

Algoritmi e computabilità

Autori

[Silvio Peroni – silvio.peroni@unibo.it](mailto:silvio.peroni@unibo.it)

Dipartimento di Filologia Classica e Italianistica, Università di Bologna, Bologna, Italia

[Aldo Gangemi – aldo.gangemi@unibo.it](mailto:aldo.gangemi@unibo.it)

Dipartimento di Filologia Classica e Italianistica, Università di Bologna, Bologna, Italia

[Matteo Pascoli - matteo.pascoli2@unibo.it](mailto:matteo.pascoli2@unibo.it)

Dipartimento di Filologia Classica e Italianistica, Università di Bologna, Bologna, Italia

Avviso sul copyright

Questo lavoro è rilasciato con licenza [Creative Commons Attribution 4.0 International License](#).

Sei libero di condividere (riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato) e modificare (remixare, trasformare il materiale e basarti su di esso per le tue opere per qualsiasi fine, anche commerciale) questo lavoro alle seguenti condizioni: attribuzione, ovvero devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale. Il licenziante non può revocare questi diritti fintanto che tu rispetti i termini della licenza.

Sommario

In questo capitolo verrà fornita una definizione generale di *algoritmo*, per poi introdurre un particolare linguaggio grafico, ovvero i diagrammi di flusso, per creare i nostri primi algoritmi. Infine, verrà discussa la tematica relativa alla computabilità algoritmica, mostrando come non tutti i problemi computazionali possono essere risolti mediante un mero processo algoritmico.

Usi possibili della Macchina Analitica

[Ada Lovelace](#) ([Figura 1](#)) era la figlia del poeta [Lord Byron](#). Matematica di formazione, è diventata famosa per il suo lavoro sulla [Macchina Analitica](#) di Babbage. La madre, Anne Isabella Milbanke, in contrasto con le abitudini del padre, che mal sopportava il suo interesse per la cultura scientifica, aveva invece da sempre supportato l'interesse che Ada aveva nella logica e nella matematica. Uno degli obiettivi della madre, infatti, era quello di evitare che la figlia incorresse nella stessa sregolatezza emotionale ed esistenziale che aveva caratterizzato la vita del padre. Tuttavia, in qualche modo, la creatività insita nella famiglia venne poi manifestata in modi assolutamente imprevedibili.

Nel 1833, Ada partecipò ad una festa organizzata da Charles Babbage per presentare la Macchina Differenziale. Fu talmente colpita dall'invenzione di Babbage che iniziò una corrispondenza epistolare con lui che durò 27 anni [Morais, 2013]. Ada fu la traduttrice in inglese del primissimo articolo sulla Macchina Analitica, scritto da [Luigi Federico Menabrea](#), e che lei stessa arricchì con un grande numero di annotazioni personali e riflessioni. Tra queste, c'era anche una descrizione di come usare la Macchina Analitica per calcolare i [numeri di Bernoulli](#) [Menabrea, 1842]. Tecnicamente, questo fu il primo programma – nonché il primo *algoritmo* – per un computer digitale mai scritto, e fu creato da Ada senza avere neppure a disposizione la macchina reale, visto che la Macchina Analitica era soltanto una macchina teorica che Babbage non costruì mai.



Figura 1. Ritratto di Ada Lovelace.

Sorgente: https://en.wikipedia.org/wiki/File:Ada_Lovelace_portrait.jpg.

Tuttavia, la sua visione sui possibili usi della Macchina Analitica andava anche oltre [Morais, 2013]:

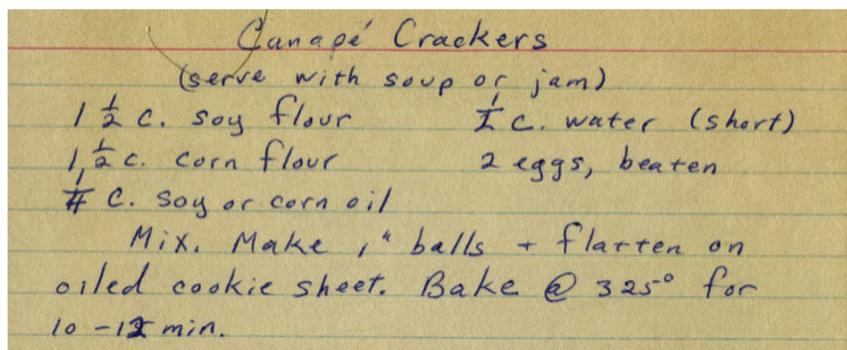
The operating mechanism can even be thrown into action independently of any object to operate upon (although of course no *result* could then be developed). Again, it might act upon other things besides *number*, were objects found whose mutual fundamental

relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine. Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.¹

Quel “science of operations” indicato nel testo è un riferimento ad uno specifico campo scientifico che fu chiaramente identificato soltanto molti anni dopo. In pratica, Ada Lovelace stava parlando dell’Informatica ben cent’anni prima della sua introduzione formale. Per il suo lavoro nel campo, Ada Lovelace è spesso riconosciuta come **la prima programmatrice della storia**.

Cos’è un algoritmo?

Gli algoritmi accompagnano sistematicamente le nostre attività della vita quotidiana. Per esempio, in [Figura 2](#) sono mostrati due esempi di procedure passo passo che dobbiamo seguire, rispettivamente, per preparare salatini e per assemblare una specifica lampada. Mentre l’obiettivo dei due esempi è estremamente diverso, in quanto nel primo è una ricetta e nel secondo è un insieme di istruzioni per assemblare un utensile, essi sono descritti nei termini della stessa nozione astratta: istruzione per *produrre qualcosa* partendo da un qualche *materiale iniziale* a disposizione – che, di fatto, rispecchia pienamente la definizione di *algoritmo*.



¹ Traduzione libera in italiano: “Il meccanismo può anche essere messo al lavoro indipendentemente dalla presenza effettiva di oggetti su cui operare (benché ovviamente in questo caso non arrivi necessariamente a un *risultato*). Inoltre, potrebbe operare su altre cose oltre ai *numeri*, se si trovassero oggetti le cui relazioni fondamentali possano essere espresse da quelle della scienza astratta delle operazioni, e che dovrebbero essere adattate all’azione della notazione delle operazioni e ai meccanismi della macchina. Supponendo, per esempio, che le relazioni fondamentali tra i suoni di varia altezza, nella scienza dell’armonia e della composizione musicale, siano suscettibili di tali espressioni e adattamenti, la macchina potrebbe comporre scientificamente brani di musica elaborati di qualunque durata o grado di complessità.”

Figura 2. Due fotografie che descrivono una ricetta (sinistra) e le istruzioni per assemblare una lampada (destra).

Foto di sinistra di Phil! Gold, sorgente: https://www.flickr.com/photos/phil_g/17282816/. Foto di destra di Richard Eriksson, sorgente: <https://www.flickr.com/photos/sillygwailo/3183183727/>.

La parola algoritmo è una combinazione della parola latina *algorismus* (che, a sua volta, è la latinizzazione del nome Al-Khwarizmi, che era un grande matematico persiano dell'ottavo secolo) e della parola greca *arithmos*, che significa *numero*. A livello generale, possiamo definire un algoritmo come l'astrazione di una procedura passo passo che prende qualcosa come *input* e produce un certo *output* [Wing, 2008]. Ogni algoritmo è scritto in un linguaggio specifico in modo che le istruzioni che definisce possano essere comunicate e comprese da un computer (sia esso umano o macchina) in modo da ottenere qualcosa come conseguenza dell'elaborazione di qualche materiale di input.

Un programmatore è una persona che crea algoritmi e li specifica in programmi usando uno specifico linguaggio comprensibile dal computer – ove, in questo caso, il termine *computer* si riferisce ai computer elettronici. Tuttavia, se ci si astrae dalla nozione di programma, un programmatore è chiunque sia in grado di creare algoritmi che possono essere interpretati da un qualunque computer (sia esso umano o macchina).

Diagrammi di flusso

Non esiste un linguaggio *standard* per descrivere un algoritmo in modo che possa essere immediatamente comprensibile da un qualunque computer. Tuttavia, spesso gli informatici si basano su uno *pseudocodice* quando vogliono descrivere un particolare algoritmo. Uno pseudocodice è un linguaggio informale che è solitamente usato per comunicare i passi principali di un algoritmo ad un umano. Mentre un algoritmo descritto mediante l'uso di pseudocodice non è eseguibile da un computer elettronico, i suoi costrutti sono strettamente connessi con quelli tipicamente definiti nei linguaggi di programmazione.

In particolare, ogni algoritmo può essere espresso in pseudocodice e, in principio, questo può essere a sua volta tradotto, abbastanza facilmente, in diversi linguaggi di programmazione. La vera differenza è che, di solito, alcuni passaggi dello pseudocodice possono essere semplificati usando delle espressioni in linguaggio naturale, mentre, in un linguaggio di programmazione, bisogna necessariamente definire in modo chiaro tutti i passaggi.

In questo capitolo, useremo una particolare alternativa grafica al comune pseudocodice che è facilmente comprensibile dagli umani: i diagrammi di flusso. Un diagramma di flusso è uno specifico tipo di diagramma che può essere usato per scrivere algoritmi, e che si basa su un limitato insieme di oggetti grafici, introdotti in Tabella 1.

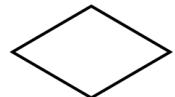
Oggetto grafico	Nome	Definizione
	Linea di flusso	La freccia è usata per definire l'ordine in cui le operazioni sono eseguite. Il flusso indicato dalla freccia inizia in un terminale di partenza e finisce in un terminale di fine (vedi l'oggetto successivo).
	Terminale	Viene usato per indicare l'inizio e la fine di un algoritmo. Contiene un testo (solitamente o "inizio" o "fine", in italiano) in modo da disambiguare qual è il ruolo del particolare oggetto terminale nel contesto dell'algoritmo.
	Processo	Viene usato per esprimere un'istruzione che è eseguita e che può cambiare lo stato corrente di qualche variabile usata nell'algoritmo. Il testo che contiene descrive l'istruzione da eseguire.
	Decisionale	Permette di esprimere operazioni condizionali, dove una condizione è verificata e, a seconda del valore di alcune variabili usate nell'algoritmo, l'esecuzione continua in un particolare ramo del flusso invece che in un altro. Di solito, questa operazione crea due possibili rami: uno seguito se la condizione è vera, e un altro che viene seguito quando la condizione è falsa.
	Input / Output	Permette di specificare un possibile input o output che viene usato o restituito dall'algoritmo solitamente all'inizio o alla fine della sua esecuzione.

Tabella 1. Gli oggetti grafici principali che possono essere usati in un diagramma di flusso, e che sono utili per scrivere un algoritmo.

Il nostro primo algoritmo

In questo capitolo svilupperemo il nostro primo algoritmo, che può essere descritto informalmente come segue: prendere in input tre stringhe, ovvero due parole e un riferimento bibliografico di un articolo pubblicato, e restituire 2 se entrambe le parole sono contenute nel riferimento bibliografico, 1 se solo una delle parole è contenuta nel riferimento bibliografico, o 0 altrimenti.

Una versione incompleta

Mediante i diagrammi di flusso, un qualunque algoritmo è definito usando due oggetti terminali, che identificano l'inizio e la fine dell'algoritmo. Il terminale di inizio ha una freccia che parte da esso e va verso l'istruzione successiva (la prima dell'algoritmo), mentre il terminale di fine può essere raggiunto da differenti punti dell'algoritmo, e quindi è collegato da almeno una freccia.

La prima versione incompleta dell'algoritmo, mostrata di seguito, semplifica un poco le istruzioni in linguaggio naturale precedentemente introdotte, in modo da mostrare come possiamo usare alcuni iniziali oggetti per creare un algoritmo, senza aggiungere ulteriore complessità, almeno per il momento. In particolare, la versione semplificata prende in input solo due stringhe, una parola e un riferimento bibliografico, e restituisce 1 se la parola è contenuta nel riferimento bibliografico, 0 altrimenti. Questa versione parziale è mostrata nel diagramma di flusso in [Figura 3](#).

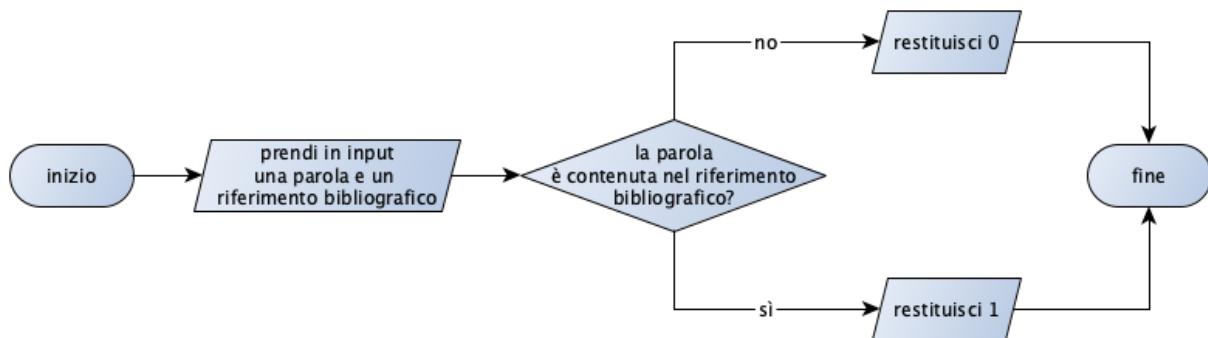


Figura 3. L'algoritmo incompleto descritto da un semplice diagramma di flusso.

Questa versione parziale usa già molti degli oggetti grafici propri ai diagrammi di flusso. In particolare, oltre ai terminali di inizio e fine, abbiamo usato tre oggetti di input / output per acquisire i valori specificati come input e per restituire 0 o 1 dipendentemente da questo input. La decisione su quale output restituire è stata codificata grazie all'oggetto decisionale dei diagrammi di flusso, in cui l'input è analizzato e, a seconda della situazione, uno specifico ramo del flusso dell'algoritmo viene percorso.

L'algoritmo completo

Mentre nella sezione precedente è stata introdotta una prima implementazione della versione parziale dell'algoritmo, l'implementazione dell'algoritmo completo attraverso lo sviluppo di un diagramma di flusso è mostrata in [Figura 4](#). In questo caso, sono stati utilizzati tutti gli oggetti grafici introdotti in [Tabella 1](#). Tuttavia, è importante sottolineare come il diagramma di flusso presentato è soltanto un possibile modo per implementare l'algoritmo originale. Infatti, è

possibile creare anche un diagramma di flusso diverso che, però, risolve il problema descritto dall'algoritmo in linguaggio naturale correttamente.

Nel diagramma in [Figura 4](#), viene utilizzato il primo oggetto di processo in cui viene inizializzato a 0, associandolo implicitamente a una variabile (ovvero, “result value” in figura), il risultato che verrà restituito alla fine dell'esecuzione dell'algoritmo. Questo risultato è quello che l'algoritmo deve ritornare se entrambe le parole in input non sono contenute nel riferimento bibliografico specificato. Questo oggetto di processo è seguito da due oggetti decisionali messi in sequenza, che controllano le due condizioni – ovvero se la prima parola è contenuta nel riferimento bibliografico, e se la seconda parola è contenuta nello stesso riferimento – e, nel caso queste siano vere, eseguono un incremento di 1 al risultato finale da restituire, mediante l'uso di altri oggetti di processo. Qualunque sia il valore che gli è stato associato, il risultato finale viene restituito da un unico oggetto di output, che conclude l'esecuzione dell'algoritmo.

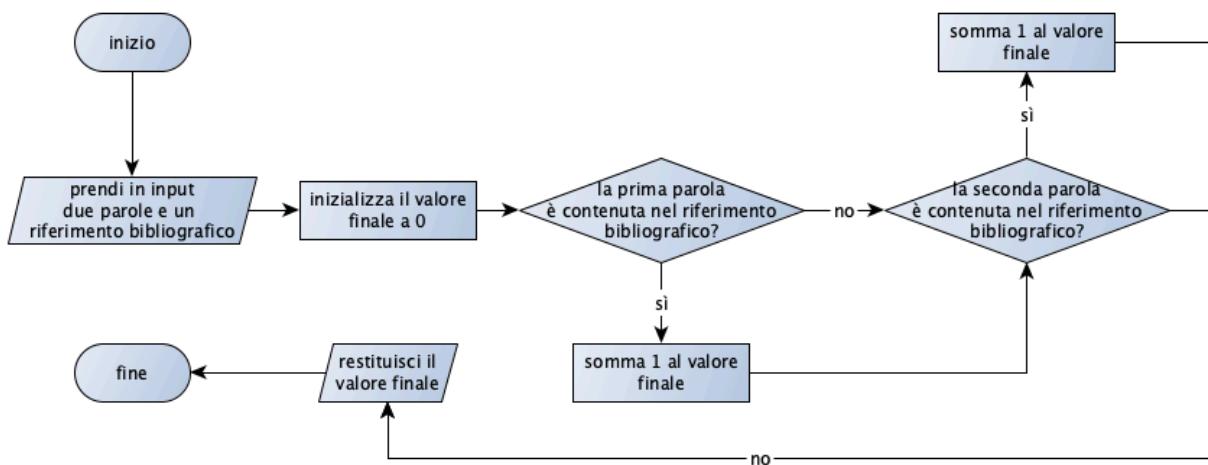


Figura 4. Il diagramma di flusso che implementa l'algoritmo completo.

Ci sono limiti alla computazione?

Una delle domande tradizionali che le persone che si avvicinano al pensiero computazionale e agli algoritmi di solito si pongono è: possiamo usare gli algoritmi per computare qualsiasi cosa vogliamo? In altre parole: esiste un limite a quello che possiamo computare? O ancora: è possibile definire un [problema computazionale](#) – ovvero un problema che può essere risolto algoritmicamente da un computer – che non può essere risolto da nessun algoritmo?

Nel caso dell'informatica, ma anche di tutte le scienze matematiche, uno degli approcci più usati per dimostrare che qualcosa non esiste è quello di costruire una situazione in apparenza plausibile che, poi, si rivela paradossale e auto-contraddittoria – in cui, per esempio, l'esistenza di un algoritmo contraddice se stessa. Questo approccio dimostrativo porta il nome di [reductio ad absurdum](#) (dimostrazione per assurdo). L'argomentazione che ne sta alla base è quella di

stabilire che una situazione è contraddittoria cercando di derivare un'assurdità dalla sua negazione, in modo da dimostrare che una tesi deve essere accettata perché la sua negazione non può essere difesa [Rescher, 2017] e, alla fine, genera un paradosso.

I paradossi sono stati usati molte volte in logica nel passato. Mentre, da un punto di vista, possono essere considerate storie divertenti da usare per insegnare, da un altro punto di vista sono strumenti potenti che mostrano i limiti di particolari aspetti formali di una situazione. Per esempio, uno dei più famosi paradossi in matematica è il [paradosso di Russell](#), scoperto da [Bertrand Russel](#) nel 1901. È stata una delle più grandi scoperte dell'inizio del ventesimo secolo, poiché ha provato che l'allora teoria degli insiemi proposta da [Georg Cantor](#), e usata come base fondazionale per tutto il lavoro che [Gottlob Frege](#) stava facendo per definire le leggi di base dell'aritmetica, portava ad una contraddizione. Di conseguenza, il paradosso di Russell invalidava l'intera teoria degli insiemi e il lavoro fatto da Frege stesso – che era in stampa quando Russell gli comunicò la sua scoperta. Una variazione del paradosso può essere formulata come segue.

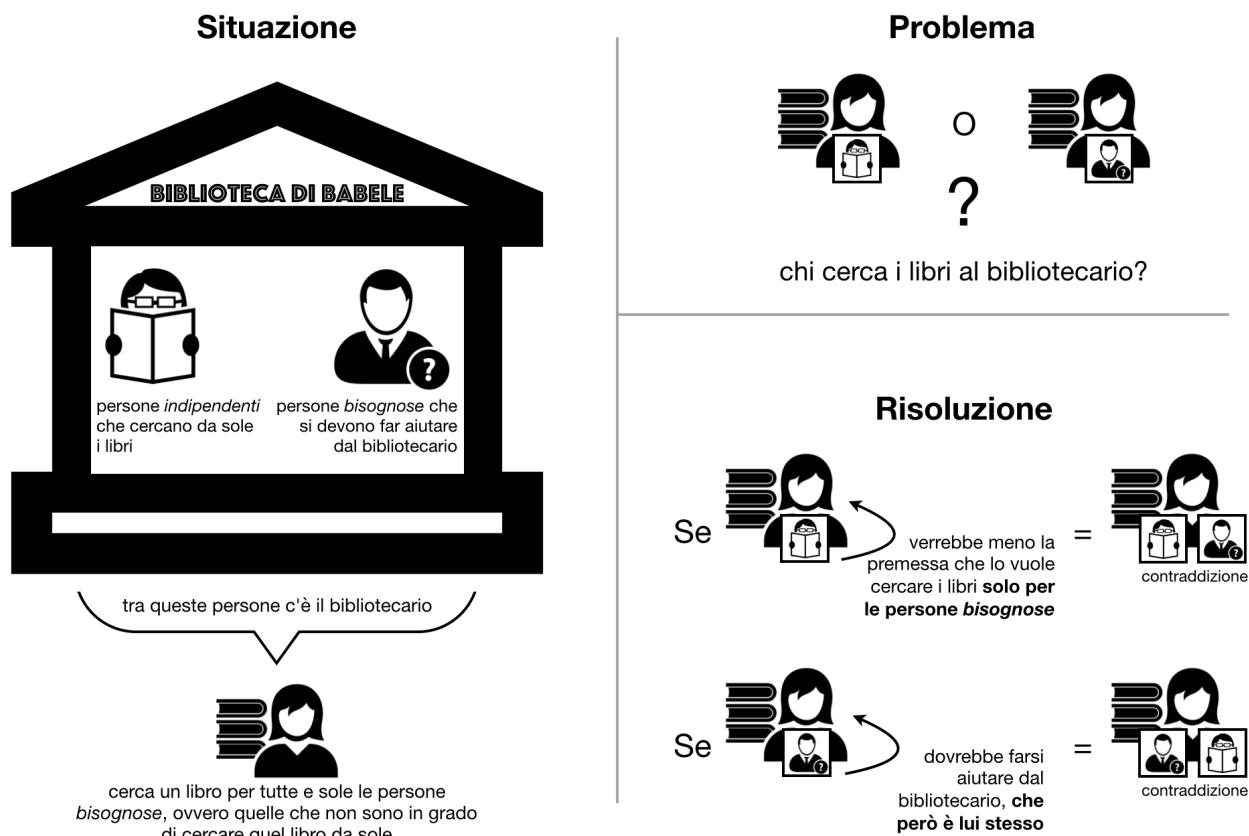


Figura 5. Una rappresentazione grafica del paradosso del bibliotecario, direttamente derivato dal paradosso di Russell.

Paradosso del bibliotecario: Nella Biblioteca di Babele, tutte le persone sono di una delle due seguenti tipologie. La prima tipologia di persone, le *indipendenti* ([Figura 5](#)), sono quelle che,

quando devono cercare un libro per loro stesse, lo fanno da sole senza chiedere aiuto a nessuno. L'altro tipo di persone, ovvero le *bisognose*, sono quelle che non sono in grado di cercare un libro da sole, e che necessitano dell'aiuto del bibliotecario per trovarlo. Ora, una delle persone nella biblioteca è il *bibliotecario*, che cerca un libro per tutte e sole le persone *bisognose*, ovvero quelle che non sono in grado di cercare quel libro da sole. Considerando questa situazione, la domanda è: chi cerca i libri al bibliotecario?

Risoluzione: Se il bibliotecario fosse *indipendente*, ovvero si cercasse i libri da solo, verrebbe meno la premessa che lo vuole cercare i libri solo per le persone *bisognose* – quindi, se fosse *indipendente* allora sarebbe *bisognoso*. Se invece non fosse in grado di cercarsi i libri da solo, e quindi fosse *bisognoso*, allora dovrebbe farsi aiutare dal bibliotecario, che però è lui stesso – quindi, se fosse *bisognoso* allora sarebbe *indipendente*.

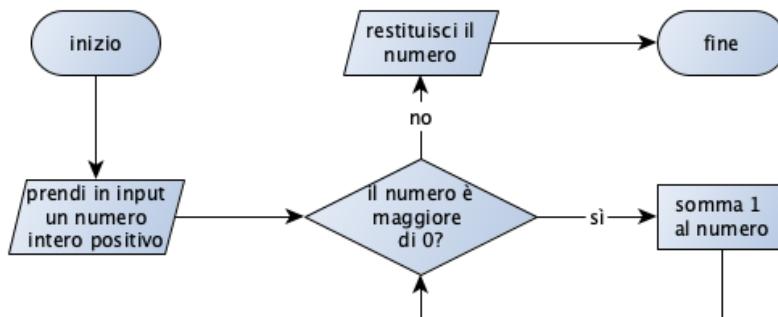


Figura 6. Un diagramma di flusso che descrive un algoritmo che non termina mai, visto che la condizione “il numero è maggiore di 0” è sempre vera per costruzione.

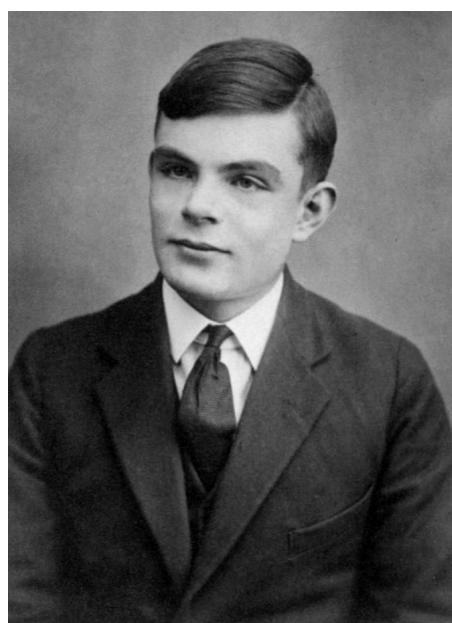


Figura 7. Una foto di Alan Turing fatta nel 1927.

Sorgente: https://en.wikipedia.org/wiki/File:Alan_Turing_Aged_16.jpg.

Uno dei più importanti problemi studiati in informatica anni fa, che faceva parte dei [23 problemi aperti della matematica](#) che [David Hilbert](#) propose nel 1900, è noto col nome di [problema della terminazione](#). Questo problema riguardava il capire se fosse possibile sviluppare un algoritmo che fosse in grado di rispondere se un altro algoritmo, specificato come input, terminasse la sua esecuzione o no. L'algoritmo proposto in [Figura 4](#) è un esempio di uno di quelli che termina, ma è possibile anche sviluppare un algoritmo che non termina mai, ad esempio come mostrato in quello definito in [Figura 6](#). Avere un modo per scoprire sistematicamente se un algoritmo termina la sua esecuzione o no sarebbe di importanza cruciale, perché permetterebbe immediatamente di identificare quegli algoritmi che non lavorano in modo appropriato.

Uno degli scienziati che di più ha lavorato alla risoluzione di questo quesito è stato [Alan Mathison Turing](#) (mostrato in [Figura 7](#)). Alan Turing è stato un informatico, nonché padre dell'[informatica teorica](#) e dell'[intelligenza artificiale](#), anche se i suoi lavori hanno interessato diverse discipline tra cui la matematica (si vedano gli studi per decodificare la [macchina Enigma](#)), la logica (con l'introduzione della [macchina di Turing \[Turing, 1937\]](#)), la filosofia (si veda lo studio sulla relazione tra i calcolatori elettronici e il concetto di intelligenza [\[Turing, 1950\]](#)) e la biologia (si veda lo studio che identifica i processi spontanei di creazione di pattern in natura [\[Turing, 1952\]](#)).

Nel 1936, Turing sviluppò la sua macchina proprio per cercare di rispondere al problema della terminazione di Hilbert. La macchina proposta da Turing era prettamente teorica, nel senso che non l'aveva costruita fisicamente, anche se recentemente molte persone hanno provato a costruire prototipi fisici dell'idea di Turing, come quello mostrato in [Figura 8](#).

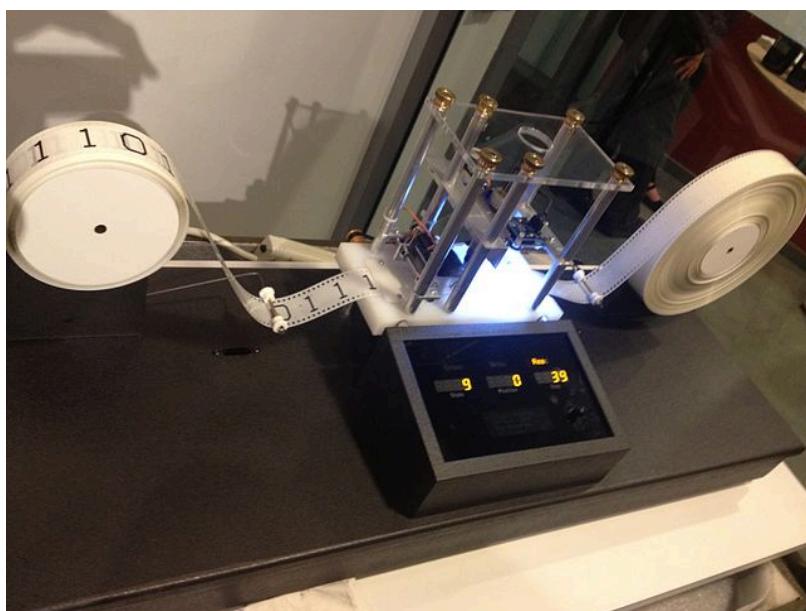


Figura 8. Un'implementazione fisica della macchina di Turing, con un nastro finito.
Foto di GabrielF, sorgente: https://commons.wikimedia.org/wiki/File:Model_of_a_Turing_machine.jpg.

La macchina, che è in grado di **simulare l'esecuzione di qualunque algoritmo realmente implementabile**, è composta da un *nastro* di memoria infinito composto da *celle*. Ogni cella può contenere un *simbolo* (o 0 o 1, dove 0 è usato come default per inizializzare le celle di tutto il nastro) che può essere letto e scritto dalla *testina* della macchina. Lo stato in cui la macchina si trova in un certo momento è altresì annotato. Le operazioni che può fare la macchina in un certo stato sono definite in una tabella (finita) di istruzioni, dove ogni istruzione dice cosa fare (scrivere un nuovo simbolo, muovere la testina a sinistra o a destra, spostarsi in un nuovo stato) in base allo stato in cui la macchina si trova e al simbolo presente nella cella sotto la testina. Infine, sono forniti anche uno stato iniziale e zero o più stati finali, in modo da sapere dove iniziare e finire il processo.

La macchina è stata usata da Turing per mostrare una soluzione per il problema della terminazione. In questo capitolo presentiamo un'approssimazione alla soluzione che Turing ha fornito, basata interamente sulla *reductio ad absurdum*, già usata per risolvere il paradosso del bibliotecario, e sui diagrammi di flusso come un'astrazione grafica di una specifica macchina di Turing.

Supponiamo sia possibile sviluppare l'algoritmo “termina?”, che prende in input un certo algoritmo e restituisce “vero” nel caso in cui l'algoritmo specificato come input termina, mentre restituisce “falso” in caso contrario. Ovviamente, questo è soltanto un algoritmo ipotetico: stiamo supponendo che possiamo svilupparlo in qualche modo, senza mostrare come farlo davvero.

Ora, usiamo l'algoritmo “termina?” per sviluppare un nuovo algoritmo, introdotto nel diagramma di flusso in [Figura 9](#). Questo nuovo algoritmo prende in input un algoritmo e restituisce 0 se l'algoritmo in input non termina, mentre non termina in caso contrario. Notate che siamo in grado di implementare davvero ogni passo di questo nuovo algoritmo, siccome scoprire se l'algoritmo in input termina è restituito dal nostro algoritmo (ipotetico) “termina?”, mentre la non terminazione è un processo chiaramente implementabile, visto che ne abbiamo introdotto un esempio in [Figura 6](#).

Ora la domanda è: cosa succede se cerchiamo di eseguire l'algoritmo descritto in [Figura 9](#) usando se stesso come input? Abbiamo due situazioni possibili:

- l'algoritmo “termina?” afferma che il nostro algoritmo in [Figura 9](#) **termina**, e conseguentemente (per come è definito) **non termina** l'esecuzione;
- l'algoritmo “termina?” afferma che il nostro algoritmo in [Figura 9](#) **non termina**, e conseguentemente (per come è definito) restituisce 0 e **termina** l'esecuzione.

Quindi, qualunque sia il comportamento del nostro algoritmo in [Figura 9](#), la sua esecuzione passando se stesso come input genera sempre una contraddizione. L'unica spiegazione possibile, quindi, è che l'algoritmo ipotetico “termina?” che usiamo per decidere se un algoritmo

termina o meno non può essere sviluppato. Di conseguenza, la risposta al problema della terminazione è che **l'algoritmo che verifica se un altro termina non può esistere**.

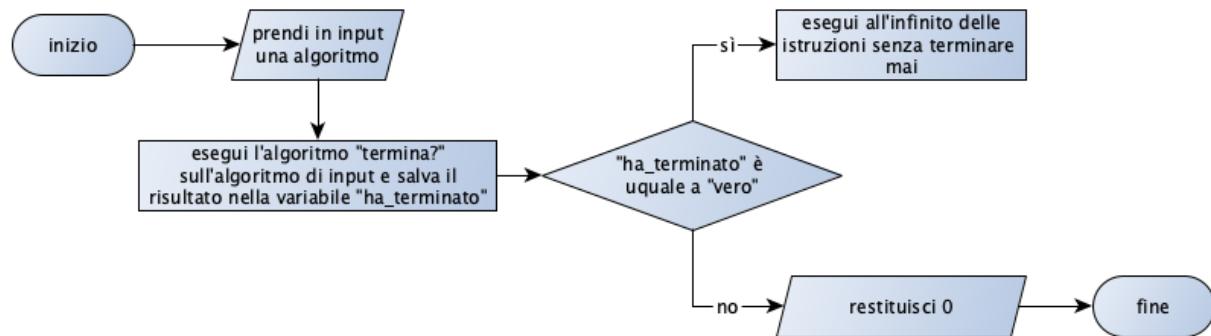


Figura 9. Il diagramma di flusso di un algoritmo che non termina se l'algoritmo specificato in input termina (verificato attraverso l'uso di dell'algoritmo ipotetico “termina?”), e restituisce 0 in caso contrario.

Questo risultato ha avuto un effetto dirompente sulla percezione delle abilità computazionali che un computer può avere. In pratica, la macchina di Turing e le relative analisi effettuate su di essa hanno imposto dei limiti chiarissimi a quello che possiamo calcolare, e hanno permesso di dimostrare che determinati problemi computazionali interessanti, come quello della terminazione, non possono essere risolti da nessun approccio algoritmico.

Ringraziamenti

Gli autori ringraziano Eugenia Galli per le preziose correzioni proposte al testo.

Bibliografia

Menabrea, L. F. (1842). Sketch of the Analytical Engine Invented by Charles Babbage – With notes upon the Memoir by the Translator: Ada Augusta, Countess of Lovelace. *Scientific Memoirs*, 3. <http://www.fourmilab.ch/babbage/sketch.html>

Morais, B. (2013). Ada Lovelace, the First Tech Visionary. *The New Yorker*. <https://www.newyorker.com/tech/elements/ada-lovelace-the-first-tech-visionary> (last visited 2 November 2017)

Rescher, N. (2017). Reductio ad Absurdum. *Internet Encyclopedia of Philosophy*. <http://www.iep.utm.edu/reductio/> (last visited 7 November 2017)

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 2 (42): 230-265. DOI: <https://doi.org/10.1112/plms/s2-42.1.230>

Turing, A. M. (1950). Computing Machinery and Intelligence. Mind, LIX (236): 433-460. DOI: <https://doi.org/10.1093/mind/LIX.236.433>

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366 (1881): 3717. <https://doi.org/10.1098/rsta.2008.0118>