Algoritmi, Complessità, Calcolabilità: l'essenza dell'informatica

Gabriele Vanoni

Politecnico di Milano

12 Aprile 2017

Indice

Introduzione

Disclaimer

1. Il contenuto di questa lezione copre normalmente un corso universitario di almeno **50 ore**. Non pretendo quindi di essere esauriente nè che capiate tutto. Lo scopo è quello di farvi vedere alcune delle principali **idee** dell'informatica teorica, in modo che possiate capire di cosa tratta questa materia.

Disclaimer

1. Il contenuto di questa lezione copre normalmente un corso universitario di almeno **50 ore**. Non pretendo quindi di essere esauriente nè che capiate tutto. Lo scopo è quello di farvi vedere alcune delle principali **idee** dell'informatica teorica, in modo che possiate capire di cosa tratta questa materia.
2. Non sempre presenterò gli argomenti in maniera rigorosa e formale perché ciò richiederebbe tecnicismi matematici pesanti. Cercherò di fare affidamento sulla vostra **intuizione**. Per una trattazione completa ma comunque accessibile si vedano i testi in bibliografia.

Informatica, alcune definizioni

"Computer science is no more about computers than astronomy is about telescopes"

Edsger Dijkstra

Informatica, alcune definizioni

"Computer science is no more about computers than astronomy is about telescopes"

Edsger Dijkstra

"I shall be sorry if computer science ever flies apart into two disciplines, one logical and one technological"

Robin Milner

Informatica, alcune definizioni

"Computer science is no more about computers than astronomy is about telescopes"

Edsger Dijkstra

"I shall be sorry if computer science ever flies apart into two disciplines, one logical and one technological"

Robin Milner

"Il padre dell'informatica è l'ingegneria, ma la madre è la logica"

Maria Emilia Maietti

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.
- Storicamente l'informatica nasce nei dipartimenti di **matematica** delle università, venti anni prima della creazione del primo calcolatore elettronico.

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.
- Storicamente l'informatica nasce nei dipartimenti di **matematica** delle università, venti anni prima della creazione del primo calcolatore elettronico.
- Quando si comincia la costruzione dei calcolatori allora entrano in campo gli **ingegneri**.

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.
- Storicamente l'informatica nasce nei dipartimenti di **matematica** delle università, venti anni prima della creazione del primo calcolatore elettronico.
- Quando si comincia la costruzione dei calcolatori allora entrano in campo gli **ingegneri**.
- Oggi anche chi fa un sito web è considerato un informatico.

Un problema anche linguistico

- In italiano il termine informatica è omnicomprensivo. In inglese esistono almeno due differenti locuzioni: **computer science** e **information technology**.
- Storicamente l'informatica nasce nei dipartimenti di **matematica** delle università, venti anni prima della creazione del primo calcolatore elettronico.
- Quando si comincia la costruzione dei calcolatori allora entrano in campo gli **ingegneri**.
- Oggi anche chi fa un sito web è considerato un informatico.
- Ci occuperemo di quella che viene chiamata **informatica teorica**, cioè dei fondamenti matematici alla base della disciplina.

Preliminari matematici

Insiemi

Definizione (assioma di comprensione di Cantor e Frege)

Un'insieme è una collezione di oggetti, caratterizzati da una proprietà.

Insiemi

Definizione (assioma di comprensione di Cantor e Frege)

Un'insieme è una collezione di oggetti, caratterizzati da una proprietà.

Esempio

$\Omega = \{x \in \mathbb{N} | x > 2\}$ i.e.
$\Omega = \{3, 4, 5, ...\}$

Insiemi

Definizione (assioma di comprensione di Cantor e Frege)

Un'insieme è una collezione di oggetti, caratterizzati da una proprietà.

Esempio

$\Omega = \{x \in \mathbb{N} | x > 2\}$ i.e.
$\Omega = \{3, 4, 5, ...\}$

Questa definizione NON è corretta, e porta al **paradosso** di Russell.

Insiemi

Definizione (assioma di comprensione di Cantor e Frege)

Un'insieme è una collezione di oggetti, caratterizzati da una proprietà.

Esempio

$\Omega = \{x \in \mathbb{N} | x > 2\}$ i.e.
$\Omega = \{3, 4, 5, ...\}$

Questa definizione NON è corretta, e porta al **paradosso** di Russell.

La moderna teoria degli insiemi **ZF** vieta la costruzione di questo tipo di insiemi "patologici", utilizzando assiomi più restrittivi.

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Principio di induzione

Sia $P(n)$ una proprietà dei numeri naturali. Se P è verificata per un $c \in \mathbb{N}$ e supponendola vera per $c \leq n \leq k$ posso dimostrarla per $n = k + 1$ allora $P(n)$ è valida per ogni $n \geq c$.

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Principio di induzione

Sia $P(n)$ una proprietà dei numeri naturali. Se P è verificata per un $c \in \mathbb{N}$ e supponendola vera per $c \leq n \leq k$ posso dimostrarla per $n = k + 1$ allora $P(n)$ è valida per ogni $n \geq c$.

Se $P(n)$ è vera per c allora è vera per $c + 1$, e quindi per $c + 2$...

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Principio di induzione

Sia $P(n)$ una proprietà dei numeri naturali. Se P è verificata per un $c \in \mathbb{N}$ e supponendola vera per $c \leq n \leq k$ posso dimostrarla per $n = k + 1$ allora $P(n)$ è valida per ogni $n \geq c$.

Se $P(n)$ è vera per c allora è vera per $c + 1$, e quindi per $c + 2$...

Sfruttando tale principio è possibile costruire **dimostrazioni** di teoremi che riguardino proprietà di numeri naturali.

Dimostrazioni per induzione

Il principio di induzione è una caratteristica intrinseca dei **numeri naturali**, infatti fa parte degli **assiomi di Peano**, le regole che li definiscono.

Principio di induzione

Sia $P(n)$ una proprietà dei numeri naturali. Se P è verificata per un $c \in \mathbb{N}$ e supponendola vera per $c \leq n \leq k$ posso dimostrarla per $n = k + 1$ allora $P(n)$ è valida per ogni $n \geq c$.

Se $P(n)$ è vera per c allora è vera per $c + 1$, e quindi per $c + 2$...

Sfruttando tale principio è possibile costruire **dimostrazioni** di teoremi che riguardino proprietà di numeri naturali.

Esempio (Somma dei primi n naturali)

$\sum_{h=1}^{n} h = \frac{n(n+1)}{2}$. Dimostrazione per induzione su n.

Insieme delle parti $\wp(\Omega)$

Definizione (cardinalità)

La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Insieme delle parti $\wp(\Omega)$

Definizione (cardinalità)

La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Definizione (insieme delle parti)

L'insieme delle parti di un insieme Ω è l'insieme $\wp(\Omega)$ che contiene tutti i suoi sottoinsiemi.

Insieme delle parti $\wp(\Omega)$

Definizione (cardinalità)

La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Definizione (insieme delle parti)

L'insieme delle parti di un insieme Ω è l'insieme $\wp(\Omega)$ che contiene tutti i suoi sottoinsiemi.

Teorema

La cardinalità dell'insieme delle parti di un insieme finito Ω è $2^{|\Omega|}$.

Insieme delle parti $\wp(\Omega)$

Definizione (cardinalità)

La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Definizione (insieme delle parti)

L'insieme delle parti di un insieme Ω è l'insieme $\wp(\Omega)$ che contiene tutti i suoi sottoinsiemi.

Teorema

La cardinalità dell'insieme delle parti di un insieme finito Ω è $2^{|\Omega|}$.

Teorema (di Cantor)

Per ogni insieme Ω, $|\wp(\Omega)| > |\Omega|$.

Insieme delle parti $\wp(\Omega)$

Definizione (cardinalità)

La cardinalità di un insieme Ω finito è n se Ω contiene n elementi e viene indicata con $|\Omega|$.

Definizione (insieme delle parti)

L'insieme delle parti di un insieme Ω è l'insieme $\wp(\Omega)$ che contiene tutti i suoi sottoinsiemi.

Teorema

La cardinalità dell'insieme delle parti di un insieme finito Ω è $2^{|\Omega|}$.

Teorema (di Cantor)

Per ogni insieme Ω, $|\wp(\Omega)| > |\Omega|$.

Il teorema si applica anche agli insiemi infiniti, definendo una **gerarchia**.

Funzioni

Definizione

Una funzione f è una corrispondenza tra due insiemi A e B che associa ad ogni elemento di A uno e un solo elemento di B.

Funzioni

Definizione

Una funzione f è una corrispondenza tra due insiemi A e B che associa ad ogni elemento di A uno e un solo elemento di B.

Ci occuperemo in particolare di funzioni da $\mathbb{N}$ ad $\mathbb{N}$.

Funzioni

Definizione

Una funzione f è una corrispondenza tra due insiemi A e B che associa ad ogni elemento di A uno e un solo elemento di B.

Ci occuperemo in particolare di funzioni da $\mathbb{N}$ ad $\mathbb{N}$.

Esempio

$f = f(n) = n + 1$ i.e.
$f(0) = 1$, $f(1) = 2$, $f(2) = 3$, ...
Possiamo immaginarla come una tabella infinita:

n	0	1	2	3	...
$f(n)$	1	2	3	4	...

Algoritmi

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.
Il termine è sulla bocca di tutti: solo un mese fa si parlava dell'algoritmo sbagliato per il calcolo delle tariffe degli abbonamenti del treno.

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.
Il termine è sulla bocca di tutti: solo un mese fa si parlava dell'algoritmo sbagliato per il calcolo delle tariffe degli abbonamenti del treno.

Intuitivamente sapete indicarmi degli esempi e descrivermi il loro funzionamento?

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.
Il termine è sulla bocca di tutti: solo un mese fa si parlava dell'algoritmo sbagliato per il calcolo delle tariffe degli abbonamenti del treno.

Intuitivamente sapete indicarmi degli esempi e descrivermi il loro funzionamento?

Possiamo dire informalmente che un **algoritmo** è una procedura che dato un **input** restituisce un **output**, utilizzando un **numero finito** di **regole**.

Che cos'è un algoritmo?

Dare una definizione precisa e rigorosa è difficile (ma si può, anche in diversi termini) e ci ritorniamo dopo.
Il termine è sulla bocca di tutti: solo un mese fa si parlava dell'algoritmo sbagliato per il calcolo delle tariffe degli abbonamenti del treno.

Intuitivamente sapete indicarmi degli esempi e descrivermi il loro funzionamento?

Possiamo dire informalmente che un **algoritmo** è una procedura che dato un **input** restituisce un **output**, utilizzando un **numero finito** di **regole**.
Considerando che possiamo **codificare** sotto forma di **numeri naturali** sia input sia output gli **algoritmi** sono un sottoinsieme delle **funzioni** da $\mathbb{N}$ ad $\mathbb{N}$.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.
Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.
Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

Esempi

- Ricerca binaria.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.
Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

Esempi

- Ricerca binaria.
- La somma in colonna.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.
Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

Esempi

- Ricerca binaria.
- La somma in colonna.
- Un algoritmo di ordinamento.

La complessità degli algoritmi

In genere si è interessati a trovari **algoritmi efficienti**, veloci per risolvere i problemi.
La misura di efficienza di un algoritmo si chiama **complessità computazionale**.
Normalmente viene espressa come una funzione della **dimensione** n dell'**input**. Per esempio $C = n$, $C = n^2$, $C = 2^n$. (Trascuro i dettagli della notazione O-grande).

Esempi

- Ricerca binaria.
- La somma in colonna.
- Un algoritmo di ordinamento.
- Il problema del commesso viaggiatore.

Classi di complessità **P** e **NP**

Definizione

La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Classi di complessità **P** e **NP**

Definizione

La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Definizione

La classe di problemi **NP** è composta da tutti i problemi la cui soluzione è verificabile in tempo polinomiale (problemi intrattabili).

Classi di complessità **P** e **NP**

Definizione

La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Definizione

La classe di problemi **NP** è composta da tutti i problemi la cui soluzione è verificabile in tempo polinomiale (problemi intrattabili).

Teorema

$\mathbf{P} \subseteq \mathbf{NP}$.

Classi di complessità **P** e **NP**

Definizione

La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Definizione

La classe di problemi **NP** è composta da tutti i problemi la cui soluzione è verificabile in tempo polinomiale (problemi intrattabili).

Teorema

$\mathbf{P} \subseteq \mathbf{NP}$.

Ci chiediamo se l'inclusione sia stretta, ovvero se ci siano dei problemi che stanno in **NP** ma non in **P**.

Classi di complessità **P** e **NP**

Definizione

La classe di problemi **P** è composta da tutti i problemi che sono risolvibili in tempo polinomiale (problemi trattabili).

Definizione

La classe di problemi **NP** è composta da tutti i problemi la cui soluzione è verificabile in tempo polinomiale (problemi intrattabili).

Teorema

$\mathbf{P} \subseteq \mathbf{NP}$.

Ci chiediamo se l'inclusione sia stretta, ovvero se ci siano dei problemi che stanno in **NP** ma non in **P**.

La risposta vale **1 milione** di dollari!

NP-completezza

Esistono alcuni problemi, chiamati **NP**-completi, che sono i più difficili della classe **NP**. Sono tutti della stessa difficoltà e le loro istanze possono essere trasformate in quelle di un altro problema **NP**-completo in tempo polinomiale.

NP-completezza

Esistono alcuni problemi, chiamati **NP**-completi, che sono i più difficili della classe **NP**. Sono tutti della stessa difficoltà e le loro istanze possono essere trasformate in quelle di un altro problema **NP**-completo in tempo polinomiale.

Sono problemi di importante interesse pratico:

NP-completezza

Esistono alcuni problemi, chiamati **NP**-completi, che sono i più difficili della classe **NP**. Sono tutti della stessa difficoltà e le loro istanze possono essere trasformate in quelle di un altro problema **NP**-completo in tempo polinomiale.

Sono problemi di importante interesse pratico:

Alcuni problemi **NP**-completi

- Il problema del commesso viaggiatore
- Il problema dello zaino
- La soddisfacibilità delle formule della logica proposizionale (SAT)
- Il Sudoku ($n^2 \times n^2$)

NP-completezza

Esistono alcuni problemi, chiamati **NP**-completi, che sono i più difficili della classe **NP**. Sono tutti della stessa difficoltà e le loro istanze possono essere trasformate in quelle di un altro problema **NP**-completo in tempo polinomiale.

Sono problemi di importante interesse pratico:

Alcuni problemi **NP**-completi

- Il problema del commesso viaggiatore
- Il problema dello zaino
- La soddisfacibilità delle formule della logica proposizionale (SAT)
- Il Sudoku ($n^2 \times n^2$)

Se trovassimo una soluzione polinomiale anche per uno solo dei seguenti problemi, vorrebbe dire che l'avremmo trovata per tutti e che $\mathbf{P} = \mathbf{NP}$.

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Per prima cosa dobbiamo capire che cosa vuol dire algoritmicamente. **Church** e **Turing** hanno definito nel 1936 due **modelli astratti** di **algoritmo**, rispettivamente il λ-**calcolo** e la **macchina di Turing**.

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Per prima cosa dobbiamo capire che cosa vuol dire algoritmicamente. **Church** e **Turing** hanno definito nel 1936 due **modelli astratti** di **algoritmo**, rispettivamente il λ-**calcolo** e la **macchina di Turing**. Noi ci accontentiamo di dire che un algoritmo è un **programma** scritto in un qualunque **linguaggio di programmazione** (C, Java, Python, etc) che termina.

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Per prima cosa dobbiamo capire che cosa vuol dire algoritmicamente. **Church** e **Turing** hanno definito nel 1936 due **modelli astratti** di **algoritmo**, rispettivamente il λ-**calcolo** e la **macchina di Turing**. Noi ci accontentiamo di dire che un algoritmo è un **programma** scritto in un qualunque **linguaggio di programmazione** (C, Java, Python, etc) che termina.

Possiamo farlo sulla base di un importante risultato: tutti i formalismi di potenza massima conosciuti sono **equivalenti**. In altre parole:

Tutti i problemi sono risolvibili algoritmicamente?

L'**informatica** moderna è nata per rispondere a questa domanda.

Per prima cosa dobbiamo capire che cosa vuol dire algoritmicamente. **Church** e **Turing** hanno definito nel 1936 due **modelli astratti** di **algoritmo**, rispettivamente il λ-**calcolo** e la **macchina di Turing**. Noi ci accontentiamo di dire che un algoritmo è un **programma** scritto in un qualunque **linguaggio di programmazione** (C, Java, Python, etc) che termina.

Possiamo farlo sulla base di un importante risultato: tutti i formalismi di potenza massima conosciuti sono **equivalenti**. In altre parole:

Tesi di Church-Turing

Una funzione intuitivamente calcolabile è calcolabile da una macchina di Turing (o formalismo equivalente).

Quanti sono gli algoritmi?

Se pensiamo all'insieme di tutti gli **algoritmi** come l'insieme di tutti i **programmi** Π capiamo subito che il loro numero è **infinito**. Però capiamo anche che li possiamo **enumerare** cioè considerare in ordine i programmi π_1,π_2,π_3...

Quanti sono gli algoritmi?

Se pensiamo all'insieme di tutti gli **algoritmi** come l'insieme di tutti i **programmi** Π capiamo subito che il loro numero è **infinito**. Però capiamo anche che li possiamo **enumerare** cioè considerare in ordine i programmi π_1,π_2,π_3...
Possiamo cioè metterli in corrispondenza con i **numeri naturali**, costruendo una funzione $f : \mathbb{N} \to \mathit{\Pi}$ tale che $f(y) = \pi_y$.

Quanti sono gli algoritmi?

Se pensiamo all'insieme di tutti gli **algoritmi** come l'insieme di tutti i **programmi** Π capiamo subito che il loro numero è **infinito**. Però capiamo anche che li possiamo **enumerare** cioè considerare in ordine i programmi π_1,π_2,π_3...
Possiamo cioè metterli in corrispondenza con i **numeri naturali**, costruendo una funzione $f : \mathbb{N} \rightarrow \mathit{\Pi}$ tale che $f(y) = \pi_y$.

L'enumerazione può essere fatta da un **algoritmo** e quindi siamo in grado di costruire un programma, chiamato **programma universale**, $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ che in ingresso prenda il numero (di Gödel) y del programma π_y da eseguire, e l'input x del programma stesso calcolando così $g(y, x) = \pi_y(x)$.

Quanti sono gli algoritmi?

Se pensiamo all'insieme di tutti gli **algoritmi** come l'insieme di tutti i **programmi** Π capiamo subito che il loro numero è **infinito**. Però capiamo anche che li possiamo **enumerare** cioè considerare in ordine i programmi π_1,π_2,π_3...
Possiamo cioè metterli in corrispondenza con i **numeri naturali**, costruendo una funzione $f : \mathbb{N} \rightarrow \mathit{\Pi}$ tale che $f(y) = \pi_y$.

L'enumerazione può essere fatta da un **algoritmo** e quindi siamo in grado di costruire un programma, chiamato **programma universale**, $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ che in ingresso prenda il numero (di Gödel) y del programma π_y da eseguire, e l'input x del programma stesso calcolando così $g(y, x) = \pi_y(x)$.
Questo procedimento si chiama **aritmetizzazione** dei programmi o gödelizzazione.

Un problema indecidibile

Un classico problema **indecidibile** è il **problema dell'arresto**, ovvero trovare un algoritmo che decida, dati in ingresso il numero di Gödel y di un programma π_y e il suo input x se questo terminerà o meno

Un problema indecidibile

Un classico problema **indecidibile** è il **problema dell'arresto**, ovvero trovare un algoritmo che decida, dati in ingresso il numero di Gödel y di un programma π_y e il suo input x se questo terminerà o meno , in formule:

$$h(y,x) = \begin{cases} 1 & \text{se } \pi_y(x) \text{ termina} \\ 0 & \text{altrimenti (cioè se } \pi_y(x) \text{ non termina)} \end{cases}$$

Un problema indecidibile

Un classico problema **indecidibile** è il **problema dell'arresto**, ovvero trovare un algoritmo che decida, dati in ingresso il numero di Gödel y di un programma π_y e il suo input x se questo terminerà o meno , in formule:

$$h(y,x) = \begin{cases} 1 & \text{se } \pi_y(x) \text{ termina} \\ 0 & \text{altrimenti (cioè se } \pi_y(x) \text{ non termina)} \end{cases}$$

Teorema (di Church)

La funzione h non è algoritmicamente calcolabile.

Riferimenti bibliografici

Scott Aaronson.
Quantum Computing Since Democritus.
Cambridge University Press, 2013.

Dino Mandrioli and Paola Spoletini.
Informatica teorica.
CittàStudi, Torino, 2011.

Giorgio Ausiello, Fabrizio D'Amore, Giorgio Gambosi, and Luigi Laura.
Linguaggi, modelli, complessità.
Franco Angeli, Milano, 2014.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman.
Automi, linguaggi e calcolabilità.
Pearson, Milano, 2009.

Thomas H. Cormen, Chrales E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduzione agli algoritmi e strutture dati.
McGraw-Hill Education, Milano, 2010.