurdo.

Nel corso dell'articolo abbiamo ad esempio dimostrato che l'insieme delle sequenze finite di numeri ha la stessa cardinalità dei naturali. Analogamente si è dimostrato che se le sequenze possono essere anche infinite non è possibile enumerarle e quindi metterle in corrispondenza biunivoca con i naturali. Per la cronaca, parti di N, può essere messo in corrispondenza biunivoca con i numeri reali ovvero ha la cardinalità del continuo.