PART THREE

COMPLEXITY THEORY

TEORIA DELLA COMPLESSITA’

II parte

13 maggio 2022

Calcolabilità e complessità

Calcolabilità: si occupa di problemi risolvibili algoritmicamente in linea di principio.

Domande che affronta:

Quali problemi sono risolvibili?

Cosa significa procedura effettiva di calcolo?

Complessità: si occupa di problemi risolvibili algoritmicamente in pratica.

La teoria della Complessità analizza problemi risolvibili.

Domande che affronta:

Quali sono le risorse minime necessarie (es. tempo di calcolo e memoria) per la risoluzione di un problema?

Come si misura il consumo delle risorse?

Teoria della complessità: argomenti trattati

Ieri:

- 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 $P = \bigcup_{k \geq 0} \mathrm{TIME}(n^k)$ e sua robustezza

Oggi:

- La classe EXPTIME
- La classe NP

La classe *EXPTIME*

Oltre la classe $P = \bigcup_{k \geq 0} TIME(n^k)$ possiamo definire la classe

$$EXPTIME = \bigcup_{k \geq 1} TIME(2^{n^k})$$

Ovviamente $P \subseteq EXPTIME$.
Inoltre P è strettamente incluso in $EXPTIME$, ovvero esistono linguaggi in $EXPTIME \setminus P$.

I linguaggi di P sono associati a problemi trattabili

I linguaggi di $EXPTIME \setminus P$ sono associati a problemi intrattabili

Un problema intrattabile

Un esempio di linguaggio di *EXPTIME \ P* ovvero di un problema intrattabile.

Abbiamo dato delle espressioni regolari una definizione ricorsiva. La regola induttiva permette di costruire una nuova espressione regolare a partire dalle espressioni regolari R_1 ed R_2, usando le operazioni $\cup$, $\circ$ e $*$.

Le espressioni regolari generalizzate (o ERG) aggiungono l'operazione $\uparrow$: se R è un'espressione regolare e $k \in \mathbb{N}$, $R \uparrow k$ è la concatenazione (o prodotto) di R con se stessa k volte.

Sia

$$EQ_{REX\uparrow} = \{\langle Q, R \rangle \mid Q \text{ ed } R \text{ sono } ERG \text{ equivalenti}\}$$

$$\boxed{EQ_{REX\uparrow} \in EXPTIME \setminus P}$$

Problemi probabilmente intrattabili

I problemi corrispondenti ai linguaggi in *EXPTIME \ P* non sono in genere importanti nelle applicazioni pratiche.

Sono più comuni problemi (ovvero linguaggi) decidibili, ma tali che gli algoritmi attualmente noti per decidere tali linguaggi richiedono tempo esponenziale.

Per comprendere meglio questi problemi è stata introdotta una nuova classe di complessità, la classe NP.

Vediamo prima un esempio.

Problema del cammino Hamiltoniano

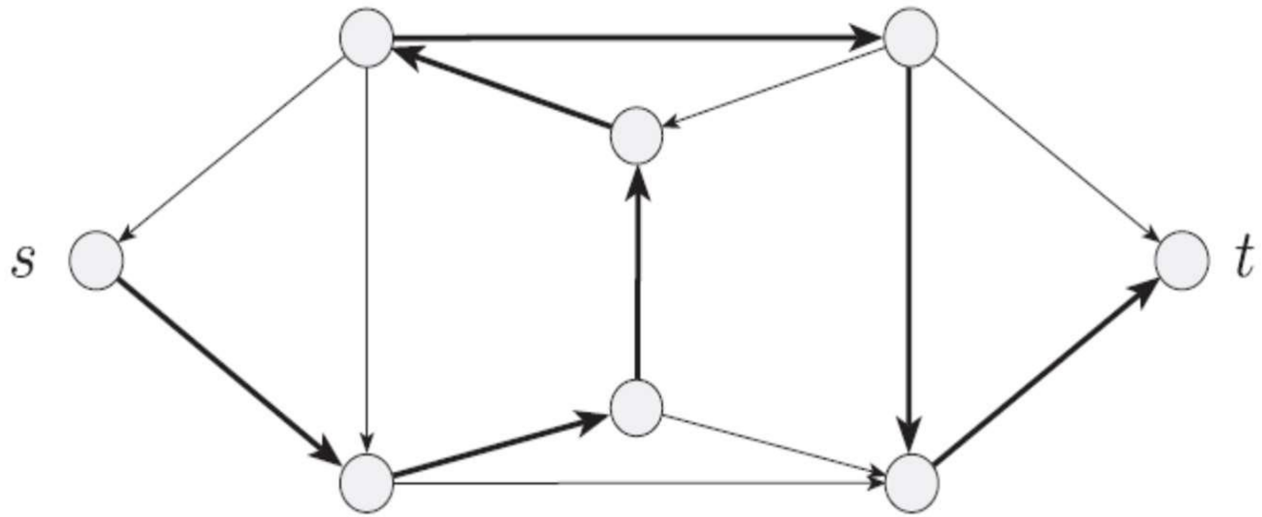


FIGURA 7.17
Un cammino Hamiltoniano attraversa ogni nodo esattamente una volta

Problema del cammino Hamiltoniano

HAMPATH = { $\langle$ G, s, t$\rangle$ | G è un grafo orientato, s e t vertici, G ha un cammino Hamiltoniano da s a t }

HAMPATH può essere deciso in tempo esponenziale da un algoritmo di forza bruta che considera tutti i cammini semplici da s e a t.

Non si conoscono algoritmi polinomiali che decidono HAMPATH, ma nemmeno si riesce a dimostrare che non ne esistano.

HAMPATH ha però una importante caratteristica chiamata verificabilità polinomiale.

Verificabilità di HAMPATH

Non si sa se esista un algoritmo polinomiale che risolve HAMPATH, però, se qualcuno ci fornisse una sequenza

$$c = (v_1, v_2 , \dots , v_k)$$

di vertici di G, potremmo facilmente verificare se c è un cammino Hamiltoniano da s a t in G.

Basterebbe verificare che

- $v_1 = s$ e $v_k = t$
- per ogni $i = 1, \dots , k-1$, (v_i, v_{i+1}) è un arco di G
- k è pari al numero di vertici
- tutti i vertici di c sono distinti

La verifica potrebbe essere fatta in tempo polinomiale.

Verificabilità di HAMPATH

HAMPATH = { $\langle$ G, s, t$\rangle$ | G è un grafo orientato, s e t vertici, G ha un cammino Hamiltoniano da s a t }

Esiste un algoritmo A di tempo polinomiale in $|\langle$ G, s, t $\rangle|$ che decide il linguaggio

{ $\langle\langle$ G, s, t $\rangle$, $\langle$ c $\rangle\rangle$ | G è un grafo orientato, s e t vertici, c = (v_1, v_2 , … , v_k) e c è un cammino Hamiltoniano da s a t }

Nota: anche $|\langle$ c $\rangle|$ è polinomiale in $|\langle$ G, s, t $\rangle|$.

Nota: c è chiamato certificato.

Un altro esempio: *COMPOSITES*

Un numero è composto se è prodotto di due interi maggiori di 1; ovvero quando non è primo.

Problema: stabilire se un intero è composto.

Il linguaggio associato è

COMPOSITES = $\{ \langle x \rangle \mid x = pq$ con p, q interi $p, q > 1\}$

Un problema non verificabile

Consideriamo il complemento di HAMPATH

{ ⟨ G, s, t⟩ | G è un grafo orientato, s e t vertici, G non ha un cammino Hamiltoniano da s a t }

Anche se riuscissimo a determinare che G non ha un cammino Hamiltoniano da s a t, non abbiamo un algoritmo polinomiale per verificarne l'inesistenza!

Algoritmo di verifica

Definizione

Un **algoritmo di verifica (o verificatore)** *V per un linguaggio A è un algoritmo tale che*

$$A = \{w \mid \exists c \text{ tale che } V \text{ accetta } \langle w, c\rangle\}$$

La stringa c prende il nome di **certificato** *o* **prova**.

A è il **linguaggio verificato** *da V.*

Algoritmo di verifica polinomiale

Definizione

Un algoritmo V è un verificatore per A in tempo **polinomiale** *se:*

- *A è il linguaggio verificato da V, cioè*

$$A = \{w \mid \exists c \text{ tale che } V \text{ accetta } \langle w, c \rangle\}$$

- *V ha complessità di tempo polinomiale in $|w|$.*

Nota: se V è un algoritmo di verifica e ha complessità polinomiale in $|w|$, allora il certificato ha **lunghezza polinomiale** nella lunghezza di w, cioè esiste t tale che per ogni w, $|c| = O(|w|^t)$.

Definizione

NP è la classe dei linguaggi verificabili in tempo polinomiale.

- Esempi.
  - Per *HAMPATH* un certificato per una stringa $\langle G, s, t\rangle \in HAMPATH$ è un cammino Hamiltoniano da s a t.
  - Per *COMPOSITES* un certificato per una stringa $\langle x\rangle \in COMPOSITES$ è uno dei divisori di x.

Nota: *NP* non è l'abbreviazione di tempo Non Polinomiale.
Il nome deriva dalla seguente caratterizzazione.

La classe *NP*

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.

Definizione

Sia $t : \mathbb{N} \rightarrow \mathbb{R}^+$ *una funzione. La classe di complessità in tempo non deterministico* $NTIME(t(n))$ *è*

$$NTIME(t(n)) = \{L \mid \exists \text{ una macchina di Turing non deterministica } M \text{ che decide } L \text{ in tempo } O(t(n))\}$$

Corollario 7.22

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

Esempi di linguaggi in *NP*

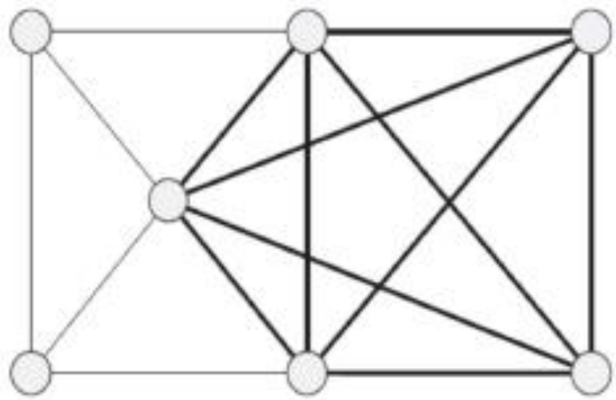


FIGURA 7.23
Un grafo con una 5-clique

Definizione

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

Esempi di linguaggi in *NP*

Il problema di stabilire se un grafo non orientato G contiene una k-clique si può formulare come un problema di decisione, il cui linguaggio associato è $CLIQUE$.

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

Teorema

$CLIQUE \in NP$

Dimostrazione.

Un algoritmo V che verifica $CLIQUE$ in tempo polinomiale:

$V =$ “Sull'input $\langle\langle G, k\rangle, c\rangle$:

1. Verifica se c è un insieme di k nodi di G, altrimenti rifiuta.
2. Verifica se per ogni coppia di nodi in c, esiste un arco in G che li connette, accetta in caso affermativo; altrimenti rifiuta.”

$\exists c : \langle\langle G, k\rangle, c\rangle \in L(V) \Leftrightarrow \langle G, k\rangle \in CLIQUE$ $\square$

Prova alternativa: utilizzare le macchine di Turing non deterministiche.

Esempi di linguaggi in *NP*

SUBSET-SUM: Dato un insieme finito S di numeri interi e un numero intero t, esiste un sottoinsieme S' di S tale che la somma dei suoi numeri sia uguale a t?

$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}$ ed esiste $S' \subseteq S$ tale che $\sum_{s \in S'} s = t\}$

Esempio: $\langle \{4, 11, 16, 21, 27\}, 25\rangle \in SUBSET\text{-}SUM$ perché $4 + 21 = 25$.

Esempi di linguaggi in *NP*

Teorema

$SUBSET\text{-}SUM \in NP$

Dimostrazione.

Un algoritmo V che verifica $SUBSET\text{-}SUM$ in tempo polinomiale:

$V =$ "Sull'input $\langle\langle S, t\rangle, c\rangle$:

1. Verifica se c è un insieme di numeri la cui somma è t, altrimenti rifiuta.
2. Verifica se S contiene tutti i numeri in c, accetta in caso affermativo; altrimenti rifiuta."

$\exists c : \langle\langle S, t\rangle, c\rangle \in L(V) \Leftrightarrow \langle S, t\rangle \in SUBSET\text{-}SUM$ $\square$

Esempi di linguaggi in *NP*

Teorema

$HAMPATH \in NP$

Dimostrazione.

Un algoritmo N che verifica $HAMPATH$ in tempo polinomiale:
$N =$ "Sull'input $\langle\langle G, s, t\rangle, c\rangle$, dove $G = (V, E)$ è un grafo orientato:

1. Verifica se $c = (u_1, \ldots, u_{|V|})$ è una sequenza di $|V|$ vertici di G, altrimenti rifiuta.
2. Verifica se i nodi della sequenza sono distinti, $u_1 = s$, $u_{|V|} = t$ e, per ogni i con $2 \leq i \leq n$, se $(u_{i-1}, u_i) \in E$, accetta in caso affermativo; altrimenti rifiuta."

$\exists c : \langle\langle G, s, t\rangle, c\rangle \in L(N)$ se e solo se $\langle G, s, t\rangle \in HAMPATH$. $\square$

P e *NP*

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 1

$P \subseteq NP$

Teorema 1

$P \subseteq NP$

Dimostrazione

Se $L \in P$, esiste un algoritmo M che decide L in tempo polinomiale.

Consideriamo l'algoritmo di verifica V che sull'input y

- Se $y \neq \langle w, \epsilon \rangle$, w stringa, rifiuta y
- Se $y = \langle w, \epsilon \rangle$, w stringa, simula M su w
- Accetta $y = \langle w, \epsilon \rangle$ se e solo se M accetta w.

V verifica L in tempo polinomiale.

Teorema 1

$P \subseteq NP$

Dimostrazione

Se $L \in P$, esiste un macchina di Turing deterministica $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ che decide L in tempo polinomiale.

Sia $M' = (Q, \Sigma, \Gamma, \delta', q_0, q_{accept}, q_{reject})$ la macchina di Turing non deterministica tale che

$$\delta'(q, \gamma) = \{\delta(q, \gamma)\}$$

per ogni $q \in Q \setminus \{q_{accept}, q_{reject}\}$ e ogni $\gamma \in \Gamma$.

È facile provare che M' è una macchina di Turing non deterministica equivalente ad M e che M' decide L in tempo polinomiale.

Relazioni fra classi di complessità

Teorema

$$P \subseteq NP = \bigcup_{k \geq 1} NTIME(n^k) \subseteq EXPTIME = \bigcup_{k \geq 1} TIME(2^{n^k})$$

Teorema 7.11

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni macchina di Turing a nastro singolo, non deterministica N avente tempo di esecuzione $t(n)$ esiste una macchina di Turing a nastro singolo, deterministica e di complessità di tempo $2^{O(t(n))}$, equivalente ad N.

Dimostrazione

Dobbiamo provare che $P \subseteq NP$ e che $NP \subseteq EXPTIME$.

Il Teorema 1 dimostra l'inclusione $P \subseteq NP$.

Sia $L \in NP$. Per il Teorema 7.20, esiste una macchina di Turing non deterministica N che decide L in tempo $O(n^k)$, per qualche $k \geq 1$.

Per il Teorema 7.11, esiste una macchina di Turing deterministica a un nastro M che decide L con complessità di tempo $2^{O(n^k)}$.

Quindi M decide L con complessità di tempo $O(2^{n^h})$, per qualche $h \geq 1$, cioè $L \in EXPTIME$.

$$P \subseteq NP = \bigcup_{k \geq 1} NTIME(n^k) \subseteq EXPTIME = \bigcup_{k \geq 1} TIME(2^{n^k})$$

E’ noto che P è un sottoinsieme proprio di EXPTIME.
Uno dei più grandi problemi aperti dell'informatica teorica:

P = NP ?

Proposizione
La classe P è chiusa rispetto al complemento.

Invece, non è noto se la classe NP sia o meno chiusa rispetto al complemento.

HAMPATH e il suo complemento

HAMPATH =
{ $\langle$ G, s, t$\rangle$ | G è un grafo orientato, s e t vertici, G ha un cammino Hamiltoniano da s a t }

Se $\langle G, s, t\rangle \in HAMPATH$ esiste un cammino Hamiltoniano c in G da s a t.

Se tale cammino è stato scoperto, è possibile verificare in tempo polinomiale che $\langle G, s, t\rangle \in HAMPATH$: basta fornire in input al verificatore per $HAMPATH$ la stringa $\langle\langle G, s, t\rangle, c\rangle$.

Ma se $\langle G, s, t\rangle \notin HAMPATH$, tale cammino non esiste e non conosciamo alcun algoritmo polinomiale per verificare la non esistenza di tale cammino.

Le osservazioni precedenti si applicano a qualsiasi linguaggio in NP e pongono il problema del rapporto tra la classe NP e la classe $coNP = \{L \mid \overline{L} \in NP\}$.

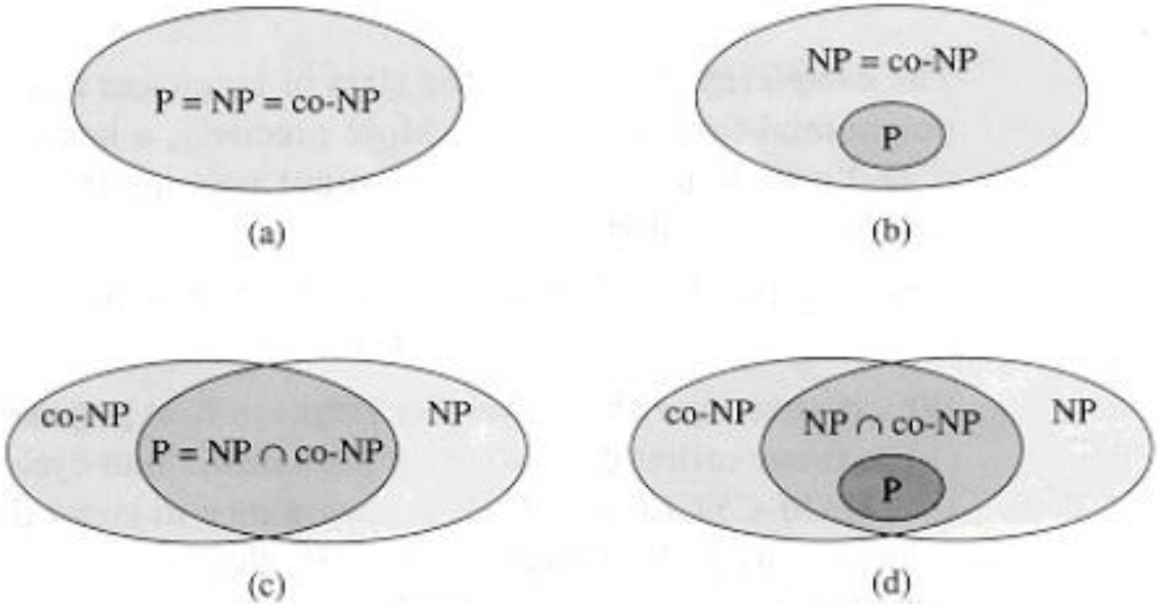
Per ognuno di questi linguaggi non è noto se tale linguaggio appartenga o meno a NP.

Le risposte alle domande

$$P = NP?$$

$$NP = coNP?$$

danno luogo ai seguenti quattro possibili scenari.



Vi sono quattro possibilità:

1. $P = NP = coNP$
2. $P \subsetneq NP = coNP$
3. $NP \neq coNP$, $P = NP \cap coNP \subsetneq NP$ (e quindi $P = NP \cap coNP \subsetneq coNP$)
4. $NP \neq coNP$, $P \subsetneq NP \cap coNP \subsetneq NP$ (e quindi $P \subsetneq NP \cap coNP \subsetneq coNP$)

NP - completezza

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

Essi scoprirono vari linguaggi appartenenti a NP la cui complessità è correlata a quella dell'intera classe NP.
Essi sono i linguaggi «più difficili» della classe NP.

Se esistesse un algoritmo di tempo polinomiale per uno qualsiasi di essi, tutti i linguaggi in NP diventerebbero decidibili in tempo polinomiale.

Questi linguaggi vengono detti NP-completi.
Il fenomeno della NP-completezza è importante sia per ragioni teoriche che pratiche.

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