Lezione di Informatica Teorica: La Macchina di Turing e i Fondamenti della Computabilità

Introduzione e Contesto Storico

L'informatica teorica rappresenta il fondamento matematico su cui poggia l'intera disciplina informatica moderna. Prima dell'avvento dei calcolatori elettronici, matematici e logici del ventesimo secolo si interrogavano sulla natura stessa del calcolo e sui limiti intrinseci di ciò che può essere computato mediante procedure algoritmiche.

Il problema centrale che animava il dibattito matematico degli anni Trenta riguardava la decidibilità: è possibile determinare meccanicamente se una proposizione matematica sia vera o falsa? Questo interrogativo, noto come *Entscheidungsproblem* (problema della decisione), formulato da David Hilbert nel 1928, costituiva uno dei pilastri del programma di Hilbert per la fondazione della matematica.

Alan Turing, matematico britannico, affrontò questo problema nel suo celebre articolo del 1936 "On Computable Numbers, with an Application to the Entscheidungsproblem", introducendo un modello astratto di calcolo che oggi conosciamo come Macchina di Turing. Parallelamente, Alonzo Church sviluppava il lambda calcolo, dimostrando l'equivalenza dei due approcci attraverso quella che oggi chiamiamo Tesi di Church-Turing.

La Macchina di Turing: Definizione Formale

Struttura e Componenti

Una Macchina di Turing (MdT) è un modello matematico di calcolo costituito da:

Definizione formale: Una Macchina di Turing deterministica è una settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ dove:

- **Q** è un insieme finito non vuoto di stati
- **Σ** è l'alfabeto di input (insieme finito di simboli, con $B \notin \Sigma$)
- **Γ** è l'alfabeto del nastro ($\Sigma \subset \Gamma$)
- **$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$** è la funzione di transizione
- **$q_0 \in Q$** è lo stato iniziale
- **$B \in \Gamma$** è il simbolo blank (vuoto)
- **$F \subseteq Q$** è l'insieme degli stati finali (o di accettazione)

Il Nastro e la Testina

Il nastro della Macchina di Turing è concettualmente infinito in entrambe le direzioni (o, equivalentemente, infinito a destra con estensione dinamica), suddiviso in celle contenenti ciascuna un simbolo dell'alfabeto Γ. La testina di lettura/scrittura può:

1. Leggere il simbolo nella cella corrente
2. Scrivere un nuovo simbolo nella cella corrente
3. Muoversi di una posizione a sinistra (L), a destra (R), o rimanere ferma (S)

Funzione di Transizione

La funzione di transizione δ rappresenta il "programma" della macchina. Dato lo stato corrente e il simbolo letto, δ determina:

- Il nuovo stato della macchina
- Il simbolo da scrivere nella cella corrente
- La direzione di movimento della testina

Formalmente: $\delta(q, a) = (p, b, D)$ significa che, trovandosi nello stato q e leggendo il simbolo a, la macchina transisce allo stato p, scrive il simbolo b e muove la testina nella direzione D.

Configurazione e Computazione

Una **configurazione** (o descrizione istantanea) della macchina descrive completamente il suo stato in un dato momento e può essere rappresentata come (q, w_1aw_2), dove:

- q è lo stato corrente
- a è il simbolo sotto la testina
- w_1 è la stringa a sinistra della testina
- w_2 è la stringa a destra della testina

Una **computazione** è una sequenza di configurazioni $C_0 \vdash C_1 \vdash C_2 \vdash \dots \vdash C_n$, dove C_0 è la configurazione iniziale e ogni C_{i+1} è ottenuta da C_i mediante l'applicazione della funzione di transizione.

La macchina **accetta** un input se, partendo dalla configurazione iniziale con quell'input sul nastro, raggiunge uno stato finale. La macchina **rigetta** se termina in uno stato non finale o se entra in un ciclo infinito.

Varianti della Macchina di Turing

Macchina di Turing a Nastro Multiplo

Una MdT a k nastri possiede k nastri indipendenti, ciascuno con la propria testina. La funzione di transizione diventa:

$\delta: Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

Teorema di equivalenza: Ogni Macchina di Turing a k nastri può essere simulata da una Macchina di Turing standard a un nastro con al più un rallentamento quadratico.

Dimostrazione (sketch): Si codificano i k nastri su un unico nastro usando un alfabeto espanso che include marcatori per le posizioni delle k testine. Ogni passo della macchina multinastro viene simulato mediante una scansione completa del nastro della macchina singolo-nastro.

Macchina di Turing Non Deterministica

Una Macchina di Turing non deterministica (NDTM) ha una funzione di transizione che restituisce un insieme di possibili transizioni:

$\delta: Q \times \Gamma \to P(Q \times \Gamma \times \{L, R, S\})$

La macchina accetta se esiste almeno un percorso computazionale che porta a uno stato finale.

Teorema: Ogni linguaggio riconosciuto da una NDTM è riconosciuto anche da una DTM.

Questo risultato è fondamentale ma nasconde una complessità cruciale: la simulazione può richiedere tempo esponenziale, portando alla famosa questione P vs NP.

Macchina di Turing Universale

Una Macchina di Turing Universale (UTM) è una MdT che può simulare qualsiasi altra MdT dato:

1. Una codifica della macchina da simulare
2. L'input per quella macchina

Formalmente, dato un sistema di codifica $\langle M \rangle$ per le macchine di Turing:

$U(\langle M \rangle, w)$ simula $M(w)$

L'esistenza di una UTM dimostra che esiste una macchina "programmabile" capace di eseguire qualsiasi algoritmo, prefigurando concettualmente il computer general-purpose.

Linguaggi Formali e Gerarchia di Chomsky

Definizioni Fondamentali

Un **linguaggio formale** L su un alfabeto Σ è un sottoinsieme (eventualmente infinito) di Σ^*, l'insieme di tutte le stringhe finite costruibili con simboli di Σ.

Gerarchia di Chomsky: I linguaggi formali si classificano in quattro categorie:

Tipo 0 - Linguaggi Ricorsivamente Enumerabili (RE): Riconosciuti dalle Macchine di Turing. Le produzioni grammaticali hanno forma $\alpha \to \beta$ con $\alpha, \beta \in (V \cup \Sigma)^*$ e α contiene almeno un simbolo non terminale.

Tipo 1 - Linguaggi Context-Sensitive (CS): Riconosciuti da automi linearly bounded. Le produzioni hanno forma $\alpha A \beta \to \alpha \gamma \beta$ con $|\gamma| \geq 1$ (produzioni non contrattive).

Tipo 2 - Linguaggi Context-Free (CF): Riconosciuti da automi a pila. Le produzioni hanno forma $A \to \gamma$ con A simbolo non terminale.

Tipo 3 - Linguaggi Regolari: Riconosciuti da automi a stati finiti. Le produzioni hanno forma $A \to aB$ o $A \to a$.

Vale l'inclusione stretta: $REG \subset CF \subset CS \subset RE$.

Relazione con le Macchine di Turing

Un linguaggio L è **Turing-riconoscibile** (o ricorsivamente enumerabile) se esiste una MdT M tale che:

- Per ogni $w \in L$, M accetta w
- Per ogni $w \notin L$, M rigetta w o non termina

Un linguaggio L è **Turing-decidibile** (o ricorsivo) se esiste una MdT M che termina sempre e:

- Per ogni $w \in L$, M accetta w
- Per ogni $w \notin L$, M rigetta w

La distinzione è cruciale: i linguaggi decidibili costituiscono un sottoinsieme proprio dei linguaggi riconoscibili.

Computabilità e Decidibilità

Funzioni Calcolabili

Una funzione f: Σ* → Σ* è **Turing-calcolabile** se esiste una MdT M tale che, per ogni input w:

- M termina con f(w) scritto sul nastro se f(w) è definita
- M non termina se f(w) è indefinita

La Tesi di Church-Turing

Tesi di Church-Turing: Ogni funzione calcolabile mediante una procedura algoritmica intuitiva è calcolabile da una Macchina di Turing.

Questa non è un teorema matematico ma una tesi filosofica che identifica la nozione intuitiva di "calcolabilità" con la calcolabilità formale delle MdT. L'evidenza a supporto include:

1. Equivalenza di tutti i modelli di calcolo proposti (lambda calcolo, funzioni ricorsive, macchine a registri, ecc.)
2. Assenza di controesempi convincenti in oltre 80 anni
3. Successo pratico nell'analisi di algoritmi

Problemi Indecidibili

Teorema dell'Arresto (Halting Problem): Il problema di determinare se una MdT M si ferma su un input w è indecidibile.

Dimostrazione per diagonalizzazione:

Supponiamo per assurdo che esista una MdT H che decide il problema dell'arresto:

H(⟨M⟩, w) = { accetta se M si ferma su w rigetta se M non si ferma su w }

Costruiamo una macchina D che, dato ⟨M⟩:

1. Simula H(⟨M⟩, ⟨M⟩)
2. Se H accetta, D entra in loop infinito
3. Se H rigetta, D si ferma e accetta

Consideriamo D(⟨D⟩):

- Se D si ferma su ⟨D⟩, allora H(⟨D⟩, ⟨D⟩) rigetta, quindi per costruzione D entra in loop (contraddizione)
- Se D non si ferma su ⟨D⟩, allora H(⟨D⟩, ⟨D⟩) accetta, quindi per costruzione D si ferma (contraddizione)

Entrambi i casi portano a contraddizione, dunque H non può esistere. ∎

Riduzioni e Completezza

Una **riduzione** da un problema A a un problema B è un algoritmo che trasforma istanze di A in istanze di B preservando la risposta. Se A è riducibile a B e B è decidibile, allora anche A è decidibile.

Esempi di problemi indecidibili dimostrati per riduzione dall'Halting Problem:

1. **Problema del nastro vuoto**: Determinare se M ferma con nastro vuoto
2. **Problema della stringa specifica**: Determinare se M scrive mai una stringa specifica s
3. **Problema dell'equivalenza**: Determinare se due MdT riconoscono lo stesso linguaggio
4. **Post Correspondence Problem (PCP)**: Data una collezione di coppie di stringhe, determinare se esiste una sequenza di coppie tale che la concatenazione delle prime componenti eguagli quella delle seconde

Teoria della Complessità Computazionale

Classi di Complessità Temporale

Definizione: Una MdT M opera in tempo f(n) se, per ogni input di lunghezza n, M termina entro f(n) passi.

DTIME(f(n)): Classe dei linguaggi decidibili da una DTM in tempo O(f(n))

NTIME(f(n)): Classe dei linguaggi decidibili da una NDTM in tempo O(f(n))

Classe P: Insieme dei problemi decidibili in tempo polinomiale $P = \cup_k \text{DTIME}(n^k)$

Classe NP: Insieme dei problemi decidibili da una NDTM in tempo polinomiale $NP = \cup_k \text{NTIME}(n^k)$

Equivalentemente, NP è la classe dei problemi per cui una soluzione può essere verificata in tempo polinomiale.

Il Problema P vs NP

Domanda aperta: P = NP?

Sappiamo che $P \subseteq NP$, ma non sappiamo se l'inclusione è stretta. Questo è considerato il più importante problema aperto in informatica teorica, con un premio di un milione di dollari del Clay Mathematics Institute.

Significato pratico: Se P = NP, problemi che oggi richiedono tempo esponenziale (come fattorizzazione, ottimizzazione combinatoria, scheduling) potrebbero essere risolti efficientemente. L'impatto su crittografia, intelligenza artificiale, logistica e molti altri campi sarebbe rivoluzionario.

NP-Completezza

Un problema L è **NP-completo** se:

1. $L \in NP$
2. Per ogni $L' \in NP$, L' è riducibile in tempo polinomiale a L (L è NP-hard)

Teorema di Cook-Levin (1971): Il problema SAT (soddisfacibilità booleana) è NP-completo.

Questo teorema è fondamentale perché fornisce il primo problema NP-completo, permettendo di dimostrare la NP-completezza di altri problemi mediante riduzione da SAT.

Esempi di problemi NP-completi:

- **3-SAT**: Soddisfacibilità con clausole di al più 3 letterali
- **Clique**: Trovare una clique di dimensione k in un grafo
- **Vertex Cover**: Coprire tutti gli archi con al più k vertici
- **Hamiltonian Path**: Determinare se esiste un cammino hamiltoniano
- **Traveling Salesman Problem (versione decisionale)**: Esistenza di tour di lunghezza ≤ k

Classi di Complessità Spaziale

SPACE(f(n)): Linguaggi decidibili usando O(f(n)) celle di nastro

NSPACE(f(n)): Linguaggi decidibili da NDTM usando O(f(n)) celle

Classe L: SPACE(log n) - problemi decidibili in spazio logaritmico

Classe PSPACE: $\cup_k$ SPACE(n^k) - problemi decidibili in spazio polinomiale

Teorema di Savitch: NSPACE(f(n)) ⊆ SPACE(f^2(n)) per f(n) ≥ log n

Questo implica che PSPACE = NPSPACE, un risultato notevole che contrasta con l'incertezza su P vs NP.

Gerarchia delle Classi di Complessità

Le relazioni note tra le principali classi sono:

L ⊆ NL ⊆ P ⊆ NP ⊆ PSPACE ⊆ EXPTIME

Sappiamo che P ≠ EXPTIME (per il teorema della gerarchia temporale), ma non sappiamo quali altre inclusioni siano strette.

Automi a Stati Finiti e Linguaggi Regolari

Automi a Stati Finiti Deterministici (DFA)

Un DFA è una quintupla M = (Q, Σ, δ, q_0, F) dove:

- Q è l'insieme finito degli stati
- Σ è l'alfabeto di input
- δ: Q × Σ → Q è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- F ⊆ Q è l'insieme degli stati finali

Teorema: Un linguaggio è regolare se e solo se è riconosciuto da un DFA.

Automi a Stati Finiti Non Deterministici (NFA)

Un NFA differisce da un DFA per la funzione di transizione:

δ: Q × (Σ ∪ {ε}) → P(Q)

Gli NFA possono avere transizioni ε (senza consumare input) e transizioni multiple dallo stesso stato con lo stesso simbolo.

Teorema di equivalenza DFA-NFA: Per ogni NFA esiste un DFA equivalente (costruzione per sottoinsiemi).

Espressioni Regolari

Le espressioni regolari forniscono una notazione algebrica per i linguaggi regolari:

- $\emptyset$ (insieme vuoto)
- ε (stringa vuota)
- a (per ogni $a \in \Sigma$)
- $R_1 \cup R_2$ (unione)
- $R_1 \cdot R_2$ (concatenazione)
- R^* (chiusura di Kleene)

Teorema di Kleene: Un linguaggio è regolare se e solo se può essere descritto da un'espressione regolare.

Pumping Lemma per Linguaggi Regolari

Lemma di Pumping: Se L è un linguaggio regolare, esiste una costante p (lunghezza di pumping) tale che ogni stringa $w \in L$ con $|w| \geq p$ può essere suddivisa in $w = xyz$ soddisfacendo:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Applicazione: Dimostrare che $L = \{a^nb^n \mid n \geq 0\}$ non è regolare.

Supponiamo per assurdo che L sia regolare con lunghezza di pumping p. Consideriamo $w = a^pb^p \in L$. Per il pumping lemma, $w = xyz$ con $y = a^k$ ($k > 0$) e $|xy| \leq p$. Allora $xy^2z = a^{p+k}b^p \notin L$, contraddizione. ∎

Automi a Pila e Linguaggi Context-Free

Automi a Pila (PDA)

Un PDA è una settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ dove:

- Q, Σ, q_0, F come nei DFA
- Γ è l'alfabeto di pila
- $Z_0 \in \Gamma$ è il simbolo iniziale di pila
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$ è la funzione di transizione

Teorema: Un linguaggio è context-free se e solo se è accettato da un PDA.

Grammatiche Context-Free

Una CFG è una quadrupla $G = (V, \Sigma, R, S)$ dove:

- V è l'insieme dei simboli non terminali
- Σ è l'alfabeto dei terminali
- $R \subseteq V \times (V \cup \Sigma)^*$ è l'insieme delle produzioni
- $S \in V$ è il simbolo iniziale

Forma Normale di Chomsky: Ogni CFG può essere convertita in forma normale di Chomsky dove tutte le produzioni hanno forma:

- A → BC (con B, C non terminali)
- A → a (con a terminale)
- S → ε (solo se ε ∈ L(G))

Pumping Lemma per Linguaggi Context-Free

Lemma di Pumping (CF): Se L è context-free, esiste p tale che ogni w ∈ L con |w| ≥ p può essere scritta come w = uvxyz con:

1. |vxy| ≤ p
2. |vy| > 0
3. Per ogni i ≥ 0, $uv^ixy^iz \in L$

Applicazione: L = $\{a^nb^nc^n \mid n \geq 0\}$ non è context-free (dimostrazione simile al caso regolare).

Modelli di Calcolo Alternativi

Lambda Calcolo

Il lambda calcolo, sviluppato da Church, è un sistema formale basato su:

- **Variabili**: x, y, z, ...
- **Astrazione**: λx.M (funzione con parametro x e corpo M)
- **Applicazione**: M N (applicazione della funzione M all'argomento N)

Regola β: (λx.M)N →β M[N/x] (sostituzione di x con N in M)

Teorema di Church-Rosser: Se un termine ha una forma normale, la β-riduzione la raggiunge indipendentemente dall'ordine delle riduzioni.

Funzioni Ricorsive

Le funzioni ricorsive sono definite a partire da:

1. **Funzioni base**: zero Z(x) = 0, successore S(x) = x+1, proiezioni P_i^n
2. **Composizione**: h = f ∘ (g_1, ..., g_m)
3. **Ricorsione primitiva**:
   - h(0, $\vec{x}$) = f($\vec{x}$)
   - h(S(y), $\vec{x}$) = g(y, h(y, $\vec{x}$), $\vec{x}$)
4. **Minimizzazione**: μy[f(y, $\vec{x}$) = 0] (il minimo y tale che f(y, $\vec{x}$) = 0)

Teorema: Una funzione è Turing-calcolabile se e solo se è ricorsiva.

Macchine a Registri

Le macchine a registri (o RAM machines) hanno:

- Registri R_0, R_1, R_2, ... contenenti numeri naturali
- Istruzioni: INC(i), DEC(i), JZ(i, l)

Teorema: Le macchine a registri sono equivalenti alle MdT in termini di potenza computazionale.

Logica e Computabilità

Sistemi Formali e Teoremi di Gödel

Un **sistema formale** consiste in:

- Un linguaggio formale con sintassi precisa
- Assiomi (formule assunte vere)
- Regole di inferenza per derivare teoremi

Primo Teorema di Incompletezza di Gödel (1931): In ogni sistema formale consistente e sufficientemente espressivo (capace di rappresentare l'aritmetica), esistono proposizioni vere ma non dimostrabili nel sistema.

Secondo Teorema di Incompletezza: Un sistema formale consistente non può dimostrare la propria consistenza.

Questi risultati hanno profonde implicazioni per i limiti della formalizzazione matematica e sono intimamente connessi alla teoria della computabilità.

Corrispondenza Curry-Howard

Esiste una profonda corrispondenza tra:

- Logica proposizionale ↔ Lambda calcolo tipato
- Formule ↔ Tipi
- Dimostrazioni ↔ Programmi
- Normalizzazione ↔ Esecuzione

Questa corrispondenza unifica logica, teoria dei tipi e computazione, costituendo la base della teoria dei tipi costruttivi e dei proof assistants moderni.

Applicazioni e Sviluppi Moderni

Complessità Parametrizzata

La teoria della complessità parametrizzata studia problemi NP-hard chiedendo: possiamo risolverli efficientemente quando un parametro k è piccolo?

Un problema è **fixed-parameter tractable (FPT)** se risolvibile in tempo $f(k)\cdot n^c$ dove f è una funzione qualsiasi di k e c è una costante.

Esempio: **Vertex Cover** è FPT nel parametro k (dimensione del cover).

Complessità Descrittiva

La complessità descrittiva collega classi di complessità e logica:

Teorema di Fagin: NP è esattamente la classe dei linguaggi descrivibili in logica del secondo ordine esistenziale.

Teorema: P coincide con i linguaggi descrivibili in logica del primo ordine con punto fisso meno generale.

Calcolo Quantistico

Le macchine di Turing quantistiche operano su sovrapposizioni di stati, utilizzando:

- Qubit invece di bit classici
- Operazioni unitarie invece di transizioni deterministiche
- Misurazioni che collassano lo stato

Classe BQP (Bounded-error Quantum Polynomial time): problemi risolvibili da computer quantistici in tempo polinomiale con probabilità di errore limitata.

Sappiamo: $P \subseteq BQP \subseteq PSPACE$, ma non sappiamo se $BQP = P$ o $BQP = NP$.

Calcolo DNA e Bio-Computing

Il calcolo con DNA sfrutta:

- Massivo parallelismo molecolare
- Complementarietà delle basi
- Operazioni biologiche (ibridazione, ligazione, PCR)

Adleman (1994) risolse un'istanza del Hamiltonian Path Problem con DNA, dimostrando la fattibilità del bio-computing.

Conclusioni e Prospettive Future

L'informatica teorica fornisce i fondamenti matematici per comprendere cosa può essere computato e con quale efficienza. I risultati principali che abbiamo esplorato includono:

1. **Universalità**: La Macchina di Turing Universale dimostra l'esistenza di macchine programmabili general-purpose
2. **Limiti fondamentali**: L'Halting Problem e altri risultati stabiliscono confini invalicabili alla computazione
3. **Separazione di complessità**: La gerarchia delle classi di complessità organizza i problemi per difficoltà computazionale
4. **Equivalenza dei modelli**: Lambda calcolo, funzioni ricorsive e MdT sono equivalenti, supportando la Tesi di Church-Turing

Le domande aperte più significative rimangono P vs NP, i limiti del calcolo quantistico rispetto al calcolo classico, e le connessioni profonde tra logica, computabilità e complessità.

La Macchina di Turing, concepita come strumento per affrontare questioni fondazionali della matematica, si è rivelata il modello teorico che meglio cattura l'essenza della computazione, influenzando profondamente lo sviluppo dell'informatica pratica e rimanendo ancora oggi il riferimento centrale per l'analisi degli algoritmi e lo studio dei limiti computazionali.