Programmazione 1
CdL Matematica
a.a. 2016-2017

Vincenzo Marra
Università degli Studi di Milano
Dipartimento di Matematica *Federigo Enriques*
1.III.2017 – Parte 1

Alan M. Turing, 1912-1954

Logistica

- **Docente**. V. Marra, Studio 1038.
- **Email**. vincenzo.marra@unimi.it
- **Ricevimento**. Su appuntamento concordato per email o a lezione.
- **Struttura del corso**.
  - **Lezioni frontali** in Aula Chisini. Turno unico: V.M.
  - **Laboratorio** in Aula 2 (via Saldini) o 309 o 307 (via Celoria). Quattro turni:
    - Dr.ssa Giovanna Lavado giovanna.lavado@unimi.it
    - Dr.ssa Violetta Lonati violetta.lonati@unimi.it
    - V.M. vincenzo.marra@unimi.it
    - Dr. Diego Valota diego.valota@unimi.it

Logistica

- **Lezioni frontali.** Il mercoledì dalle 8.45 alle 11.30.
- **Laboratorio.**
  - Il mercoledì:
    - Aula 2, 14.30–17.30 (Lonati)
    - Aula 307, 14.30–17.30 (Valota)
  - Il giovedì:
    - Aula 307, 14.30–17.30 (Lavado)
    - Aula 309, 14.30–17.30 (Marra)
- **Prima lezione di laboratorio.** Mercoledì 

2017.
- **Pagine web.**
  - Del corso. http://marra.di.unimi.it/prog1
  - Del laboratorio. http://marra.di.unimi.it/prog1/lab
  - Pagina web del docente. http://marra.di.unimi.it/

Pagina Iniziale

Lezioni Svolte

Avvisi

28.I.14 La prima lezione del corso si terrà mercoledì 5 marzo 2013 alle ore 9.00 in Aula Chisini.

Programmazione 1

Corso di Laurea in Matematica, Università degli Studi di Milano

Queste sono le pagine del corso di Programmazione 1 del Corso di Laurea in Matematica dell'Università degli Studi di Milano. Il corso prevede una serie di lezioni frontali affiancate da esercitazioni in laboratorio suddivise in quattro turni.

- **Bibliografia.** Il libro di testo del corso è:
  - Dennis M. Ritchie, Brian W. Kernighan. *Il linguaggio C. Principi di programmazione e manuale di riferimento*. Nuova edizione italiana, Pearson Education Italia, Milano 2004.

Agli studenti non frequentanti può essere utile consultare anche:

  - Stephen G. Kochan. *Programmare in C. Introduzione al linguaggio*. Terza edizione, Pearson Education Italia, Milano 2011.

Link

(Nessun link.)

Ultima modifica
28 febbraio 2014

Logistica

Importante

- **Iscrizione turno di laboratorio obbligatoria**. Tramite SIFA.
- **Account laboratorio**.
  - Aula 2: Login & password per il laboratorio del dipartimento, da ritirare presso il sig. Luca Galizzi, piano -1 prima dell'inizio lab.
  - Aule 307 & 309: Login & password dell'email `@studenti.unimi.it`, per eseguire registrazione account UniCloud in aula. Cercare di eseguire registrazione prima dell'inizio lab.
- **Da portare in laboratorio**. Login & password per accedere alle macchine dell'Aula 2, login & password dell'email `@studenti.unimi.it` per Aule 307 & 309.
- **Da portare agli esami**. Login & password per accedere alle macchine dell'Aula 2. Indirizzo di posta elettronica e credenziali d'Ateneo.

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Parole Chiave:

- Problema
- Algoritmo
- Linguaggio
- Macchina calcolatrice
- Programmazione

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Parole Chiave:

- Problema
- Algoritmo
- Linguaggio
- Macchina calcolatrice
- Programmazione

Cominciamo dalla nozione di algoritmo, chiedendo lumi al Santo Patrono dei matematici.

Euclide (ca. 325 a.C. – ca. 265 a.C.)

Prologo

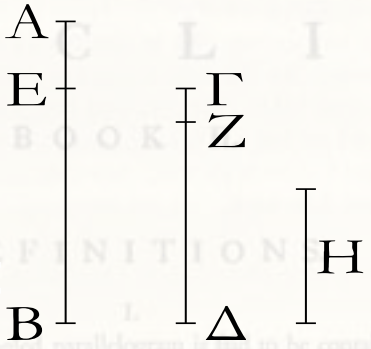
L’algoritmo euclideo

Nel VII Libro degli Elementi, Euclide dimostra costruttivamente che due numeri naturali qualunque hanno un massimo comun divisore (m.c.d):

Nel VII Libro degli Elementi, Euclide dimostra costruttivamente che due numeri naturali qualunque hanno un massimo comun divisore (m.c.d):

β′.

Δύο ἀριθμῶν δοθέντων μὴ πρώτων πρὸς ἀλλήλους τὸ μέγιστον αὐτῶν κοινὸν μέτρον εὑρεῖν.



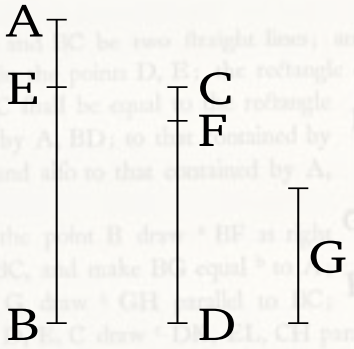
Ἔστωσαν οἱ δοθέντες δύο ἀριθμοὶ μὴ πρῶτοι πρὸς ἀλλήλους οἱ ΑΒ, ΓΔ. δεῖ δὴ τῶν ΑΒ, ΓΔ τὸ μέγιστον κοινὸν μέτρον εὑρεῖν.

Εἰ μὲν οὖν ὁ ΓΔ τὸν ΑΒ μετρεῖ, μετρεῖ δὲ καὶ ἑαυτόν, ὁ ΓΔ ἄρα τῶν ΓΔ, ΑΒ κοινὸν μέτρον ἐστίν. καὶ φανερόν, ὅτι καὶ μέγιστον· οὐδεὶς γὰρ μείζων τοῦ ΓΔ τὸν ΓΔ μετρήσει.

Εἰ δὲ οὐ μετρεῖ ὁ ΓΔ τὸν ΑΒ, τῶν ΑΒ, ΓΔ ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεταί τις ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οὐ λειφθήσεται· εἰ δὲ μή, ἔσονται οἱ ΑΒ, ΓΔ πρῶτοι πρὸς ἀλλήλους· ὅπερ οὐχ ὑπόκειται. λειφήσεταί τις ἄρα ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν ΓΔ τὸν ΒΕ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΕΑ, ὁ δὲ ΕΑ τὸν ΔΖ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΖΓ, ὁ δὲ ΓΖ τὸν ΑΕ μετρείτω. ἐπεὶ οὖν ὁ ΓΖ τὸν ΑΕ μετρεῖ, ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ, καὶ ὁ ΓΖ ἄρα τὸν ΔΖ μετρήσει. μετρεῖ δὲ καὶ ἑαυτόν· καὶ ὅλον ἄρα τὸν ΓΔ μετρήσει. ὁ δὲ ΓΔ τὸν ΒΕ μετρεῖ· καὶ ὁ ΓΖ ἄρα τὸν ΒΕ μετρεῖ· μετρεῖ δὲ καὶ τὸν ΕΑ·

Proposition 2

To find the greatest common measure of two given numbers (which are) not prime to one another.



Let AB and CD be the two given numbers (which are) not prime to one another. So it is required to find the greatest common measure of AB and CD.

In fact, if CD measures AB, CD is thus a common measure of CD and AB, (since CD) also measures itself. And (it is) manifest that (it is) also the greatest (common measure). For nothing greater than CD can measure CD.

But if CD does not measure AB then some number will remain from AB and CD, the lesser being continually subtracted, in turn, from the greater, which will measure the (number) preceding it. For a unit will not be left. But if not, AB and CD will be prime to one another [Prop. 7.1]. The very opposite thing was assumed. Thus, some number will remain which will measure the (number) preceding it. And let CD measuring BE leave EA less than itself, and let EA measuring DF leave FC less than itself, and let CF measure AE. Therefore, since CF measures AE, and AE measures DF, CF will thus also measure DF. And it also measures itself. Thus, it will

(*Trad. inglese R. Kirkpatrick, 2007, sull'ed. Heiberg.*)

La *procedura* descritta da Euclide si può riassumere in modo semi-formale – o, come anche si dice, in *pseudo-codice* – come segue. Si assuma che *a* e *b* siano due interi positivi.

```
procedura mcd(a, b)
    finquando a ≠ b
        se a > b
            a := a − b
        altrimenti
            b := b − a
    restituisci a
```

La procedura di Euclide è un esempio di algoritmo: la prescrizione di una serie di operazioni "elementari" da compiere per risolvere un problema dato.

La Proposizione 2 del VII Libro degli Elementi dimostra che l'algoritmo euclideo è corretto e completo, come si dice in gergo: ossia, per ciascun valore possibile dei dati in ingresso (gli interi a, $b > 0$), termina restituendo il risultato corretto (il m.c.d. di a e b).

Domanda: Quanto rapidamente termina l'algoritmo euclideo in funzione dei dati in ingresso?

Per dare un senso un po' più preciso alla domanda, e rendere possibile una risposta, notiamo che le sottrazioni iterate dell'algoritmo sono equivalenti in modo ovvio a delle divisioni.

Denotiamo con

$$T(a,b)$$

il numero di divisioni richieste dall'algoritmo in funzione dei dati in ingresso a e b.

La *successione di Fibonacci* è: 1,1,2,3,5…, ossia $F_1=F_2=1$ e $F_n := F_{n-1}+F_{n-2}$ per $n>2$.

Domanda: Quanto rapidamente termina l'algoritmo euclideo in funzione dei dati in ingresso?

Per dare un senso un po' più preciso alla domanda, e rendere possibile una risposta, notiamo che le sottrazioni iterate dell'algoritmo sono equivalenti in modo ovvio a delle divisioni.

Denotiamo con

$$T(a,b)$$

il numero di divisioni richieste dall'algoritmo in funzione dei dati in ingresso a e b.

La *successione di Fibonacci* è: 1,1,2,3,5…, ossia $F_1=F_2=1$ e $F_n := F_{n-1}+F_{n-2}$ per $n>2$.

Proposizione (Gabriel Lamé, 1847). Sia $n > 0$ un intero, e siano $a > b > 0$ interi tali che l'algoritmo euclideo applicato ad a e b soddisfi $T(a,b)=n$. Se a è il minimo intero che soddisfa tali condizioni, si ha

$$a = F_{n+2}$$

$$b = F_{n+1}$$

Proposizione (Gabriel Lamé, 1847). Sia $n > 0$ un intero, e siano $a > b > 0$ interi tali che l'algoritmo euclideo applicato ad a e b soddisfi $T(a,b)=n$. Se a è il minimo intero che soddisfa tali condizioni, si ha

$$a = F_{n+2}$$

$$b = F_{n+1}$$

n	1	2	3	4	5	6	7	8	9	10	11	12	13
F_n	1	1	2	3	5	8	13	21	34	55	89	144	233

TABELLA 1. I primi termini della successione di Fibonacci

$a \setminus b$	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	1	2	1					
4	1	1	2	1				
5	1	2	3	2	1			
6	1	1	1	2	2	1		
7	1	2	2	3	3	2	1	
8	1	1	3	1	4	2	2	1

TABELLA 2. Il numero di divisioni eseguito dall'algoritmo euclideo

I termini della successione di Fibonacci ammettono una forma chiusa, nota col nome di *formula di Binet-de Moivre*:

$$F_{n-3} = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$$

dove

$$\phi := \frac{1+\sqrt{5}}{2} \approx 1.61803$$

è la *sezione aurea*.

I termini della successione di Fibonacci ammettono una forma chiusa, nota col nome di *formula di Binet-de Moivre*:

$$F_{n-3} = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$$

dove

$$\phi := \frac{1+\sqrt{5}}{2} \approx 1.61803$$

è la *sezione aurea*.

Usando la forma chiusa qui sopra, dalla proposizione dimostrata da Lamé si può ottenere:

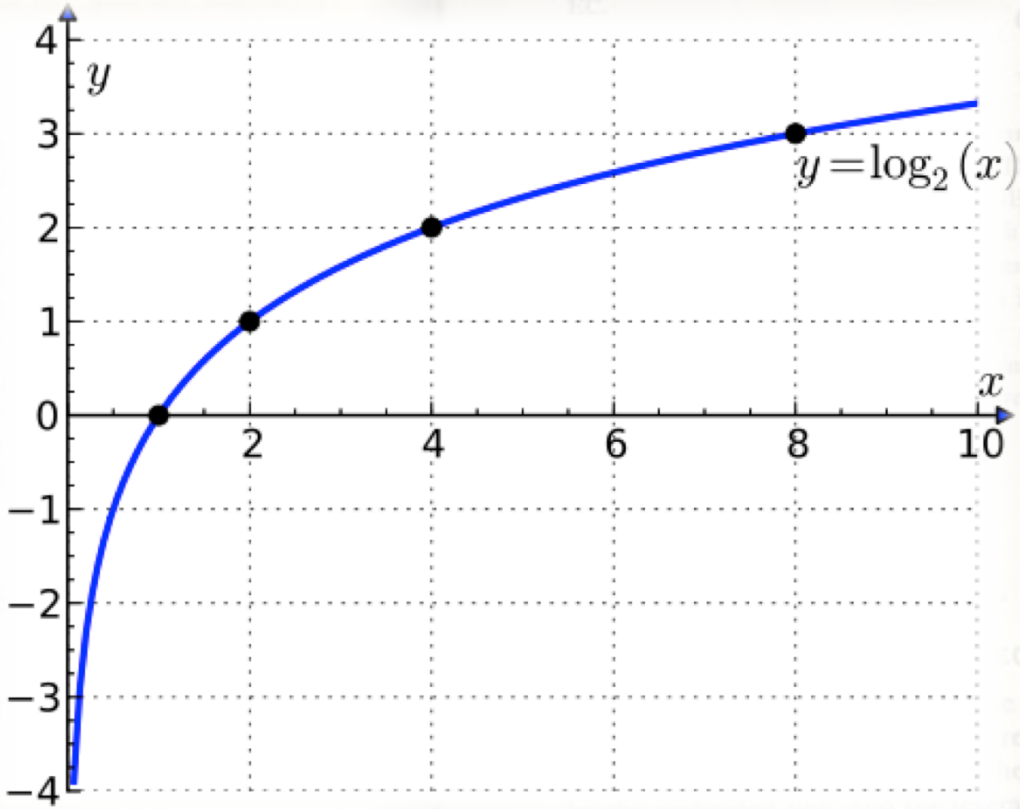
Corollario (Caso pessimo dell'algoritmo di euclide). Dati interi $a > b > 0$, si ha

$$\mathrm{T}(a,b) \leq \lfloor \log_\phi (3-\phi)a \rfloor$$

La funzione

$$\lfloor \log_\phi(3-\phi)a \rfloor$$

cresce molto lentamente rispetto ad *a*. Per esempio, ecco la funzione logaritmo in base *2*:



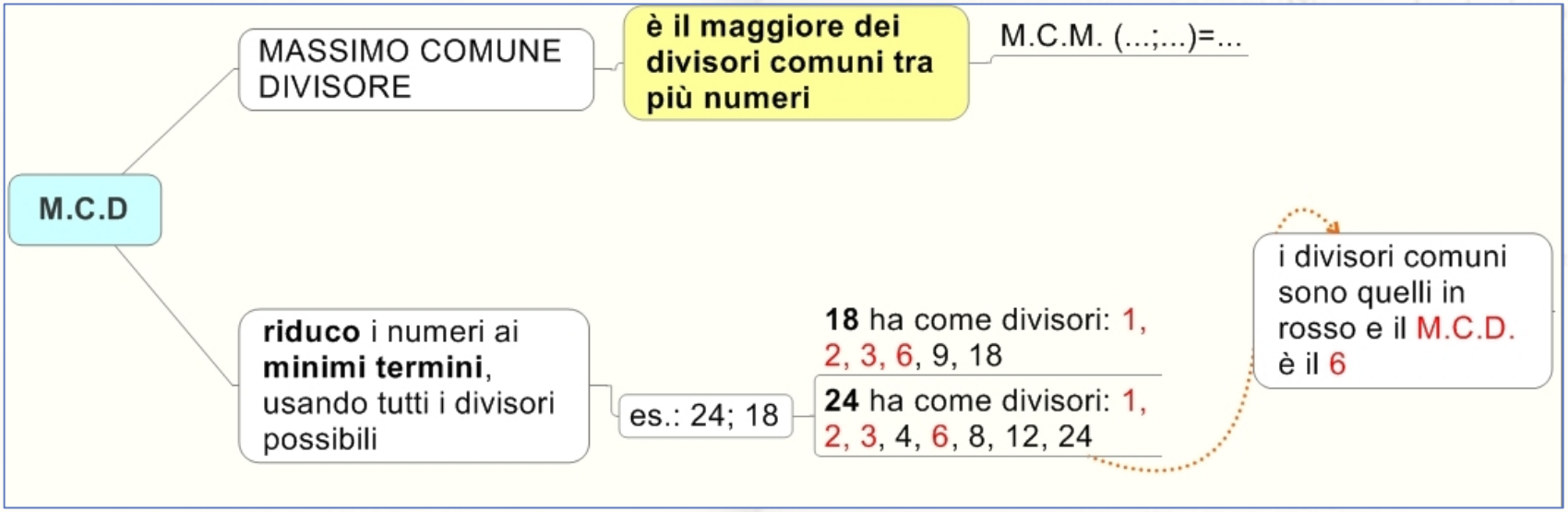
Informalmente: *L'algoritmo euclideo termina in* "pochi passi"*, rispetto alla magnitudine dei valori in ingresso*. In questo senso (impreciso), esso è efficiente.

Informalmente: *L'algoritmo euclideo termina in* "pochi passi"*, rispetto alla magnitudine dei valori in ingresso*. In questo senso (impreciso), esso è efficiente.

Confrontiamo queste conclusioni con un algoritmo che ancora oggi si insegna nelle scuole elementari per risolvere il medesimo problema. Da un sito web per le scuole elementari:

Informalmente: *L'algoritmo euclideo termina in* "pochi passi"*, rispetto alla magnitudine dei valori in ingresso*. In questo senso (impreciso), esso è efficiente.

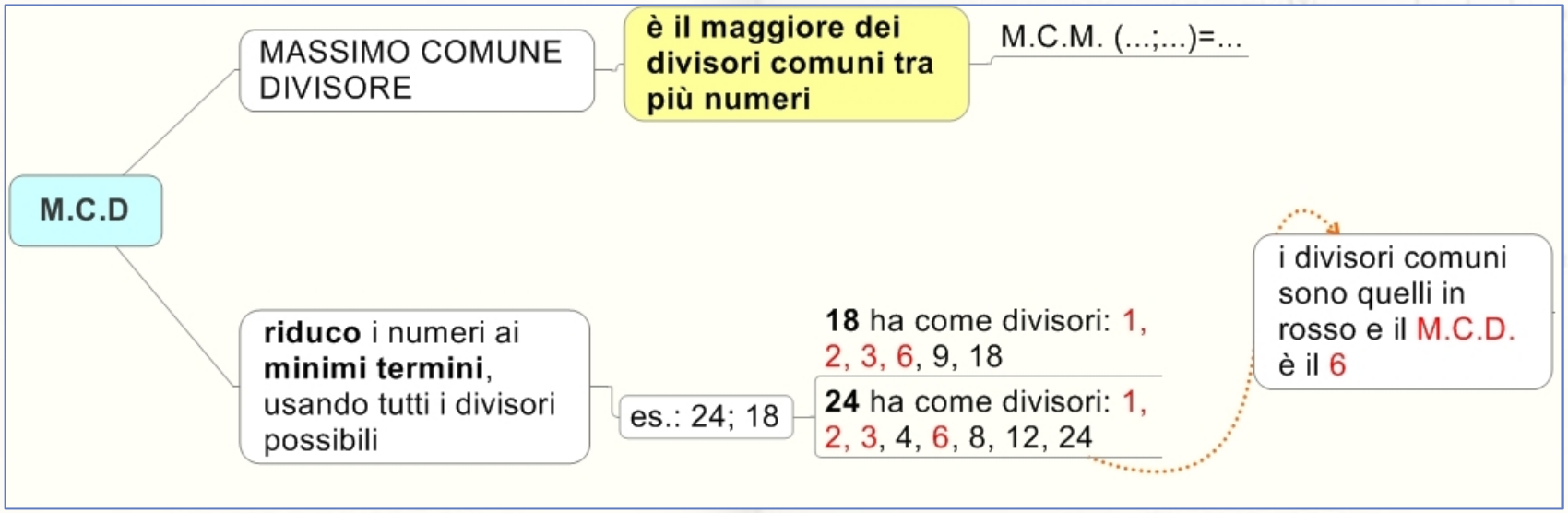
Confrontiamo queste conclusioni con un algoritmo che ancora oggi si insegna nelle scuole elementari per risolvere il medesimo problema. Da un sito web per le scuole elementari:



Tutto si riduce dunque a risolvere il problema ausiliario: scomporre un intero positivo nei suoi fattori primi.

Informalmente: *L'algoritmo euclideo termina in* "pochi passi"*, rispetto alla magnitudine dei valori in ingresso*. In questo senso (impreciso), esso è efficiente.

Confrontiamo queste conclusioni con un algoritmo che ancora oggi si insegna nelle scuole elementari per risolvere il medesimo problema. Da un sito web per le scuole elementari:



Tutto si riduce dunque a risolvere il problema ausiliario: scomporre un intero positivo nei suoi fattori primi.

Fatto Importante: *Non è noto alcun algoritmo efficiente per risolvere tale problema ausiliario*.

Un po' più precisamente: *Tutti gli algoritmi noti che risolvono il problema di scomporre un numero intero positivo x nei suoi fattori primi richiedono un numero di "passi" super-polinomiale: cioè, che cresce con x più di qualunque assegnata funzione polinomiale del numero di cifre (bit) necessarie per esprimere x nel sistema di numerazione binario.*

Un po' più precisamente: *Tutti gli algoritmi noti che risolvono il problema di scomporre un numero intero positivo x nei suoi fattori primi richiedono un numero di "passi" super-polinomiale: cioè, che cresce con x più di qualunque assegnata funzione polinomiale del numero di cifre* (bit) *necessarie per esprimere x nel sistema di numerazione binario.*

Il risvolto pratico di questi fatti è importante. La sicurezza di innumerevoli transazioni elettroniche con carta di credito, per esempio, dipende crucialmente dalla non-esistenza di un algoritmo efficiente per scomporre un numero intero nei suoi fattori primi.

Un po' più precisamente: *Tutti gli algoritmi noti che risolvono il problema di scomporre un numero intero positivo x nei suoi fattori primi richiedono un numero di "passi" super-polinomiale: cioè, che cresce con x più di qualunque assegnata funzione polinomiale del numero di cifre* (bit) *necessarie per esprimere x nel sistema di numerazione binario.*

Il risvolto pratico di questi fatti è importante. La sicurezza di innumerevoli transazioni elettroniche con carta di credito, per esempio, dipende crucialmente dalla non-esistenza di un algoritmo efficiente per scomporre un numero intero nei suoi fattori primi.

Se tale sicurezza dipendesse, invece, dalla possibilità di computare rapidamente il m.c.d., la Proposizione 2 del VII Libro di Euclide – ma non l'algoritmo dei nostri Maestri di scuola elementare – ci toglierebbe il sonno.

Un po' più precisamente: *Tutti gli algoritmi noti che risolvono il problema di scomporre un numero intero positivo x nei suoi fattori primi richiedono un numero di "passi" super-polinomiale: cioè, che cresce con x più di qualunque assegnata funzione polinomiale del numero di cifre* (bit) *necessarie per esprimere x nel sistema di numerazione binario.*

Il risvolto pratico di questi fatti è importante. La sicurezza di innumerevoli transazioni elettroniche con carta di credito, per esempio, dipende crucialmente dalla non-esistenza di un algoritmo efficiente per scomporre un numero intero nei suoi fattori primi.

Se tale sicurezza dipendesse, invece, dalla possibilità di computare rapidamente il m.c.d., la Proposizione 2 del VII Libro di Euclide – ma non l'algoritmo dei nostri Maestri di scuola elementare – ci toglierebbe il sonno.

Il sonno, d'altro canto, l'avevano già perso schiere di matematici che, prima dell'avvento dei moderni computer, si dedicavano a fattorizzare a mano numeri interi grandi – senza perdere il loro tempo, dal punto di vista concettuale, perché come abbiamo appena detto il problema è "difficile". Per cominciare ad apprezzare *quanto* "difficile", ascoltiamo un'altra storia.

David Hilbert (1862–1943)

“Problemi” e Problemi

Facili, difficili e irrisolubili

I *numeri di Mersenne* sono i numeri naturali della forma

$$M_p := 2^p - 1 ,$$

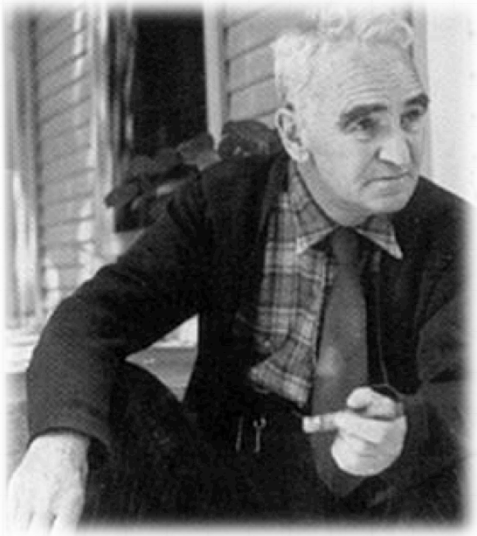
con p intero positivo *primo*. Nel XVII secolo, Padre Mersenne asserì che M_{67} era un numero primo. Lucas dimostrò invece nel 1876 che M_{67} deve avere fattori non banali, senza però riuscire ad esibirne alcuno.

Marin Mersenne (1588–1648)

Édouard Lucas (1842–1891)

Nel 1903, il matematico statunitense Frank Cole riuscì per la prima volta a computare una fattorizzazione non banale di M_{67}. Egli comunicò il suo risultato in occasione di un incontro della Società Matematica Americana. Ecco come lo storico della matematica statunitense di origini scozzesi Eric T. Bell rievocava l'episodio molti anni dopo.

Frank N. Cole (1861–1926)



Eric T. Bell (1883–1960)

Vorrei qui tramandare un episodio storico, prima che tutti i matematici americani della prima metà del ventesimo secolo siano scomparsi. Quando, nel 1911, chiesi a Cole quanto avesse impiegato a fattorizzare M_{67}, mi rispose: "tre anni di domeniche". Sebbene interessante, non è questo l'episodio storico. Nella seduta della Società Matematica Americana che si tenne a New York nell'ottobre del 1903, Cole aveva una comunicazione in programma con il modesto titolo: "Sulla fattorizzazione dei numeri grandi". Quando il presidente annunciò il suo intervento, Cole – che non fu mai uomo di molte parole – si avvicinò alla lavagna e, in silenzio, calcolò il valore di 2 elevato alla sessantasettesima potenza. Con cura, sottrasse 1 al risultato. Senza dire una parola, si spostò quindi verso una porzione pulita di lavagna ed eseguì a mano la moltiplicazione

193.707.721 x 761.838.257.287

I due conti concordavano. […] [La] platea della Società Matematica Americana applaudì vigorosamente l'autore […]. Cole riprese posto senza aver proferito parola. Nessuno gli pose domande.

E. T. Bell. *Mathematics, Queen and Servant of Science*. 1951. p. 228. (Trad. mia.)

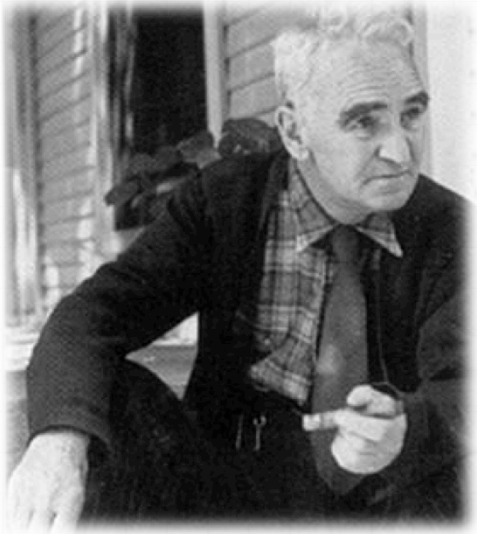
Oggi, un qualunque personal computer (adeguatamente programmato) è in grado di fattorizzare M_{67} in una manciata di secondi.

"Baby" (*Small-Scale Experimental Machine*) alla Victoria University di Manchester, UK, 1948. Servizio televisivo della BBC.

Oggi, un qualunque personal computer (adeguatamente programmato) è in grado di fattorizzare M_{67} in una manciata di secondi.

Sarebbe però un errore concettuale grave ritenere che il "problema" generale di cui Cole affrontò un caso particolare sia banale. Come abbiamo visto, non ci sono algoritmi "efficienti" per risolvere il "problema" generale della scomposizione di un numero intero nei suoi fattori primi. Si tratta di un "problema difficile".

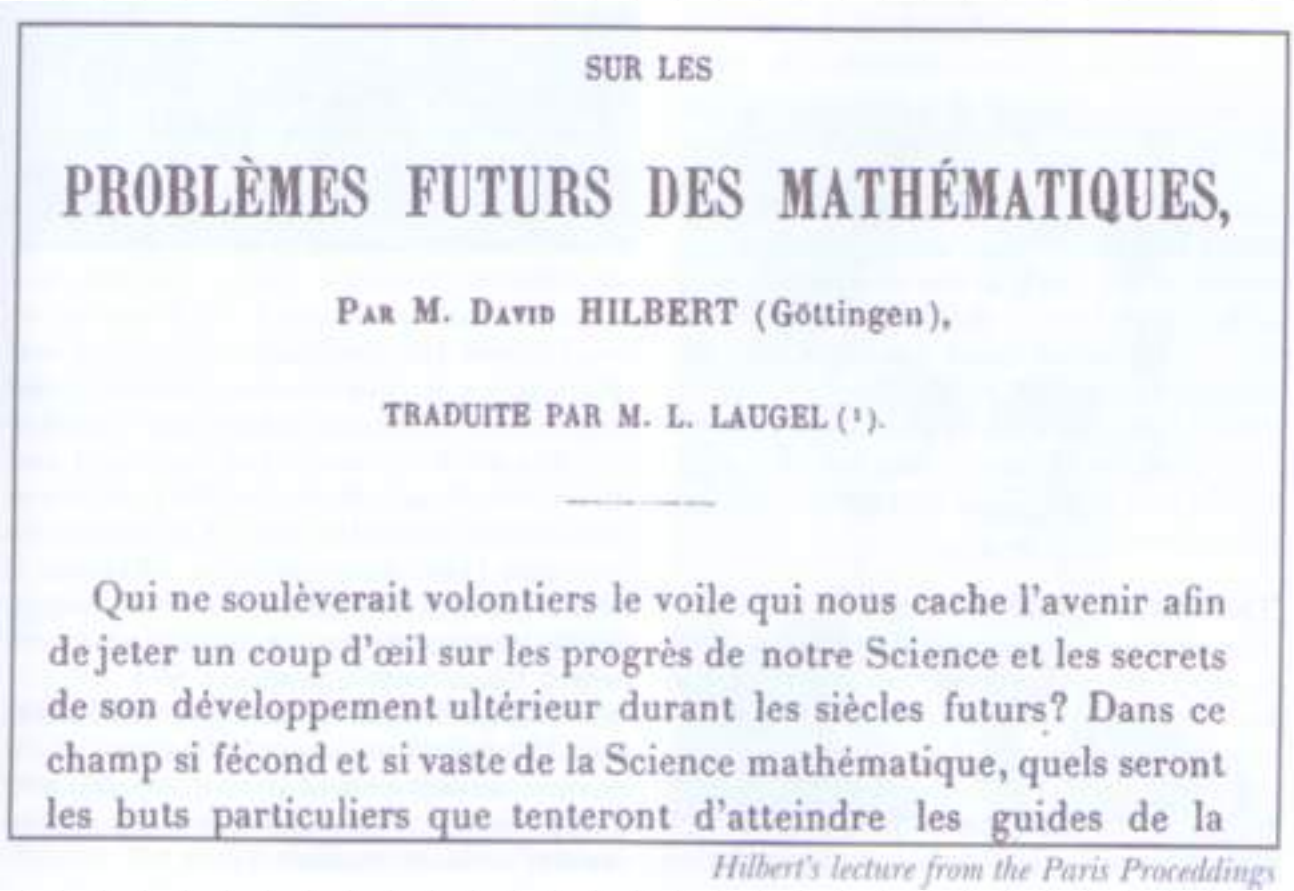
Frank N. Cole (1861–1926)



Eric T. Bell (1883–1960)

Finora abbiamo dunque visto che esistono "problemi facili" (*M.C.D.*), e "problemi difficili" (*fattorizzazione*). C'è dell'altro, però. Esistono anche dei problemi "irrisolubili".

Finora abbiamo dunque visto che esistono "problemi facili" (*M.C.D.*), e "problemi difficili" (*fattorizzazione*). C'è dell'altro, però. Esistono anche dei problemi "irrisolubili".



SUR LES

PROBLÈMES FUTURS DES MATHÉMATIQUES,

PAR M. DAVID HILBERT (Göttingen),

TRADUITE PAR M. L. LAUGEL (1).

Qui ne soulèverait volontiers le voile qui nous cache l'avenir afin de jeter un coup d'œil sur les progrès de notre Science et les secrets de son développement ultérieur durant les siècles futurs? Dans ce champ si fécond et si vaste de la Science mathématique, quels seront les buts particuliers que tenteront d'atteindre les guides de la

Hilbert's lecture from the Paris Proceddings

D. Hilbert, *I problemi futuri della matematica*, Atti del Congresso Internazionale dei Matematici, Parigi, 1900.

Finora abbiamo dunque visto che esistono "problemi facili" (*M.C.D.*), e "problemi difficili" (*fattorizzazione*). C'è dell'altro, però. Esistono anche dei problemi "irrisolubili".

Un'equazione diofantea è un'equazione della forma:

$$p(x_1, \ldots, x_n) = 0$$

dove p è un polinomio a coefficienti interi nelle variabili $x_1,\ldots,x_n$.

Finora abbiamo dunque visto che esistono "problemi facili" (*M.C.D.*), e "problemi difficili" (*fattorizzazione*). C'è dell'altro, però. Esistono anche dei problemi "irrisolubili".

Il Decimo Problema di Hilbert

Trovare una (singola) procedura che permetta, per ogni data equazione diofantea, di determinare tramite un numero finito di operazioni se l'equazione abbia o non abbia soluzioni intere.

Un'equazione diofantea è un'equazione della forma:

$$p(x_1, \ldots, x_n) = 0$$

dove p è un polinomio a coefficienti interi nelle variabili $x_1,\ldots,x_n$.

Finora abbiamo dunque visto che esistono "problemi facili" (*M.C.D.*), e "problemi difficili" (*fattorizzazione*). C'è dell'altro, però. Esistono anche dei problemi "irrisolubili".

Il Decimo Problema di Hilbert

Trovare una (singola) procedura che permetta, per ogni data equazione diofantea, di determinare tramite un numero finito di operazioni se l'equazione abbia o non abbia soluzioni intere.

Per esempio, il metodo di risoluzione delle equazioni di secondo grado (a coefficienti in **Z**) che abbiamo imparato a scuola costituisce una tale procedura nel caso particolare delle equazioni diofantee in una variabile e di grado al più 2: infatti, dalla espressione generale per le due soluzioni in **C**, si determina agevolmente se ciascuna soluzione è in **Z**.

Ma quanto abbiamo imparato a scuola sulle equazioni di secondo grado non si estende al caso generale.

"**Teorema**" (1970). Non esiste alcun algoritmo che risolva il Decimo Problema di Hilbert. Non esiste cioè alcun algoritmo che, avendo come dato in ingresso una qualsivoglia equazione diofantea, termini sempre, con risultato **SI** se l'equazione in ingresso ammette soluzioni intere, e con risultato **NO** se essa non ne ammette.

Ma quanto abbiamo imparato a scuola sulle equazioni di secondo grado non si estende al caso generale.

"**Teorema**" (1970). Non esiste alcun algoritmo che risolva il Decimo Problema di Hilbert. Non esiste cioè alcun algoritmo che, avendo come dato in ingresso una qualsivoglia equazione diofantea, termini sempre, con risultato **SI** se l'equazione in ingresso ammette soluzioni intere, e con risultato **NO** se essa non ne ammette.

Il "Teorema" è dovuto allo sforzo congiunto di molti matematici. È il *Teorema Matiyasevich/MRDP* di Matiyasevich, basato su risultati precedenti di Matiyasevich stesso, Robinson, Davis, e Putnam.

Ma quanto abbiamo imparato a scuola sulle equazioni di secondo grado non si estende al caso generale.

"**Teorema**" (1970). Non esiste alcun algoritmo che risolva il Decimo Problema di Hilbert. Non esiste cioè alcun algoritmo che, avendo come dato in ingresso una qualsivoglia equazione diofantea, termini sempre, con risultato **SI** se l'equazione in ingresso ammette soluzioni intere, e con risultato **NO** se essa non ne ammette.

Il "Teorema" è dovuto allo sforzo congiunto di molti matematici. È il *Teorema Matiyasevich/MRDP* di Matiyasevich, basato su risultati precedenti di Matiyasevich stesso, Robinson, Davis, e Putnam.

In matematica non esistono però "teoremi", ma solo teoremi. Per rimuovere le virgolette dall'enunciato qui sopra, è necessario definire formalmente la nozione di algoritmo. Il fatto che ciò sia possibile, in modo utile e sensato, è una delle conquiste della matematica del XX secolo.

Macchine di Carta

La nozione di Passo di Calcolo

La difficoltà nella definizione formale della nozione di algoritmo risiede nella necessità di isolare con chiarezza la nozione ausiliaria, ma centrale in tutta la teoria della computazione, di passo di calcolo elementare.

La difficoltà nella definizione formale della nozione di algoritmo risiede nella necessità di isolare con chiarezza la nozione ausiliaria, ma centrale in tutta la teoria della computazione, di passo di calcolo elementare.

Nell'algoritmo euclideo, cosa è da prendersi come passo elementare? Le singole sottrazioni? O le divisioni, come abbiamo fatto noi? O altro ancora?

La difficoltà nella definizione formale della nozione di algoritmo risiede nella necessità di isolare con chiarezza la nozione ausiliaria, ma centrale in tutta la teoria della computazione, di passo di calcolo elementare.

Nell'algoritmo euclideo, cosa è da prendersi come passo elementare? Le singole sottrazioni? O le divisioni, come abbiamo fatto noi? O altro ancora?

E in un dato algoritmo per la scomposizione in fattori, quale sarà la nozione appropriata di passo elementare? Sempre che ve ne sia una, non sarà forse diversa da quella appropriata per l'algoritmo euclideo?

La difficoltà nella definizione formale della nozione di algoritmo risiede nella necessità di isolare con chiarezza la nozione ausiliaria, ma centrale in tutta la teoria della computazione, di passo di calcolo elementare.

Nell'algoritmo euclideo, cosa è da prendersi come passo elementare? Le singole sottrazioni? O le divisioni, come abbiamo fatto noi? O altro ancora?

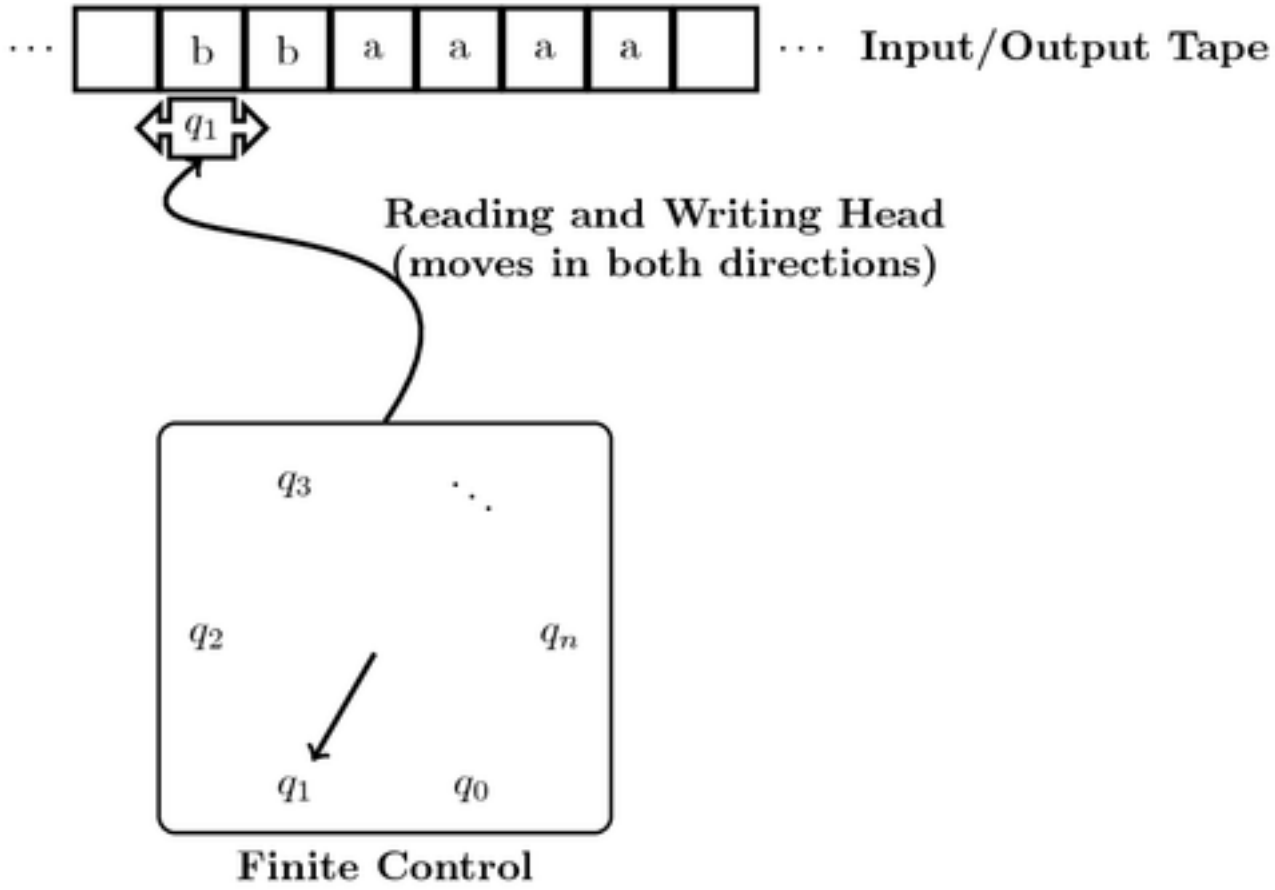
E in un dato algoritmo per la scomposizione in fattori, quale sarà la nozione appropriata di passo elementare? Sempre che ve ne sia una, non sarà forse diversa da quella appropriata per l'algoritmo euclideo?

Ma se infine *ciascun algoritmo si porta dietro la sua propria nozione di passo elementare di calcolo*, cosa rimane dell'idea *prima facie* intuitiva che "un algoritmo è una qualunque procedura che permetta di risolvere un problema tramite un numero finito di passi di calcolo ben specificati"? Nulla, naturalmente.

Nel 1937, il matematico inglese Alan M. Turing, allora venticinquenne, pubblica sui *Proceedings of the London Mathematical Society* un articolo che diverrà celebre: *On computable numbers, with an application to the Entscheidungsproblem*.

In questo lavoro Turing identifica la nozione di passo di calcolo nel modo più semplice e diretto possibile, e per ciò stesso tanto più ammirevole: egli definisce il passo di calcolo relativamente a un *modello astratto di macchina calcolatrice*, che chiamò nel suo lavoro *a-machine*, ma che oggi è nota come la macchina di Turing.

> [Una macchina di Turing è costituita da] una capacità di memorizzazione [finita ma] illimitata, sotto forma di un nastro indefinitamente lungo suddiviso in caselle, su ciascuna delle quali si può scrivere un simbolo [estratto da un fissato alfabeto finito, per esempio l'alfabeto binario 0 e 1.] In ogni dato istante vi è un solo simbolo nella macchina; è detto "il simbolo letto". La macchina può modificare il simbolo letto, e il suo comportamento è in parte determinato da quel simbolo, ma i rimanenti simboli sul nastro non influiscono sul comportamento della macchina. Il nastro, però, può essere fatto scorrere avanti e indietro attraverso [la testina di lettura e scrittura de]la macchina, e ciò costituisce una delle operazioni elementari della macchina. (Turing 1948; trad. mia; enfasi mia.)



Visualizzazione intuitiva di una macchina di Turing

Definizione formale (da Hopcroft e Ullman, 1979)

Una *macchina di Turing* è una settupla

$$(Q, \Gamma, b, \Sigma, \delta, q_0, F),$$

dove:

- Q è un insieme finito e non vuoto di *stati*.
- Γ è un insieme finito e non vuoto, detto *alfabeto* (*del nastro*).
- $b \in \Gamma$ è il simbolo *spazio*.
- $\Sigma \subseteq \Gamma \setminus \{b\}$ è l'*insieme dei simboli in ingresso*.
- $q_0 \in Q$ è lo *stato iniziale*.
- $F \subseteq Q$ è l'insieme degli *stati finali*.
- $\delta \colon Q \setminus F \times \Gamma \to Q \times \Gamma \times \{L, R\}$ è una funzione parziale detta *funzione di transizione*, dove L sta per *spostamento di una casella del nastro a sinistra* ed R per *spostamento di una casella della nastro a destra*.

Cosa computa la macchina di Turing descritta dalla tabella?
(Configurazione iniziale del nastro: tutte le caselle contengono b. Insieme degli stati finali: vuoto.)

Stato	Simbolo sul nastro	Operazione	Stato successivo
q_0	b	Scrivi 0, R	q_1
q_1	b	R	q_2
q_2	b	Scrivi 1, R	q_3
q_3	b	R	q_0

Cosa computa la macchina di Turing descritta dalla tabella?
(Configurazione iniziale del nastro: tutte le caselle contengono b. Insieme degli stati finali: vuoto.)

Stato	Simbolo sul nastro	Operazione	Stato successivo
q_0	b	Scrivi 0, R	q_1
q_1	b	R	q_2
q_2	b	Scrivi 1, R	q_3
q_3	b	R	q_0

La macchina non termina mai, e continua a scrivere sul nastro, da sinistra a destra, la successione di numeri (separati da spazi):

0□1□ 0□1□0□1 ...

Un algoritmo è adesso definibile come *qualunque procedura che possa essere modellata da una macchina di Turing*.

Un algoritmo è adesso definibile come *qualunque procedura che possa essere modellata da una macchina di Turing*.

Un passo elementare di calcolo dell'algoritmo è adesso definibile come *un'operazione elementare della macchina* (cfr. la citazione di Turing *supra*), e diviene una nozione matematica del tutto rigorosa: ad esempio, possiamo convenire che un passo elementare di una macchina di Turing è una singola applicazione della funzione δ nella definizione di Hopcroft e Ullman.

Un algoritmo è adesso definibile come *qualunque procedura che possa essere modellata da una macchina di Turing*.

Un passo elementare di calcolo dell'algoritmo è adesso definibile come *un'operazione elementare della macchina* (cfr. la citazione di Turing *supra*), e diviene una nozione matematica del tutto rigorosa: ad esempio, possiamo convenire che un passo elementare di una macchina di Turing è una singola applicazione della funzione δ nella definizione di Hopcroft e Ullman.

Delicate affermazioni di non esistenza, della forma "Non esiste un algoritmo (=macchina di Turing) tale che …", divengono così suscettibili di dimostrazione matematica. E tutto grazie, appunto, alla *definizione* di Turing: a riprova del fatto che, in matematica, le definizioni sono spesso più importanti dei teoremi.

Un algoritmo è adesso definibile come *qualunque procedura che possa essere modellata da una macchina di Turing*.

Un passo elementare di calcolo dell'algoritmo è adesso definibile come *un'operazione elementare della macchina* (cfr. la citazione di Turing *supra*), e diviene una nozione matematica del tutto rigorosa: ad esempio, possiamo convenire che un passo elementare di una macchina di Turing è una singola applicazione della funzione δ nella definizione di Hopcroft e Ullman.

Delicate affermazioni di non esistenza, della forma "Non esiste un algoritmo (=macchina di Turing) tale che …", divengono così suscettibili di dimostrazione matematica. E tutto grazie, appunto, alla *definizione* di Turing: a riprova del fatto che, in matematica, le definizioni sono spesso più importanti dei teoremi.

La soluzione (negativa) del Decimo Problema di Hilbert ammonta alla dimostrazione dell'impossibilità dell'esistenza di una macchina di Turing con determinate proprietà:

Teorema (1970). Non esiste alcun algoritmo che risolva il Decimo Problema di Hilbert. Non esiste cioè alcuna macchina di Turing che, avendo come dato in ingresso una qualsivoglia equazione diofantea, termini sempre, con risultato **SI** se l'equazione in ingresso ammette soluzioni intere, e con risultato **NO** se essa non ne ammette.

Si dice nel gergo della teoria della computazione che il problema di stabilire (algoritmicamente) se le equazioni diofantee ammettano soluzioni intere è indecidibile.

Si potrebbe compilare una *lunga* lista di problemi di interesse matematico che sono indecidibili nel senso appena illustrato. Poiché però noi dobbiamo occuparci in questo corso di ciò che *si può* programmare, e non di ciò che *non si può* programmare, abbandoneremo l'argomento; con un ultimo commento, però.

Turing stesso dimostrò che esiste un problema concettualmente molto semplice, detto problema dell'arresto (delle macchine di Turing), che è indecidibile. Si tratta, in un certo senso, del prototipo di tutti i problemi indecidibili. Prima della fine della lezione ne riparleremo brevemente .

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Parole Chiave:

- Problema
- Algoritmo
- Linguaggio
- Macchina calcolatrice
- Programmazione

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Si potrebbe, nel resto del corso, studiare la programmazione delle macchine di Turing. Ma poiché il nostro laboratorio ne è sprovvisto, studieremo invece la programmazione dei moderni calcolatori digitali in un linguaggio specifico, che si chiama linguaggio C. Dal punto di vista della decidibilità, o della teoria della computabilità, la scelta di questo specifico linguaggio è irrilevante:

Argomento del Corso

Argomento del corso è la programmazione in un dato linguaggio (il *linguaggio C*) delle macchine calcolatrici (in particolare, i calcolatori digitali, o *computer*) affinché risolvano problemi per via automatica eseguendo programmi che mettono in atto (*implementano*) specifici algoritmi di risoluzione.

Si potrebbe, nel resto del corso, studiare la programmazione delle macchine di Turing. Ma poiché il nostro laboratorio ne è sprovvisto, studieremo invece la programmazione dei moderni calcolatori digitali in un linguaggio specifico, che si chiama linguaggio C. Dal punto di vista della decidibilità, o della teoria della computabilità, la scelta di questo specifico linguaggio è irrilevante:

Tesi di Church-Turing (adattata). Qualunque programma che sarete in grado di scrivere in laboratorio alla fine del corso, potrebbe essere riscritto in modo equivalente – in modo, cioé, che computi i medesimi dati in uscita a fronte dei medesimi dati in ingresso – nel formalismo delle macchine di Turing. E viceversa. Ossia: *Qualunque funzione dai dati in ingresso a quelli in uscita calcolabile da un programma in C è parimenti calcolabile da qualche macchina di Turing, e viceversa.*