Ti piacciono le riviste di meccanica? Settant'anni di macchine di Turing

Francesco Belardinelli

30 agosto 2005

Indice

1 Algoritmi e procedure effettive

1.1 Che cosa è un algoritmo?

Algoritmo o procedura effettiva: Regola meccanica o metodo automatico o programma per eseguire una qualche operazione (matematica).

Il termine deriva dal nome del matematico persiano Al-Khowarizmi (780-860), attivo a Baghdad presso la corte del califfo Al Ma'mun, che viene considerato l'inventore della moderna algebra, nonché del termine 'algebra'.

Al-Khowarizmi riteneva che ogni problema matematico, indipendentemente dalla sua difficoltà, potesse essere risolto se suddiviso in passi più semplici e derivata logicamente la risposta corretta.

Alcuni esempi:

1. preparare un piatto seguendo una ricetta di cucina;
2. prenotare un volo in aereo su internet;
3. dato n, trovare l'n-esimo numero primo (crivello di Eratostene);
4. trovare il massimo comune divisore di due numeri (algoritmo di Euclide);
5. fare la somma di due numeri x e y.

Ma nella vita di tutti i giorni ci troviamo di fronte a compiti che possono essere svolti per mezzo di procedure meccaniche (riordinare i canali televisivi sul televisore).

Solitamente si ritiene che i lavori che possono essere svolti in modo meccanico non implichino un comportamento particolarmente intelligente, in quanto non necessitano di un apporto creativo.

Visualizzazione di un esempio attraverso i diagrammi di flusso.

L'esempio appena illustrato mostra due caratteristiche fondamentali degli algoritmi:

(a) l'algoritmo si compone in un numero finito di passaggi;

(b) ogni passaggio si conclude in un tempo finito.

Quindi l'intero algoritmo termina e il risultato appare dopo un tempo finito.

La finitezza è una caratteristica essenziale dell'algoritmo.

1.2 Perchè sono importanti gli algoritmi?

- Sono la base della teoria dei calcolatori: i computer lavorano attraverso algoritmi (programmi), ciò che un calcolatore può o non può calcolare in linea di principio - indipendentemente da questioni di memoria o potenza - è determinato dalla possibilità o impossibilità di trovare algoritmi atti allo scopo.
- Rappresentano un modello per il comportamento intelligente.

1.3 Aspetto filosofico della questione

Chiarire concettualmente la nozione informale di procedimento effettivo che cosa sia un procedimento effettivo e quali siano i suoi caratteri formali.

1.4 Osservazioni preliminari

- Ci occuperemo di algoritmi sui numeri naturali $\mathbb{N}$ (i numeri 0,1,2,3,4...).
- Quando un'algoritmo è usato per calcolare i valori di una funzione numerica, allora la funzione in questione è definita *computabile*.

Una nozione è interessante solo se non è banale. Ecco un esempio di funzione definita tramite una procedura non algoritmica:

$$g(n) = \begin{cases} 1 & \text{se c'è una sequenza di esattamente } n \text{ consecutivi '7' nell'espansione decimale di } \pi; \\ 0 & \text{otherwise.} \end{cases}$$

Per calcolare g con input n, abbiamo a disposizione il seguente algoritmo:

1. Svolgi l'espansione decimale di π greco fino all'n-esima cifra.
2. Se vi è una sequenza di esattamente n consecutivi '7', allora produci output 1, altrimenti svolgi l'espansione decimale di π greco fino all'$n+1$-esima cifra.

Ma nel caso in cui non vi sia una sequenza di esattamente n consecutivi '7' nell'espansione decimale di π, questa procedura non termina mai, quindi non produce nessun output.

La conclusione è che non esiste alcun algoritmo per calcolare g, che perciò non è computabile.

2 Le macchine di Turing

2.1 Alan Mathison Turing

Alan Mathison Turing (1912-54), matematico:

1939-45 Durante la Seconda Guerra Mondiale, lavora nel laboratorio di Bletchley Park, il quartier generale delle comunicazioni di guerra del governo britannico, dove decodifica il codice E.N.I.G.M.A. usato per le comunicazioni dalla Germania nazista.

1948 Partecipa alle qualificazioni per la squadra britannica di maratona alle Olimpiadi del 1948.

Nell'articolo *On computable numbers, with an application to the Entscheidungsproblem*, scritto nel 1936 e apparso nel '37, Turing introduce un dispositivo divenuto poi famoso come *Macchina di Turing* - MT in breve - per formulare una definizione rigorosa della nozione di *effettivo*.

Il punto di partenza di Turing è l'analisi di un 'computer' - uomo o macchina - che implementa un algoritmo (svolge dei calcoli) con carta e penna, avendo a disposizione due tipi di azione:

1. scrivere o cancellare un simbolo;
2. trasferire l'attenzione da una parte all'altra del foglio.

Le azioni del 'computer' sono determinate (i) dal simbolo che è visualizzato nel momento presente, e (ii) dallo stato corrente in cui si trova il 'computer'.

2.2 Come è fatta una macchina di Turing?

Una macchina di Turing M è composta da:

- Una nastro infinito verso destra, diviso in caselle per la lunghezza. Ciascuna cella sul nastro contiene uno e un solo simbolo dall'alfabeto $\{0, 1\}$. Al massimo un numero finito di celle contiene il simbolo 1, nelle altre appare 0.
- Un cursore che si sposta da una casella all'altra del nastro.

Una macchina di Turing è capace di compiere tre operazioni:

- Cancellare il simbolo nella cella che il cursore sta leggendo e rimpiazzarlo con un altro simbolo.
- Spostare il cursore una casella a destra della casella che sta leggendo.
- Spostare il cursore una casella a sinistra della casella che sta leggendo.

Ad ogni momento della computazione, la macchina di Turing M si trova in uno stato q_i. Questi stati della computazione sono in numero finito $q_1, \ldots, q_n$.

Infine la macchina di Turing M contiene un programma, cioè una successione di istruzioni del tipo q_k, n, α, q_j, dove q_k, q_j sono stati di M, n può essere uguale a 0 o a 1, α può essere uguale a 0, a 1 o al comando di spostamento a destra D, o al comando di spostamento a sinistra S.

Ad esempio la quadrupla $q_k, 0, D, q_j$ corrisponde intuitivamente al comando:

- Se ti trovi nello stato q_k e la casella che stai leggendo contiene il simbolo 0, allora spostati una casella a destra e mettiti nello stato q_j.

Importante: ciò che distingue una macchina di Turing M da un'altra è proprio il programma di M, quindi si identifica M con il suo programma.

2.3 Come funziona una macchina di Turing?

Si assume che la computazione inizi nello stato q_0, in cui il cursore legge la casella più a sinistra del nastro. La macchina di Turing M prosegue nella computazione in accordo con le istruzioni contenute nel programma. La computazione termina quando M è nello stato q_m, il suo cursore legge il simbolo n e nel programma di M non ci sono istruzioni del tipo q_m, n, α, q_l.

2.4 Cosa calcola una macchina di Turing?

Benchè la struttura ed il funzionamento di una macchina di Turing siano estremamente semplici, queste ultime sono estremamente potenti, tant'è vero che riescono a calcolare tutto ciò che calcola un moderno computer - anche se il tempo impiegato per la computazione è molto maggiore - e qualcosa di più.

Esempio: esiste una macchina di Turing M per calcolare la somma tra due numeri naturali x e y, in cui:

- Le celle del nastro contengono $x + 1$ occorrenze di 1, seguite da 0 e da altre $y + 1$ occorrenze di 1, tutte le altre caselle contengono 0. Le prime $x + 1$ occorrenze di 1 rappresentano il numero x, mentre le altre $y + 1$ occorrenze di 1 rappresentano il numero y.

- M contiene il seguente programma:

$$q_1, 1, D, q_1$$
$$q_1, 0, 1, q_2$$
$$q_2, 1, D, q_2$$
$$q_2, 0, S, q_3$$
$$q_3, 1, 0, q_4$$
$$q_4, 0, S, q_5$$
$$q_5, 1, 0, q_6$$

Il comportamento di questa macchina di Turing può essere visualizzato nel seguente modo per $x = 3$ e $y = 2$.

Una funzione è definita computabile se e solo se è computabile da una macchina di Turing.

Le macchine di Turing possono anche essere composte tra di loro. Ad esempio, se abbiamo una macchina per computare la somma di due numeri ed un'altra per calcolare il doppio di un numero, possiamo unirle assieme - modificando opportunamente la numerazione degli stati della seconda MT - per ottenerne una terza che calcoli la funzione $2(x + y)$.

2.5 Diversi tipi di macchina di Turing

Esistono diverse varianti della precedente definizione della macchina di Turing:

- Il nastro è infinito nelle due direzioni destra/sinistra.
- Il nastro è infinito in due dimensioni.
- Il cursore può compiere spostamenti di più caselle.
- La macchina di Turing ha più cursori.
- L'alfabeto contiene un numero arbitrario finito di simboli.
- La macchina di Turing può essere non deterministica.

Tutte queste formulazioni della MT sono equivalenti alla precedente: per qualsiasi computazione eseguibile da una di queste varianti della MT, esiste una MT che la esegue.

2.6 La tesi di Church

Osservazione banale: Se una funzione è computabile da una macchina di Turing allora è computabile in senso informale.

Tesi di Church-Turing: Una funzione è computabile in senso informale sse è computabile da una macchina di Turing.

Questa affermazione non è dimostrabile, al massimo refutabile da una funzione f che sia computabile in senso informale ma non computabile da una MT. Vi sono però due motivi per ritenerla vera:

1. Una simile funzione f non è stata ancora trovata, cioè, per tutte le funzioni computabili in senso informale, è risultato possibile trovare una macchina di Turing che le computasse.

2. Precisazioni formali indipendenti della nozione di funzione computabile individuano la medesima classe di funzioni:

   - funzioni ricorsive di Gödel (1936);
   - funzioni λ-definibili di Church (1936);
   - sistemi di Post (1943);
   - algoritmi di Markov (1951);
   - macchine universali a registri di Shepherdson e Sturgis (1963).

   Se una funzione è computabile secondo una delle precedenti proposte, allora è computabile anche per tutte le altre.

3 Lo studio delle macchine di Turing

Le macchine di Turing rappresentano un'idealizzazione del funzionamento di un calcolatore: se qualcosa è computabile da una macchina di Turing, possiamo sperare che prima o poi qualcuno costruisca un calcolatore in grado di eseguire la medesima operazione.

Ma se in linea di principio non esiste alcuna MT in grado svolgere una determinata operazione, allora, per la tesi di Church, non vi è possibilità che si riesca a progettare un computer capace di elaborare il problema in questione.

Lo studio delle macchine di Turing ci fornisce i limiti di ciò che è - e sarà mai - effettivamente calcolabile da un computer.

3.1 La codifica delle MT

Per poter considerare alcuni risultati fondamentali, dobbiamo prima osservare certi fatti.

Innanzitutto è possibile fare in modo che le macchine di Turing parlino delle stesse macchine di Turing, infatti è possibile codificare una macchina di Turing M in un numero naturale n^M, soprattutto è possibile farlo utilizzando funzioni computabili.

Ad esempio possiamo assegnare ai simboli '0', '1', 'D', 'S', i primi quattro numeri naturali 0, 1, 2, 3, agli stati $q_1, \ldots, q_n$ i numeri naturali da 4 a n+4.

All'istruzione $q_1, 1, D, q_1$ verrà assegnato il numero $\phi(\phi(\phi(q_1^*, 1^*), D^*), q_1^*) = 4463$, dove $\phi(x, y) = 2^y(2x+1)-1$ è una funzione computabile e α^* è il numero corrispondente al simbolo α

Al programma per sommare due numeri verrà assegnato il codice $2^{I_1} + 2^{I_1+I_2+1} + 2^{I_1+I_2+I_3+2} + 2^{I_1+I_2+I_3+I_4+3} + 2^{I_1+I_2+I_3+I_4+I_5+4} + 2^{I_1+I_2+I_3+I_4+I_5+I_6+5} + 2^{I_1+I_2+I_3+I_4+I_5+I_6+I_7+6}$, dove I_i è l'i-esima istruzione del programma, cioè il numero naturale $2^{4463} + 2^{6607} + 2^{17582} + 2^{39022} + 2^{51950} + 2^{170478} + 2^{238574}$.

In modo analogo, ad ogni macchina di Turing M associamo un numero naturale n^M. Quindi possiamo enumerare tutte le MT: $M_1, M_2, \ldots$, e dimostrare alcuni fatti fondamentali sulle capacità computazionali delle macchine di Turing.

3.2 Esiste una funzione non computabile

Consideriamo la seguente funzione f sui numeri naturali, definita come:

$$f(n) \quad = \quad \begin{cases} M_n(n) + 1 & \text{se la computazione di } M_n \text{ con input } n \text{ termina,} \\ 0 & \text{altrimenti.} \end{cases}$$

Supponiamo che questa funzione sia computata da una qualche macchina di Turing con codice k. Consideriamo i due casi possibili:

1. Se la computazione di M_k con input k termina, allora $M_k(k) = f(k)$, in quanto M_k computa la funzione f. D'altro canto per definizione $f(k) = M_k(k) + 1$, quindi arriviamo alla contraddizione $M_k(k) = f(k) \neq f(k) = M_k(k) + 1$.

2. Se la computazione di M_k con input k non termina allora il valore $M_k(k) = f(k)$ è indefinito. D'altro canto per definizione $f(k) = 0$, quindi arriviamo di nuovo alla contraddizione $f(k) \neq f(k)$.

Il risultato è che non esiste una macchina di Turing in grado di computare f.

Per mezzo della Tesi di Church possiamo concludere che la funzione f non è computabile *tout court*.

3.3 La macchina di Turing Universale

L'importanza di codificare le macchine di Turing in numeri naturali risiede anche nella possibilità di utilizzare questi codici come input per altre MT. Questa idea trova una delle sue più chiare applicazioni nei moderni calcolatori, in cui i programmi e i dati vengono inseriti nella macchina in un stesso linguaggio binario.

Partendo da questa idea si dimostra che una sola macchina è capace di svolgere le computazione di diverse macchine. In particolare esiste una MT - chiamata la macchina di Turing universale - che è in grado di svolgere qualsiasi operazione eseguibile da una MT.

Come funziona questa sorta di supercalcolatore?

Supponiamo che vogliamo fare sommare alla MTU i numeri 2 e 3.

1. Inseriamo come input nella macchina di Turing universale il codice della MT per sommare due numeri, cioè $2^{4463} + 2^{6607} + 2^{17582} + 2^{39022} + 2^{51950} + 2^{170478} + 2^{238574}$, 2 e 3.

2. La MTU decodifica il codice, questo passaggio è possibile in quanto abbiamo specificato che la codificazione avveniva attraverso funzioni computabili, e ricava il programma per sommare due numeri.

3. A questo punto la macchina universale applica il programma per sommare due numeri agli input 2 e 3.

Quello che abbiamo appena descritto è un algoritmo informale, non una macchina di Turing, ma sempre per la tesi di Church, se esiste un procedura informale per svolgere un'operazione, allora la medesima operazione è eseguibile da una macchina di Turing, in questo caso la MTU.

Allo stesso modo qualsiasi operazione eseguibile da una MT può essere svolta dalla macchina di Turing universale.

3.4 Il problema della fermata

Un altro importante risultato della teoria della computazione, riguarda la possibilità di stabilire se dato un certo input n, una determinata MT M_k termini la computazione oppure no.

La risposta al problema della fermata è negativa: non è possibile avere un metodo generale per sapere se una computazione avrà termine.

Per dimostrare questo fatto, supponiamo per assurdo che abbiamo a disposizione un tale algoritmo e definiamo la seguente funzione f sui naturali:

$$f(k,n) \quad = \quad \begin{cases} 0 & \text{se la computazione di } M_k \text{ con input } n \text{ non termina,} \\ indefinito & \text{altrimenti.} \end{cases}$$

Questa funzione è computabile in senso informale: dati k e n facciamo partire la computazione di M_k con input n. Per ipotesi abbiamo stabilito che è sempre possibile determinare se questa computazione termini o no, quindi è possibile stabilire il valore di $f(k,n)$.

Per la tesi di Church, se f è computabile in senso informale, allora esiste una macchina di Turing M_q che calcola f. A questo punto vediamo cosa succede alla funzione f quando le viene assegnato input (q,q).

1. Se la computazione di M_q con input q termina, allora $M_q(q) = j = f(q, q)$, in quanto M_q computa la funzione f. D'altro canto per definizione $f(q, q) = indefinito$, quindi arriviamo alla contraddizione $f(q, q,) \neq f(q, q)$.
2. Se la computazione di M_q con input q non termina allora il valore $M_q(q) = f(q, q)$ è indefinito. D'altro canto per definizione $f(q, q) = 0$, quindi arriviamo di nuovo a $f(q) \neq f(q)$.

L'ipotesi di avere a disposizione un algoritmo per decidere il problema della fermata porta comunque a contraddizione. La conclusione è che non è possibile stabilire per una qualsiasi input n se una macchina di Turing si fermerà e produrrà un output o se la computazione andrà avanti all'infinito.

Von Neumann commentano in questo modo i risultati della macchina di Turing univesale e il problema della fermata:

> Si può costruire un'automa che può fare qualunque cosa fa un altro automa, ma non si può costruire un'automa che predica il comportamento di un automa arbitrario. In altre parole, si può costruire un organo che fa qualsiasi cosa che può essere fatta, ma non uno che dica se può essere fatta. [**?**]

3.5 Altri problemi irrisolvibili

Nel paragrafo percedente abbiamo stabilito che non potrà mai essere progettato un calcolatore che stabilisca se una computazione qualsiasi termini o meno. Esistono però molti altri risultati sui limiti teorici alla potenza di calcolo degli elaboratori. Ne elenchiamo alcuni di seguito:

- Nessun calcolatore potrà mai stabilire se una macchina di Turing M_k computa la funzione zero, cioè se per ogni $n \in \mathbb{N}$, $M_k(n) = 0$.
- Nessun calcolatore potrà mai stabilire se due macchine di Turing M_j e M_k computano la stessa funzione, cioè se per ogni $n \in \mathbb{N}$, $M_j(n) = M_k(n)$.
- Teorema di Rice: se A è un insieme di macchine di Turing non banale, allora non è possibile stabilire se una macchina di Turing M_k appartiene a A.
- Nessun calcolatore potrà mai stabilire se una formula del calcolo dei predicati è valida.

4 Sviluppi successivi

4.1 Calcolabilità e complessità

Un importante sviluppo della teoria della computabilità riguarda lo studio della complessità degli algoritmi. Infatti per le applicazioni ai calco-

latori, non interessa solo che esista un algoritmo per eseguire una certa operazione, ma che l'algoritmo sia implementabili in un tempo ragionevole. Inoltre per eseguire una stessa operazione possono esistere procedure più o meno efficienti, quindi è importante la ricerca di algoritmi sempre più veloci.

Per poter confrontare l'efficienza dei vari algoritmi bisogna poterne confrontare la complessità. L'unità di misura è il passo della computazione.

In base a come varia la durata della computazione in rapporto alla lunghezza n dell'input si distingue tra algoritmi in tempo polinomiale P, tempo polinomiale non deterministico NP, spazio polinomiale $PSPACE$, tempo esponenziale $EXPTIME$.

Ad esempio la macchina di Turing per calcolare la somma di due numeri x e y, impiega $x + y + 7$ passi di computazione, quindi la somma è un problema risolvibile in tempo polinomiale. Ma è possibile rallentare in modo opportuno il programma, ad esempio facendogli calcolare tutte le sequenze di 0 e 1 di lunghezza $x + y + 2$, in modo che la somma venga calcolata in un tempo esponenziale $2^{f(x,y)}$.

Invece per stabilire se una formula del calcolo delle proposizioni in cui appaiono n lettere proposizionali è valida, bisogna procedere nella computazione per 2^n passi, quindi il problema della validità per il calcolo delle proposizioni è risolvibile in tempo esponenziale.

Sono state dimostrate le seguenti inclusioni $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$, e sappiamo che $P \subset EXPTIME$. Inoltre si ipotizza che $P \subset NP \subset PSPACE \subset EXPTIME$, ma non si è certi nemmeno che $P \subset PSPACE$, né che $NP \subset EXPTIME$.

Problema aperto: $P = NP$, cioè: ogni funzione computabile da una MT non deterministica in tempo polinomiale, è computabile da una MT in tempo polinomiale?

4.2 Intelligenza artificiale

Tesi forte dell'intelligenza artificiale: la mente funziona come una macchina di Turing.

Esempio delle api.

Turing riteneva che le operazioni computabili fossero sufficienti per includere tutte le funzioni mentali eseguite dal cervello, che il cervello umano deve essere in qualche modo organizzato per l'intelligenza, e che l'organizzazione del cervello deve essere descrivibile come una macchina finita a stati discreti.

4.3 Risorse Internet

- Turing Digital Archive
- Alan Turing Home Page
- The Turing's World Homepage

Riferimenti bibliografici