Lezione di Informatica Teorica: Il Teorema di Cook

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Introduzione e Richiami

La lezione odierna chiude la parte dedicata alla classe di complessità **NP**, concentrandosi sul Teorema di Cook, fondamentale per la comprensione della teoria della NP-completezza.

1.1 Richiami sulle Classi di Complessità

Definizione 1 (Classe NP)**.** *La classe **NP** (Nondeterministic Polynomial time) include tutti i problemi di decisione (linguaggi) che possono essere decisi da una* Macchina di Turing Non Deterministiche (NDTM) *in tempo polinomiale.*

Definizione 2 (NP-Hard)**.** *Un linguaggio L è **NP-Hard** se per ogni linguaggio L' appartenente a NP, L' si riduce polinomialmente a L ($L' \leq_P L$). Intuitivamente, i problemi NP-Hard sono almeno difficili quanto tutti i problemi in NP.*

Definizione 3 (NP-Complete)**.** *Un linguaggio L è **NP-Complete** se L è sia in NP che NP-Hard.*

1.2 Il Problema della Dimostrazione Iniziale

Fino ad ora, per dimostrare che un problema B è NP-Hard, abbiamo usato la *transitività delle riduzioni polinomiali*:

Teorema 1 (Proprietà di Transitività)**.** *Se A è NP-Hard e $A \leq_P B$, allora B è NP-Hard.*

Questo metodo ci ha permesso di dimostrare che molti problemi (e.g., Knapsack, Exact Cover, 3-SAT, Vertex Cover, Click) sono NP-Complete o NP-Hard, partendo da un problema già noto come NP-Hard. Tuttavia, questo approccio si basa sull'assunzione che esista almeno un problema NP-Hard conosciuto. Come abbiamo dimostrato che SAT è NP-Hard? In realtà, non l'abbiamo ancora fatto. Tutte le nostre dimostrazioni precedenti si fondano su questa assunzione.

Per dimostrare che **SAT** è NP-Hard, non possiamo usare la transitività, poiché non abbiamo un problema NP-Hard di partenza. Dobbiamo usare la definizione originale di NP-Hardness: mostrare che *ogni linguaggio L in NP si riduce polinomialmente a SAT*. Il problema è che ci sono infiniti linguaggi in NP, quindi non possiamo fornire una riduzione specifica per ognuno di essi. È necessario un approccio generico. Questo è il contenuto del **Teorema di Cook**.

2 Teorema di Cook: SAT è NP-Completo

Teorema 2 (Teorema di Cook)**.** *Il problema SAT (Boolean Satisfiability Problem) è NP-Complete.*

Per dimostrare che SAT è NP-Complete, dobbiamo mostrare due cose:

1. SAT è in NP. (Questo è già noto: una soluzione (assegnamento di verità) può essere verificata in tempo polinomiale).
2. SAT è NP-Hard. (Questo è l'obiettivo della lezione: mostrare che $L \leq_P$ SAT per ogni $L \in$ NP).

2.1 La Strategia della Riduzione

L'idea chiave della dimostrazione di NP-Hardness di SAT è costruire una riduzione polinomiale da un *generico* linguaggio $L \in$ NP a SAT. Poiché $L \in$ NP, per definizione, esiste una *Macchina di Turing Non Deterministica (NDTM) M* che decide L in tempo polinomiale. Sia $P(n)$ il polinomio che definisce il tempo di esecuzione di M per un input di lunghezza n.

La riduzione f prenderà in input una stringa W (istanza generica di L) e produrrà una *formula booleana* $F(W)$ in forma congiuntiva normale (CNF). La formula $F(W)$ sarà soddisfacibile se e solo se la macchina M accetta la stringa W.

$$W \in L \iff M \text{ accetta } W \iff F(W) \text{ è soddisfacibile}$$

La funzione di riduzione f (che costruisce $F(W)$) dovrà essere calcolabile in tempo polinomiale.

2.1.1 Assunzioni semplificative sulla Macchina di Turing M

Per facilitare la costruzione della formula $F(W)$, facciamo alcune assunzioni semplificative su M:

- Il nastro di M è **semi-infinito** (parte dalla cella 0, non va a sinistra di 0). Qualsiasi NDTM può essere convertita in una equivalente con nastro semi-infinito.
- M esegue **esattamente** $P(n)$ **passi** per ogni ramo di computazione. Se un ramo termina prima (accetta o rifiuta), M continua a ciclare in uno stato finale fittizio per riempire i restanti passi fino a $P(n)$. Questo semplifica la gestione del tempo.
- M non scrive mai il simbolo "blank" (si può assumere che utilizzi un simbolo speciale diverso dal blank per marcare celle vuote o usate).
- Il numero di stati R di M e il numero di simboli dell'alfabeto del nastro Γ sono costanti e indipendenti dalla lunghezza dell'input n.

Dato che M compie al più $P(n)$ passi, l'unica parte del nastro che può essere letta o scritta è quella compresa tra la cella 0 e la cella $P(n)$ (inclusa).

2.2 Variabili Proposizionali per la Simulazione

Per simulare il comportamento di M su W all'interno di una formula booleana, definiamo un insieme di variabili proposizionali. Queste variabili rappresenteranno lo stato di M in ogni istante di tempo e su ogni cella del nastro. Sia $N_{max} = P(n)$ il numero massimo di passi e celle rilevanti.

a) $Q_{i,k}$ (Stato): Variabile booleana vera se e solo se la macchina M al passo i si trova nello stato q_k.
   - $i \in [0, N_{max}]$ (passi di computazione)
   - $k \in [0, R-1]$ (indici degli stati, dove R è il numero totale di stati di M)
   - Numero di variabili: $O(N_{max} \cdot R) = O(P(n))$, che è polinomiale.

b) $H_{i,j}$ (Head - Testina): Variabile booleana vera se e solo se la testina di M al passo i si trova sulla cella j del nastro.
   - $i \in [0, N_{max}]$

- $j \in [0, N_{max}]$ (indici delle celle del nastro)
- Numero di variabili: $O(N_{max}^2) = O(P(n)^2)$, che è polinomiale.

c) $T_{i,j,l}$ (Tape - Nastro): Variabile booleana vera se e solo se al passo i la cella j del nastro contiene il simbolo α_l.

- $i \in [0, N_{max}]$
- $j \in [0, N_{max}]$
- $l \in [0, |\Gamma| - 1]$ (indici dei simboli dell'alfabeto del nastro Γ)
- Numero di variabili: $O(N_{max}^2 \cdot |\Gamma|) = O(P(n)^2)$, che è polinomiale.

Il numero totale di variabili proposizionali è polinomiale rispetto alla lunghezza dell'input n.

2.3 Struttura della Formula $F(W)$

La formula $F(W)$ sarà una congiunzione di quattro macro-clausole (o gruppi di clausole):

$$F(W) = C \land S \land N \land F$$

Dove:

- C (Consistency): Assicura che l'assegnamento di verità alle variabili descriva una configurazione valida della macchina a ogni passo.
- S (Start): Codifica la configurazione iniziale della macchina (stato, posizione testina, contenuto del nastro).
- N (Next Step): Codifica come la configurazione della macchina evolve da un passo all'altro, seguendo la funzione di transizione di M.
- F (Finish): Codifica che la macchina raggiunge uno stato di accettazione all'ultimo passo.

Prima di definire le macro-clausole, ripassiamo alcune proprietà utili per convertire le implicazioni in CNF:

- **Implicazione**: $A \implies B$ è equivalente a $\neg A \lor B$.
- **Distributività**: $A \land (B \lor C)$ è equivalente a $(A \land B) \lor (A \land C)$.
- **De Morgan**: $\neg(A \land B)$ è equivalente a $\neg A \lor \neg B$.
- **Implicazione annidata**: $A \implies (B \land C)$ è equivalente a $(A \implies B) \land (A \implies C)$.
- **Implicazione annidata (OR)**: $A \implies (B \lor C)$ è equivalente a $(\neg A \lor B \lor C)$.

Esempio 1 (Esempio di Riformulazione)**.** *Consideriamo la formula:* $A \land B \implies (C \land D) \lor (E \land F)$

1. *Applichiamo la distributività sulla parte destra dell'OR:* $A \land B \implies (C \lor E) \land (C \lor F) \land (D \lor E) \land (D \lor F)$
2. *Applichiamo l'implicazione annidata:* $X \implies Y \land Z$ *è* $(X \implies Y) \land (X \implies Z)$*. Questo si traduce in una congiunzione di 4 implicazioni:* $(A \land B \implies C \lor E) \land$
   $(A \land B \implies C \lor F) \land$
   $(A \land B \implies D \lor E) \land$
   $(A \land B \implies D \lor F)$

3. *Ogni implicazione* $X \implies Y$ *può essere riscritta come* $\neg X \vee Y$: $(\neg(A \wedge B) \vee C \vee E) \wedge$
   $(\neg(A \wedge B) \vee C \vee F) \wedge$
   $(\neg(A \wedge B) \vee D \vee E) \wedge$
   $(\neg(A \wedge B) \vee D \vee F)$
4. *Infine, applichiamo De Morgan:* $\neg(A \wedge B)$ *è* $\neg A \vee \neg B$: $(\neg A \vee \neg B \vee C \vee E) \wedge$
   $(\neg A \vee \neg B \vee C \vee F) \wedge$
   $(\neg A \vee \neg B \vee D \vee E) \wedge$
   $(\neg A \vee \neg B \vee D \vee F)$

Questa forma finale è una formula in CNF, composta da clausole di 4 letterali.

2.3.1 1. Consistenza (C)

La clausola C impone vincoli affinché l'assegnamento di verità rappresenti una configurazione coerente della macchina di Turing ad ogni passo.

- **Unicità dello stato per passo**: Ad ogni passo i, la macchina si trova in *al più un* stato.

$$\bigwedge_{i=0}^{N_{max}} \bigwedge_{k=0}^{R-1} \bigwedge_{k'=k+1}^{R-1} (\neg Q_{i,k} \vee \neg Q_{i,k'})$$

  Questo è equivalente a $Q_{i,k} \implies \neg Q_{i,k'}$ (se è nello stato k, non può essere nello stato k').

- **Unicità della posizione della testina per passo**: Ad ogni passo i, la testina si trova in *al più una* cella.

$$\bigwedge_{i=0}^{N_{max}} \bigwedge_{j=0}^{N_{max}} \bigwedge_{j'=j+1}^{N_{max}} (\neg H_{i,j} \vee \neg H_{i,j'})$$

  Questo è equivalente a $H_{i,j} \implies \neg H_{i,j'}$ (se la testina è in j, non può essere in j').

- **Unicità del simbolo per cella per passo**: Ad ogni passo i e per ogni cella j, la cella contiene *al più un* simbolo.

$$\bigwedge_{i=0}^{N_{max}} \bigwedge_{j=0}^{N_{max}} \bigwedge_{l=0}^{|\Gamma|-1} \bigwedge_{l'=l+1}^{|\Gamma|-1} (\neg T_{i,j,l} \vee \neg T_{i,j,l'})$$

  Questo è equivalente a $T_{i,j,l} \implies \neg T_{i,j,l'}$ (se la cella j contiene α_l, non può contenere $\alpha_{l'}$).

Il numero totale di clausole in C è polinomiale. Queste clausole assicurano che l'assegnamento di verità sia internamente consistente.

2.3.2 2. Inizio (S)

La clausola S impone la configurazione iniziale della macchina M all'istante $i = 0$. Assumiamo q_0 sia lo stato iniziale di M. La stringa input è $W = w_0 w_1 \dots w_{n-1}$.

$$S = Q_{0,q_0} \wedge H_{0,0} \wedge \left(\bigwedge_{j=0}^{n-1} T_{0,j,w_j} \right) \wedge \left(\bigwedge_{j=n}^{N_{max}} T_{0,j,blank} \right)$$

- Q_{0,q_0}: La macchina è nello stato iniziale q_0 al passo 0.
- $H_{0,0}$: La testina è sulla cella 0 al passo 0.
- T_{0,j,w_j} per $j \in [0, n-1]$: Le prime n celle del nastro contengono l'input W.
- $T_{0,j,blank}$ per $j \in [n, N_{max}]$: Tutte le celle successive sono blank.

Anche S è composta da un numero polinomiale di clausole (tutte unitarie).

2.3.3 3. Prossimo Passo (N)

La clausola N codifica il comportamento passo-passo della macchina. È divisa in tre sottocategorie:

3a. Inerzia (N_I): Celle distanti dalla testina Le celle del nastro che non sono sotto la testina al passo i non cambiano il loro contenuto al passo $i+1$.

$$N_I = \bigwedge_{i=0}^{N_{max}-1} \bigwedge_{j=0}^{N_{max}} \bigwedge_{l=0}^{|\Gamma|-1} ((\neg H_{i,j} \wedge T_{i,j,l}) \implies T_{i+1,j,l})$$

Questa implicazione, convertita in CNF, diventa $\neg(\neg H_{i,j} \wedge T_{i,j,l}) \vee T_{i+1,j,l}$, che è equivalente a $(H_{i,j} \vee \neg T_{i,j,l} \vee T_{i+1,j,l})$.

3b. Transizione di stato/testina/simbolo (N_H): Cella sotto la testina Le celle sotto la testina cambiano il loro contenuto e la posizione della testina si sposta, in base alla funzione di transizione δ di M. Per ogni $i \in [0, N_{max}-1]$, $j \in [0, N_{max}]$ e per ogni $k \in [0, R-1]$, $l \in [0, |\Gamma|-1]$: Se al passo i, la macchina è nello stato q_k, la testina è in posizione j, e la cella j contiene α_l, allora devono essere vere le condizioni sul prossimo passo $(i+1)$ dettate dalla funzione di transizione.

Per ogni tupla (q_k, α_l), sia $\delta(q_k, \alpha_l) = \{(q_{k_1}, \alpha_{l_1}, D_1), (q_{k_2}, \alpha_{l_2}, D_2), \dots\}$ dove $D_x \in \{L, R, S\}$ (Left, Right, Stay).

$$N_H = \bigwedge_{i=0}^{N_{max}-1} \bigwedge_{j=0}^{N_{max}} \bigwedge_{k=0}^{R-1} \bigwedge_{l=0}^{|\Gamma|-1} ((Q_{i,k} \wedge H_{i,j} \wedge T_{i,j,l}) \implies \bigvee_{(q_{k'}, \alpha_{l'}, D) \in \delta(q_k, \alpha_l)} (Q_{i+1,k'} \wedge T_{i+1,j,l'} \wedge H_{i+1,j'}))$$

Dove j' è la nuova posizione della testina basata su D:

- Se $D = R$, allora $j' = j+1$.
- Se $D = L$, allora $j' = j-1$.
- Se $D = S$, allora $j' = j$.

(Occorre gestire i casi limite di $j = 0$ per $D = L$ e $j = N_{max}$ per $D = R$, assicurando che la testina non esca dai limiti del nastro rilevante o che tali transizioni non siano ammesse). Questa formula, pur complessa, è polinomiale in dimensione e può essere trasformata in CNF.

3c. Padding (N_P): Ripetizione degli stati finali/bloccanti Questa clausola assicura che tutti i rami di computazione abbiano lunghezza $P(n)$, ripetendo la configurazione finale (accettante o bloccante) fino al passo $P(n)$.

- **Se M entra in uno stato finale accettante** q_{acc}:

$$\bigwedge_{i=0}^{N_{max}-1} \bigwedge_{j=0}^{N_{max}} \bigwedge_{l=0}^{|\Gamma|-1} ((Q_{i,q_{acc}} \wedge H_{i,j} \wedge T_{i,j,l}) \implies (Q_{i+1,q_{acc}} \wedge H_{i+1,j} \wedge T_{i+1,j,l}))$$

  Questa clausola afferma che se al passo i la macchina è nello stato accettante q_{acc}, con testina in j e simbolo α_l, allora al passo $i+1$ la macchina rimane nello stesso stato, la testina non si muove e il contenuto del nastro rimane invariato.

- **Se M si blocca (nessuna transizione definita)**: Per ogni stato q_k e simbolo α_l tale che $\delta(q_k, \alpha_l) = \emptyset$:

$$\bigwedge_{i=0}^{N_{max}-1} \bigwedge_{j=0}^{N_{max}} \bigwedge_{k \text{ s.t. } \delta(q_k,\alpha_l)=\emptyset} \bigwedge_{l=0}^{|\Gamma|-1} ((Q_{i,k} \wedge H_{i,j} \wedge T_{i,j,l}) \implies (Q_{i+1,k} \wedge H_{i+1,j} \wedge T_{i+1,j,l}))$$

  Questa clausola assicura che se la macchina si trova in una configurazione in cui non sono definite transizioni, essa "stalla", ripetendo la stessa configurazione nel passo successivo.

Queste clausole garantiscono che tutte le computazioni, sia quelle accettanti che quelle bloccanti, vengano estese fino alla lunghezza $P(n)$.

2.3.4 4. Finale (F)

La clausola F impone che la macchina M debba terminare in uno stato accettante all'ultimo passo $N_{max} = P(n)$. Assumiamo q_{acc} sia lo stato accettante finale.

$$F = Q_{N_{max},q_{acc}}$$

Questa è una singola clausola unitaria, che forza l'assegnamento di verità a $Q_{P(n),q_{acc}}$ ad essere vero.

3 Conclusione della Dimostrazione

La formula completa $F(W) = C \wedge S \wedge N \wedge F$ ha le seguenti proprietà:

1. **Dimensione Polinomiale**: Tutte le parti di $F(W)$ sono congiunzioni di un numero polinomiale di clausole, e ogni clausola ha una dimensione costante. Pertanto, la dimensione totale di $F(W)$ è polinomiale rispetto alla lunghezza dell'input n. Questo significa che la funzione di riduzione f è calcolabile in tempo polinomiale.
2. **Forma CNF**: Come dimostrato nell'esempio e per la natura delle implicazioni utilizzate, $F(W)$ può essere espressa in forma congiuntiva normale.
3. **Correttezza**:
   - Se $W \in L$: Per definizione di NP, esiste almeno un ramo di computazione accettante di M su W che termina entro $P(n)$ passi. Un assegnamento di verità che descrive fedelmente questo ramo di computazione renderà $F(W)$ vera, quindi $F(W)$ è soddisfacibile.

- Se $W \notin L$: Nessun ramo di computazione di M su W accetta. Qualsiasi assegnamento di verità che tenti di rendere $F(W)$ vera dovrà violare una delle clausole (es. una transizione non valida in N, o non raggiungendo q_{acc} in F), rendendo $F(W)$ insoddisfacibile.

Quindi, abbiamo dimostrato che per ogni linguaggio $L \in$ NP, $L \leq_P$ SAT. Per definizione, questo implica che SAT è **NP-Hard**. Poiché SAT è anche in NP (come ricordato all'inizio), ne consegue che **SAT è NP-Complete**.

Questo risultato è di fondamentale importanza, in quanto stabilisce SAT come il "problema prototipo" della classe NP-Complete. Molti altri problemi NP-Complete vengono dimostrati tali attraverso riduzioni da SAT. Inoltre, esistono programmi chiamati "SAT solvers" che cercano di determinare la soddisfacibilità di formule booleane. La capacità di ridurre problemi NP-Complete a SAT significa che questi SAT solvers possono essere usati per risolvere istanze di un'ampia varietà di problemi difficili, trasformandoli in problemi di soddisfacibilità booleana.