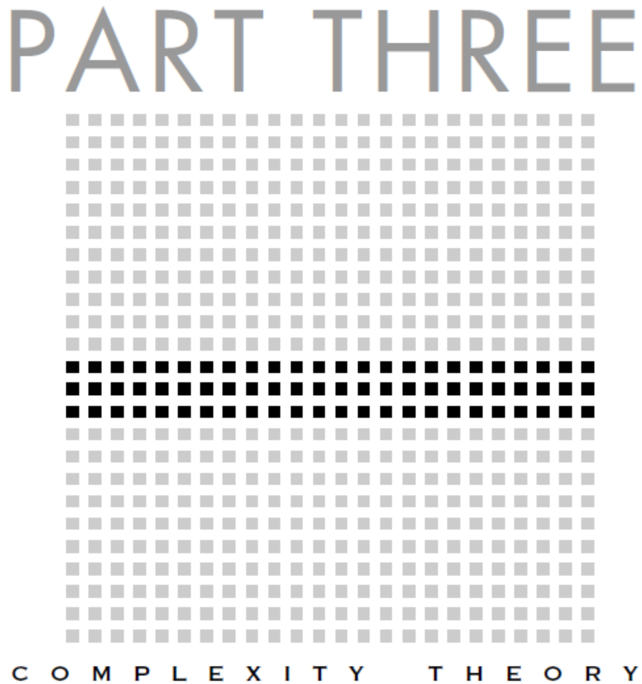

TEORIA DELLA COMPLESSITA'

NP-completezza

25 maggio 2023

Teoria della complessità: argomenti trattati

Scorse lezioni:

- Definizione di complessità di tempo
- La complessità di tempo dipende dal modello di calcolo; useremo decisori e modelli polinomialmente equivalenti
- La complessità di tempo dipende dalla codifica utilizzata: useremo codifica in binario o polinomialmente correlata
- TIME (f(n)) = insieme dei linguaggi decisi in tempo O(f(n))
- La classe $P = \bigcup_{k \geq 0} TIME(n^k)$ e sua robustezza
- La classe EXPTIME
- Algoritmi di verifica e la classe NP

Oggi:

- Il concetto di riduzione polinomiale
- Il concetto di NP-completezza

Definizione

Siano A, B linguaggi sull'alfabeto Σ.

Una riduzione in tempo polinomiale f di A in B è

- una funzione $f : \Sigma^* \to \Sigma^*$
- calcolabile in tempo polinomiale
- tale che per ogni $w \in \Sigma^*$

$$w \in A \Leftrightarrow f(w) \in B$$

Definizione

Un linguaggio $A \subseteq \Sigma^*$ è riducibile in tempo polinomiale a un linguaggio $B \subseteq \Sigma^*$, e scriveremo $A \leq_p B$, se esiste una riduzione di tempo polinomiale di A in B.

3 SAT

Definizione

Una clausola è un OR di letterali.

Esempio: $(\overline{x} \vee x \vee y \vee z)$

CNF

Definizione

Una formula booleana ϕ è in forma normale congiuntiva (o forma normale POS) se è un AND di clausole, cioè è un AND di OR di letterali.

Definizione

Una formula booleana è in forma normale 3-congiuntiva se è un AND di clausole e tutte le clausole hanno tre letterali.

Esempio:

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_3 \vee \overline{x_6} \vee x_6) \wedge (x_3 \vee \overline{x_5} \vee x_5)$$

$$\boxed{3SAT = \{\langle\phi\rangle \mid \phi \text{ è una formula 3CNF soddisfacibile}\}}$$

3SAT e *CLIQUE*

Teorema

$$\boxed{3SAT \leq_P CLIQUE}$$

$3SAT = \{\langle \phi \rangle \mid \phi$ è una formula 3CNF soddisfacibile$\}$

Una formula 3CNF è un *AND* di clausole e tutte le clausole hanno tre letterali.

$CLIQUE =$
$\{\langle G, k \rangle \mid G$ è un grafo non orientato in cui esiste una k-clique$\}$

Ricorda:
Una clique (o cricca) in un grafo non orientato G è un sottografo G' di G in cui ogni coppia di vertici è connessa da un arco.
Una k-clique è una clique che contiene k vertici.

$3SAT \leq_p CLIQUE$

Dobbiamo dimostrare che esiste una funzione $f : \Sigma^* \rightarrow \Sigma^*$

- calcolabile in tempo polinomiale
- tale che per ogni $w \in \Sigma^*$ $w \in 3SAT \Leftrightarrow f(w) \in CLIQUE$

Convenzione: non specificheremo il valore di f sulle stringhe che non rappresentano un'istanza del problema.

Quindi definiremo la f solo su stringhe che codificano formule booleane in 3CNF ϕ e ad esse associeremo stringhe che codificano (G, k).

$$f : \langle\phi\rangle \rightarrow \langle G, k\rangle$$

$$\langle\phi\rangle \in 3SAT \Leftrightarrow \langle G, k\rangle \in CLIQUE$$

$3SAT \leq_p CLIQUE$

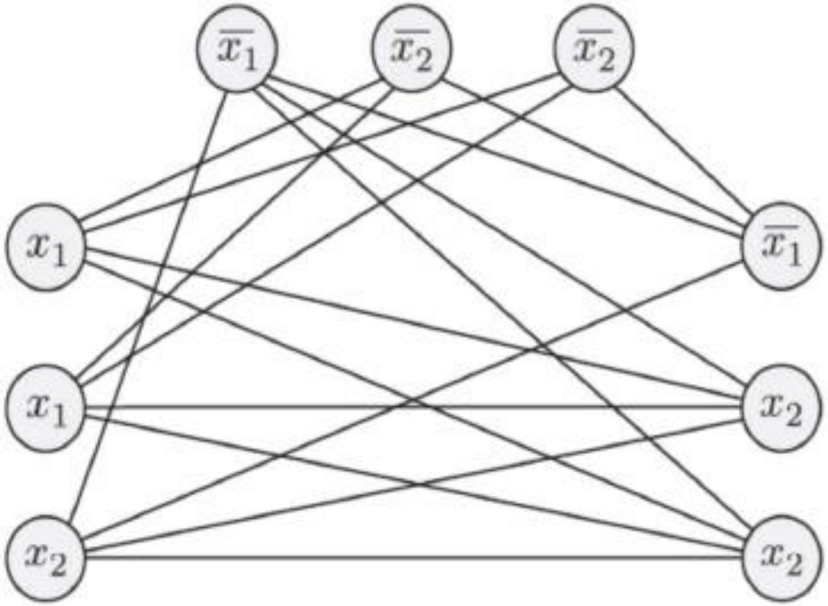

FIGURA 7.33
Il grafo che la riduzione produce per $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

Riducibilità in tempo polinomiale: teoremi

Teorema

Se $A \leq_p B$ e $B \in P$, allora $A \in P$.

Dimostrazione

- Per ipotesi $B \in P$, quindi esiste un algoritmo M, di complessità $O(m^t)$, che decide B.
- Inoltre $A \leq_p B$: sia f la riduzione di tempo polinomiale di A in B e sia F l'algoritmo, di complessità $O(n^k)$, che calcola la funzione f .
- Consideriamo l'algoritmo N che sull'input w:
  - simula F su w e calcola f (w)
  - simula M sull'input f (w) per decidere se $f(w) \in B$
  - N accetta w se M accetta f (w), N rifiuta w se M rifiuta f (w)

$$N : w \rightarrow \boxed{\boxed{F} \rightarrow f(w) \rightarrow \boxed{M}}$$

Riducibilità in tempo polinomiale: teoremi

$$N : w \rightarrow \boxed{\boxed{F} \rightarrow f(w) \rightarrow \boxed{M}}$$

N decide A (correttezza dell'algoritmo N).

N è un decider. Infatti N si ferma su w se si fermano F ed M. Ora, per ogni w, F si ferma con $f(w)$ sul nastro e per ogni w, M si ferma su $f(w)$ perché M è un decider.

Inoltre N riconosce A. Infatti

$w \in L(N) \Leftrightarrow f(w) \in L(M)$ (per la definizione di N)

$\Leftrightarrow f(w) \in B$ (perché M decide B)

$\Leftrightarrow w \in A$ (perché f è una riduzione polinomiale di A a B)

Riducibilità in tempo polinomiale: teoremi

$$N : w \rightarrow \boxed{\boxed{F} \rightarrow f(w) \rightarrow \boxed{M}}$$

- N è un algoritmo polinomiale in $n = |w|$.

  Infatti, F calcola $f(w)$ in $O(n^k)$ passi (primo passo dell'algoritmo: polinomiale).

  Inoltre risulta $|f(w)| = O(n^k)$ (cioè, per n sufficientemente grande, $|f(w)| \leq cn^k$ perché **la lunghezza dell'output di F è limitata dalla complessità di tempo di F**)

  Al secondo passo M viene eseguito sull'input $f(w)$ e si arresterà dopo $c'|f(w)|^t \leq c'(cn^k)^t$ passi, cioè dopo $O(n^{kt})$ passi (c', c, k, t costanti. Secondo passo dell'algoritmo: polinomiale, composizione dei due polinomi).

  In conclusione N ha complessità $O(n^k) + O(n^{kt}) = O(n^{kt})$.
- Quindi $A \in P$.

Teorema

Se $A \leq_p B$ e $B \leq_p C$, allora $A \leq_p C$.

Dimostrazione

- Per ipotesi: esiste una riduzione di tempo polinomiale $f : \Sigma^* \to \Sigma^*$ di A a B ed esiste una riduzione di tempo polinomiale $g : \Sigma^* \to \Sigma^*$ di B a C.
- Consideriamo la composizione $g \circ f : \Sigma^* \to \Sigma^*$ delle funzioni f e g, definita da $(g \circ f)(w) = g(f(w))$.
- Risulta, per ogni $w \in \Sigma^*$:

$$
\begin{aligned}
w \in A \quad &\Leftrightarrow \quad f(w) \in B \text{ (perché } f \text{ è una riduzione polinomiale} \\
&\qquad \text{di } A \text{ a } B\text{)} \\
&\Leftrightarrow \quad g(f(w)) \in C \text{ (perché } g \text{ è una riduzione polinomiale} \\
&\qquad \text{di } B \text{ a } C\text{)}
\end{aligned}
$$

- Inoltre la funzione $g \circ f$ è una funzione calcolabile in tempo polinomiale.

Transitività

- Infatti, sia F l'algoritmo di complessità $O(n^k)$ che calcola la funzione f, sia G l'algoritmo di complessità $O(m^t)$ che calcola la funzione g.
- Consideriamo l'algoritmo GF che sull'input w:
  1. simula F su w e calcola $f(w)$,
  2. simula G sull'input $f(w)$ e calcola $g(f(w))$
  3. fornisce in output l'output di G.

- L'algoritmo GF calcola $g \circ f$ perchè prima esegue F su w calcolando $f(w)$ (primo passo dell'algoritmo) e poi G su $f(w)$ (secondo passo dell'algoritmo) fornendo quindi in output $g(f(w))$.

- Quindi $g \circ f$ è una riduzione di tempo polinomiale di A a C.

P = la classe dei linguaggi L per i quali l'appartenenza di una stringa w ad L può essere decisa da un algoritmo polinomiale in |w|.

NP = la classe dei linguaggi L per i quali l'appartenenza di una stringa w ad L può essere verificata da un algoritmo polinomiale in |w|.

Teorema 7.20
Un linguaggio L è in NP se e solo se esiste una macchina di Turing non deterministica che decide L in tempo polinomiale.

Teorema
$P \subseteq NP$

Uno dei più grandi problemi aperti dell'informatica teorica:

$$P = NP\ ?$$

Uno dei più grandi problemi aperti dell'informatica teorica:

P = NP ?

Come affrontare il problema? Approccio seguito:

Cerchiamo i problemi più difficili della classe NP.
Così se L è (il linguaggio associato a) uno dei problemi più difficili di NP:

1. Dovrebbe essere più semplice dimostrare che non è in P, e quindi concludere che: P $\neq$ NP
2. Se, invece, riuscissimo a dimostrare che L è in P (cioè trovare un algoritmo polinomiale per decidere L) anche tutti gli altri linguaggi di NP (che sono meno difficili), dovrebbero essere in P e potremmo concludere che NP = P

Un progresso importante sulla questione ‘’P = NP?" ci fu all'inizio degli anni '70 con il lavoro di Stephen Cook e Leonid Levin.

Essi definirono i linguaggi «più difficili» della classe NP e li chiamarono linguaggi NP-completi.

Il fenomeno della NP-completezza è importante sia per ragioni teoriche che pratiche.

Il primo linguaggio NP-completo che fu (da loro) scoperto è SAT, il problema della soddisfacibilità di una formula booleana.

Vogliamo definire quando un linguaggio B è uno dei linguaggi «più difficili» della classe NP.

Abbiamo visto un modo per definire quando B è «più difficile» di A, ovvero quando A è di difficoltà «minore o uguale» a B:

$$A \leq_p B$$

Quindi B è uno dei linguaggi «più difficili» della classe NP…..

Definizione

Un linguaggio B è *NP-completo* se soddisfa le seguenti due condizioni:

1. B appartiene a NP
2. Per ogni linguaggio A in NP, $A \leq_p B$ (ovvero B è NP-hard)

NP – completezza: teoremi fondamentali

Teorema

Se B è NP-completo e B è in P allora $P = NP$.

Dimostrazione.

- Siccome B è NP-completo, per ogni $A \in NP$, risulta $A \leq_P B$
- Ma abbiamo provato che se $A \leq_P B$ e $B \in P$ allora $A \in P$
- Quindi $NP \subseteq P$ e siccome $P \subseteq NP$ risulta $P = NP$.

NP – completezza: teoremi fondamentali

Ma esistono linguaggi NP-completi?

Ma esistono linguaggi indecidibili?

Come abbiamo dimostrato che A_{TM}, $HALT_{TM}$, E_{TM}, $REGULAR_{TM}$, EQ_{TM} e tanti altri linguaggi sono indecidibili?

Abbiamo provato che:

1. A_{TM} è indecidibile, dalla definizione, per assurdo
2. Se B è indecidibile e $B \leq_m C$ allora anche C è indecidibile

Analogamente affermeremo che:

1. SAT è NP-completo (Teorema di Cook-Levin)
2. Se B è NP-completo e $B \leq_p C$, con $C \in NP$, allora anche C è NP-completo.

Nota: $\leq_m$ e $\leq_p$

Teorema (Cook-Levin)
SAT è NP-completo.

Si ha facilmente che SAT $\in$ NP (segue dimostrazione).
Il resto della dimostrazione è complessa (non è in programma).
Occorre mostrare che **per ogni** A $\in$ NP (e sappiamo solo questo) si ha $A \leq_p SAT$.
L'idea è la seguente.
La riduzione di tempo polinomiale si ottiene definendo per ogni input w una formula booleana che simula la macchina di Turing non deterministica che decide A sull'input w.

Conseguenza: il teorema non dice che SAT non si possa risolvere in tempo polinomiale, ma che:

SAT $\in$ P se e solo se P = NP.

SAT

Una formula booleana ϕ è soddisfacibile se esiste un insieme di valori 0 o 1 per le variabili di ϕ (o assegnamento) che renda la formula uguale a 1 (assegnamento di soddisfacibilità). Diremo che tale assegnamento soddisfa ϕ o anche che rende vera ϕ.

Il problema della **soddisfacibilità di una formula booleana**:
Data una formula booleana ϕ, ϕ è soddisfacibile?

Il linguaggio associato è:

$$\boxed{SAT = \{\langle\phi\rangle \mid \phi \text{ è una formula booleana soddisfacibile}\}}$$

Teorema

$SAT \in NP$.

Dimostrazione.

Un certificato per $\langle \phi \rangle$ sarà un assegnamento c di valori alle variabili di ϕ. Un algoritmo V che verifica SAT in tempo polinomiale nella lunghezza di $\langle \phi \rangle$:

$V =$ "Sull'input y:

1. Verifica se $y = \langle \langle \phi \rangle, c \rangle$, dove ϕ è una formula booleana e c è un assegnamento di valori alle variabili di ϕ, altrimenti rifiuta.
2. Sostituisce ogni variabile della formula con il suo corrispondente valore e quindi valuta l'espressione.
3. Accetta se ϕ assume valore 1; altrimenti rifiuta."

$\exists c : \langle \langle \phi \rangle, c \rangle \in L(V) \Leftrightarrow \langle \phi \rangle \in SAT$ $\square$

NP – completezza: teoremi fondamentali

Teorema

Se B è NP-completo e $B \leq_p C$, con $C \in NP$, allora C è NP-completo.

Dimostrazione

Per ipotesi:

1. $C \in NP$
2. Per ogni $A \in NP$, $A \leq_p B$ (B è NP-completo)
3. $B \leq_p C$

Allora, utilizzando la proprietà transitiva di $\leq_p$:

1. $C \in NP$

5.(=2+3) Per ogni $A \in NP$, $A \leq_p C$

Cioè C è NP-completo.

Provare la *NP* – completezza

Teorema

Se B è NP-completo e $B \leq_p C$, con $C \in NP$, allora C è NP-completo.

Una possibile strategia per provare che un linguaggio C è NP-completo:

1. Mostrare che $C \in NP$
2. Scegliere un linguaggio B che sia NP-completo
3. Definire una riduzione di tempo polinomiale di B in C.

Provare la *NP* – completezza

Una possibile strategia per provare che un linguaggio C è NP-completo:

1. Mostrare che $C \in NP$
2. Scegliere un linguaggio B che sia NP-completo
3. Definire una riduzione di tempo polinomiale di B in C.

Proveremo che alcuni linguaggi sono NP-completi mostrando una riduzione di tempo polinomiale da 3SAT che utilizza la tecnica di ‘’riduzione mediante progettazione di componenti" o ‘’gadgets".

Occorre prima dimostrare che 3SAT è NP-completo.

3SAT è *NP* – completo

Occorre prima dimostrare che 3SAT è NP-completo.

- 3SAT $\in$ NP. Infatti 3SAT è verificabile in tempo polinomiale perché è un caso particolare di SAT (che sappiamo essere in NP).

- E' possibile dimostrare poi che:

$$\text{SAT} \leq_p \text{SAT}_{\text{CNF}} \leq_p \text{3 SAT}$$

Nota: 3SAT pur essendo un caso particolare di SAT è di «difficoltà maggiore o uguale» a SAT.

SAT_{CNF} è NP – completo

$$SAT_{CNF} = \{\langle\phi\rangle \mid \phi \text{ è una formula booleana soddisfacibile in } CNF\}$$

$SAT_{CNF} \in NP$,

SAT è riducibile in tempo polinomiale a SAT_{CNF}

$$SAT \leq_P SAT_{CNF}$$

Teorema

SAT_{CNF} è NP-completo.

(Senza dimostrazione)

- Nota: la trasformazione classica di un'espressione booleana nella sua forma normale congiuntiva non definisce, in generale, una riduzione di tempo polinomiale.

$3SAT$ è NP-completo

3CNF = formula booleana in forma normale 3-congiuntiva

$$3SAT = \{\langle\phi\rangle \mid \phi \text{ è una formula 3CNF soddisfacibile}\}$$

Teorema

$3SAT$ è NP-completo.

Dimostrazione

- $3SAT$ è in NP.
- Per provare che $3SAT$ è NP-completo basta dimostrare che $SAT_{CNF} \leq_P 3SAT$.
- La prova consiste nel costruire, a partire da ϕ in CNF, una formula booleana ψ in 3CNF tale che ϕ è soddisfacibile se e solo se ψ è soddisfacibile.
- Inoltre ψ può essere costruita a partire da ϕ in tempo polinomiale.

3SAT è *NP* – completo

Ora che abbiamo scritto la formula in forma cnf, la convertiamo in una con tre letterali per clausola. In ciascuna clausola che correntemente ha uno o due letterali, duplichiamo uno dei letterali, fino a quando il numero totale diventa tre. Ciascuna clausola che ha più di tre letterali la dividiamo in più clausole e aggiungiamo ulteriori variabili per preservare la soddisfacibilità o non soddisfacibilità della clausola originale.

Per esempio, sostituiamo la clausola $(a_1 \vee a_2 \vee a_3 \vee a_4)$, dove ciascun a_i è un letterale, con l'espressione di due clausole $(a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$, in cui z è una variabile nuova. Se qualche impostazione degli a_i soddisfa la clausola originale, possiamo trovare una impostazione di z in modo tale che le due nuove clausole siano soddisfatte e viceversa. In generale, se la clausola contiene l letterali,

$$(a_1 \vee a_2 \vee \cdots \vee a_l),$$

possiamo sostituirla con le $l-2$ clausole

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l).$$

È possibile verificare facilmente che la nuova formula è soddisfacibile se e solo se la formula originale lo era, quindi la dimostrazione è completa.

Teorema

CLIQUE è NP-completo.

Dimostrazione.

- Sappiamo che $CLIQUE \in NP$.
- Inoltre, $3SAT$ è NP-completo e $3SAT \leq_P CLIQUE$
- Quindi $CLIQUE$ è NP-completo.

$\square$

Problemi *NP* – completi

- SAT (Cook-Levin)
- SAT_{CNF} (senza dimostrazione)
- 3SAT (cenni)
- CLIQUE (da 3SAT)

- CLIQUE (da 3SAT coi gadget)
- VERTEX-COVER (da 3SAT coi gadget)
- SUBSET-SUM (da 3SAT)
- HAMPATH (da 3SAT coi gadget)

- UHAMPATH (da HAMPATH)

Esercizio

La seguente affermazione è vera?

"Comunque prendo due linguaggi NP-completi A e B, si ha:

$$A \leq_p B \quad e \quad B \leq_p A\ ."$$

Cioè, i linguaggi NP-completi hanno tutti «uguale difficoltà».