Complessità computazionale

— ○ —

Principali relazioni tra le classi di complessità

Piero A. Bonatti

Università di Napoli Federico II

Laurea Magistrale in Informatica

Tema della lezione I

- Col procedere del corso studieremo le mutue relazioni tra le classi di complessità, come ad esempio

$$P \subseteq \left\{ \begin{array}{c} NP \\ coNP \end{array} \right\} \subseteq PSPACE \subseteq EXP \ldots$$

- Una ragione per farlo è capire quante risorse aggiuntive servono per risolvere qualche problema in più
  - domanda non banale: ovviamente con più risorse la classe di problemi risolubili non si restringe
  - ma sappiamo già che moltiplicandole semplicemente per una costante non fa risolvere più problemi (speedup theorems)

Tema della lezione II

- Una seconda ragione è capire se esistano relazioni tra spazio e tempo
  - è immediato vedere che il tempo costituisce un limite superiore allo spazio: se una MdT M usa x celle per la computazione, allora il tempo di computazione è $\geq x$
  - dualmente se lo spazio è limitato anche le computazioni possono essere rese limitate (cf. la terminazione delle MdT space bounded)
  - ma esistono anche altre relazioni...

Questo ci può risparmiare la fatica di cercare algoritmi inesistenti, e darci metodi per migliorare le versioni naive degli algoritmi...

Tema della lezione III

- Anche il nondeterminismo genera diverse domande:
  - permette di risolvere più problemi?
  - riduce il tempo o lo spazio necessari per la soluzione?
  - in che relazione è con gli algoritmi deterministici che risolvono gli stessi problemi?

  Queste domande generali mirano a capire
  - cosa succederebbe un domani se qualcuno dimostrasse che due classi coincidono (ad es. **P=NP**): che impatto avrebbe – diciamo – sulla sicurezza dei protocolli crittografici?
  - e oggi: dato un problema X che appartiene a una classe nondeterministica (ad es. **NP**) qual è la quantità minima di risorse che un programma "vero" deve usare per risolvere X?

Teoremi di Separazione

Introduzione

- Questa sezione riguarda la prima domanda: quante risorse aggiuntive fanno aumentare i problemi risolubili?
  - tecnicamente bisogna dimostrare che una classe di complessità è strettamente contenuta in un'altra
- Purtroppo nella maggior parte dei casi non sappiamo se le inclusioni sono strette
  - ovvero non abbiamo *risultati di separazione*, come tra **P** e **NP**
- In questa lezione illustriamo i pochi risultati di separazione disponibili, cominciando con il *teorema di gerarchia*:
  - nel modello deterministico, con una quantità logaritmica di tempo aggiuntivo si risolve una classe strettamente più ampia di problemi

Accenneremo anche alle gerarchie di spazio e alle corrispettive nondeterministiche

Introduzione

- Questa sezione riguarda la prima domanda: quante risorse aggiuntive fanno aumentare i problemi risolubili?
  - tecnicamente bisogna dimostrare che una classe di complessità è strettamente contenuta in un'altra
- Purtroppo nella maggior parte dei casi non sappiamo se le inclusioni sono strette
  - ovvero non abbiamo *risultati di separazione*, come tra **P** e **NP**
- In questa lezione illustriamo i pochi risultati di separazione disponibili, cominciando con il *teorema di gerarchia*:
  - nel modello deterministico, con una quantità logaritmica di tempo aggiuntivo si risolve una classe strettamente più ampia di problemi

  Accenneremo anche alle gerarchie di spazio e alle corrispettive nondeterministiche

Introduzione

- Questa sezione riguarda la prima domanda: quante risorse aggiuntive fanno aumentare i problemi risolubili?
  - tecnicamente bisogna dimostrare che una classe di complessità è strettamente contenuta in un'altra
- Purtroppo nella maggior parte dei casi non sappiamo se le inclusioni sono strette
  - ovvero non abbiamo *risultati di separazione*, come tra **P** e **NP**
- In questa lezione illustriamo i pochi risultati di separazione disponibili, cominciando con il *teorema di gerarchia*:
  - nel modello deterministico, con una quantità logaritmica di tempo aggiuntivo si risolve una classe strettamente più ampia di problemi

Accenneremo anche alle gerarchie di spazio e alle corrispettive nondeterministiche

Introduzione

- Questa sezione riguarda la prima domanda: quante risorse aggiuntive fanno aumentare i problemi risolubili?
  - tecnicamente bisogna dimostrare che una classe di complessità è strettamente contenuta in un'altra
- Purtroppo nella maggior parte dei casi non sappiamo se le inclusioni sono strette
  - ovvero non abbiamo *risultati di separazione*, come tra **P** e **NP**
- In questa lezione illustriamo i pochi risultati di separazione disponibili, cominciando con il *teorema di gerarchia*:
  - nel modello deterministico, con una quantità logaritmica di tempo aggiuntivo si risolve una classe strettamente più ampia di problemi

  Accenneremo anche alle gerarchie di spazio e alle corrispettive nondeterministiche

Preliminari

- La tecnica di dimostrazione utilizzata è la *diagonalizzazione*
  - come nella separazione dei linguaggi ricorsivi e ricorsivamente enumerabili (Elem. di Inf. Teorica)
- Questo richiede di trattare le MdT come *strutture dati*
  - rappresentandole con una stringa
  - analogamente a quanto viene fatto nel Teorema di Gödel con gli interi

Preliminari

- La tecnica di dimostrazione utilizzata è la *diagonalizzazione*
  - come nella separazione dei linguaggi ricorsivi e ricorsivamente enumerabili (Elem. di Inf. Teorica)
- Questo richiede di trattare le MdT come *strutture dati*
  - rappresentandole con una stringa
  - analogamente a quanto viene fatto nel Teorema di Gödel con gli interi

Rappresentare le MdT con 0, 1, parentesi e virgole I

- Rappresentiamo stati e simboli dell'alfabeto con interi:
  - $\Sigma \rightsquigarrow \{1, \ldots, |\Sigma|\}$
  - $K \rightsquigarrow \{|\Sigma| + 1, \ldots, |\Sigma| + |K|\}$
  - $\leftarrow, \rightarrow, -, h, "yes", "no" \rightsquigarrow |\Sigma| + |K| + 1, \ldots, |\Sigma| + |K| + 6$
  - lo stato iniziale s è mappato su $|\Sigma| + 1$
- Tutti gli interi sono rappresentati in binario
  - con stringhe di lunghezza $\log \lceil |\Sigma| + |K| + 6 \rceil$
  - denotiamo con $\underline{i}$ la codifica dell'intero i
  - inoltre con $\underline{q}$ la codifica di $q \in K$
  - e con $\underline{\sigma}$ la codifica di $\sigma \in \Sigma$

Rappresentare le MdT con 0, 1, parentesi e virgole II

- Codifica di $(q, \sigma, q', \sigma', D) \in \Delta$:
  - $(\underline{q}, \underline{\sigma}, \underline{q'}, \underline{\sigma'}, \underline{D})$
- Codifica di Δ, denotata con $\underline{\Delta}$:
  - $(\underline{q_1}, \underline{\sigma_1}, \underline{q'_1}, \underline{\sigma'_1}, \underline{D_1}), \ldots, (\underline{q_m}, \underline{\sigma_m}, \underline{q'_m}, \underline{\sigma'_m}, \underline{D_m}) \qquad (m = |\Delta|)$
- Codifica di M, denotata con $\underline{M}$:[1]
  - $\underline{|\Sigma|}, \underline{|K|}, \underline{\Delta}$

[1] Il libro usa M anche per la codifica, per alleggerire la notazione

La MdT universale

- Esiste una MdT "universale" U che data la stringa $\underline{M};x$ simula M sull'input x
  - descriviamo una U a due nastri, tanto sappiamo che si può ridurre a un nastro solo con un rallentamento solo quadratico
  - U mantiene la configurazione di M sul secondo nastro nella forma $(\underline{w,q,u})$. Simula ogni passo di M:
    - cercando la codifica di uno stato q nel 2^o nastro
    - cercando nel 1^o nastro una quintupla $(\underline{q},\underline{\sigma},\ldots)$
    - spostandosi a sinistra sul 2^o nastro per vedere se c'è proprio $\underline{\sigma}$
    - se si, applica le modifiche previste dalla quintupla
    - se no, cerca la prossima quintupla $(\underline{q},\ldots)$ e ripete
  - Se trova un errore nel formato di $\underline{M}$ sposta i cursori a destra all'infinito (diverge)

La MdT universale

- Esiste una MdT "universale" U che data la stringa $\underline{M};x$ simula M sull'input x
  - descriviamo una U a due nastri, tanto sappiamo che si può ridurre a un nastro solo con un rallentamento solo quadratico
  - U mantiene la configurazione di M sul secondo nastro nella forma $(\underline{w,q,u})$. Simula ogni passo di M:
    - cercando la codifica di uno stato q nel 2^o nastro
    - cercando nel 1^o nastro una quintupla $(\underline{q}, \underline{\sigma}, \ldots)$
    - spostandosi a sinistra sul 2^o nastro per vedere se c'è proprio $\underline{\sigma}$
    - se si, applica le modifiche previste dalla quintupla
    - se no, cerca la prossima quintupla $(\underline{q}, \ldots)$ e ripete
  - Se trova un errore nel formato di $\underline{M}$ sposta i cursori a destra all'infinito (diverge)

La MdT universale

- Esiste una MdT "universale" U che data la stringa $\underline{M}; x$ simula M sull'input x
  - descriviamo una U a due nastri, tanto sappiamo che si può ridurre a un nastro solo con un rallentamento solo quadratico
  - U mantiene la configurazione di M sul secondo nastro nella forma $(\underline{w, q, u})$. Simula ogni passo di M:
    - cercando la codifica di uno stato q nel 2^o nastro
    - cercando nel 1^o nastro una quintupla $(\underline{q}, \underline{\sigma}, \ldots)$
    - spostandosi a sinistra sul 2^o nastro per vedere se c'è proprio $\underline{\sigma}$
    - se si, applica le modifiche previste dalla quintupla
    - se no, cerca la prossima quintupla $(\underline{q}, \ldots)$ e ripete
  - Se trova un errore nel formato di $\underline{M}$ sposta i cursori a destra all'infinito (diverge)

La MdT universale

- Esiste una MdT "universale" U che data la stringa $\underline{M}; x$ simula M sull'input x
  - descriviamo una U a due nastri, tanto sappiamo che si può ridurre a un nastro solo con un rallentamento solo quadratico
  - U mantiene la configurazione di M sul secondo nastro nella forma $(\underline{w, q, u})$. Simula ogni passo di M:
    - cercando la codifica di uno stato q nel 2^o nastro
    - cercando nel 1^o nastro una quintupla $(\underline{q, \sigma}, \dots)$
    - spostandosi a sinistra sul 2^o nastro per vedere se c'è proprio $\underline{\sigma}$
    - se si, applica le modifiche previste dalla quintupla
    - se no, cerca la prossima quintupla $(\underline{q}, \dots)$ e ripete
  - Se trova un errore nel formato di $\underline{M}$ sposta i cursori a destra all'infinito (diverge)

La MdT universale

- Esiste una MdT "universale" U che data la stringa $\underline{M};x$ simula M sull'input x
  - descriviamo una U a due nastri, tanto sappiamo che si può ridurre a un nastro solo con un rallentamento solo quadratico
  - U mantiene la configurazione di M sul secondo nastro nella forma $(\underline{w,q,u})$. Simula ogni passo di M:
    - cercando la codifica di uno stato q nel 2^o nastro
    - cercando nel 1^o nastro una quintupla $(\underline{q,\sigma},\ldots)$
    - spostandosi a sinistra sul 2^o nastro per vedere se c'è proprio $\underline{\sigma}$
    - se si, applica le modifiche previste dalla quintupla
    - se no, cerca la prossima quintupla $(\underline{q},\ldots)$ e ripete
  - Se trova un errore nel formato di $\underline{M}$ sposta i cursori a destra all'infinito (diverge)

Il linguaggio usato per la diagonalizzazione

- Sia f una funzione propria
- Problema: Dati M e x, M accetta x entro $f(|x|)$ passi?

$$H_f = \{\underline{M}; x \mid M \text{ accetta } x \text{ entro } f(|x|) \text{ passi}\}$$

Lemma 1 (complessità di H_f: limite superiore)

$H_f \in \mathrm{TIME}(f(n)^3)$

Prova del 1^o lemma su H_f – I

- Descriveremo una MdT a 4 nastri U_f che decide H_f in tempo $f(n)^3$
- Bisogna combinare diverse MdT viste in precedenza:
  - La MdT universale
  - La MdT che simula le MdT multinastro con 1 nastro solo
  - La MdT per il linear speedup, che "toglie" le costanti
  - La MdT M_f che produce $\sqcap^{f(|x|)}$

1. Applicare (un adattamento di) M_f per scrivere $\sqcap^{f(|x|)}$ sul 4^o nastro
   - tempo $O(f(|x|))$

Prova del 1^o lemma su H_f – II

2. U_f copia $\underline{M}$ sul 3^o nastro e trasforma il primo in $\triangleright x$
   - ora la simulazione di M può usare direttamente il 1^o nastro e scrivere solo $\underline{q}$ sul secondo, invece di $(\underline{w, q, u})$

   Inoltre controlla il formato di $\underline{M}$ e rigetta l'input in caso di errore
   - in tutto, tempo $O(f(|x|) + n) = O(f(n))$
3. Se M è multinastro, i nastri vengono tutti rappresentati in sequenza sul 1^o di U_f, come già visto in passato
   - con una passata, U_f raccoglie i simboli correnti e li memorizza sul 2^o nastro (dopo $\underline{q}$)
   - poi cerca sul 3^o nastro una quintupla corrispondente al 2^o nastro ed eventualmente la applica (come in U)
   - ad ogni applicazione, sposta a destra il cursore su $\sqcap^{f(|x|)}$ (4^o nastro) e se arriva alla fine rigetta

Prova del 1^o lemma su H_f – III

- Tempo per simulare 1 transizione di M:
  $O(\ell_M k_M f(|x|)) = O(f(n)^2)$
  - ℓ_M: lunghezza codifica stati e simboli di M
  - k_M: numero nastri di M
  - $f(|x|)$: limite sup. a lunghezza di ciascun nastro di M
  - ℓ_M e k_M sono logaritmici in $\underline{M}$ quindi pessimisticamente, $\ell_M k_M = O(\log^2 n) \Rightarrow \ell_M k_M = O(f(n))$

- L'uso del 4^o nastro garantisce che l'accettazione avviene entro $f(|x|)$ passi simulati di M quindi il costo totale è $O(f(n)^3)$
- Con le modifiche viste nello speedup theorem (memorizzazione a blocchi di m celle) si può "accelerare" U_f portando la computazione a $f(n)^3$ passi al massimo

QED

Seconda proprietà di H_f

$$H_f = \{\underline{M}; x \mid M \text{ accetta } x \text{ entro } f(|x|) \text{ passi}\}$$

Lemma 2 (limite inferiore alla complessità di H_f)

$H_f \notin \mathrm{TIME}(f(\lfloor \frac{n}{2} \rfloor))$

Prova del 2^o lemma su H_f – I

- La dimostrazione è per assurdo: assumiamo che una MdT M_0 decida H_f in tempo $f(\lfloor \frac{n}{2} \rfloor)$ e deriviamo una contraddizione.
- Costruiamo una MdT per la diagonalizzazione, chiamata D_f:

$$D_f(\underline{M}) : \text{if } M_0(\underline{M}, \underline{M}) = \text{"}\mathit{yes}\text{" then "}\mathit{no}\text{" else "}\mathit{yes}\text{"}$$

- Per ipotesi $M_0(\underline{M}, \underline{M})$ termina in tempo $f(\lfloor \frac{2|\underline{M}|+1}{2} \rfloor)$, quindi D_f termina in tempo

$$f\left(\left\lfloor \frac{2n+1}{2} \right\rfloor\right) = f(n)$$

Prova del 2° lemma su H_f – II

- Adesso ci domandiamo: D_f accetta $\underline{D_f}$? ($D_f(\underline{D_f}) =$ "*yes*"?)

se si : significa che $M_0(\underline{D_f}, \underline{D_f}) =$ "*no*", quindi $\underline{D_f}; \underline{D_f} \notin H_f$

- ma per def. di H_f questo significa che D_f non accetta $\underline{D_f}$ in tempo $f(n)$
- tuttavia abbiamo visto che D_f termina sempre in tempo $f(n)$
- quindi l'unica possibilità è che $D_f(\underline{D_f}) =$ "*no*" (assurdo!)

se no : significa che $M_0(\underline{D_f}, \underline{D_f}) =$ "*yes*", quindi $\underline{D_f}; \underline{D_f} \in H_f$

- ma per def. di H_f questo significa che D_f accetta $\underline{D_f}$ in tempo $f(n)$
- quindi $D_f(\underline{D_f}) =$ "*yes*" (assurdo!)

- In ogni caso otteniamo una contraddizione QED

The Time Hierarchy Theorem I

Coi due lemmi precedenti possiamo ora dimostrare che:

Teorema [7.1 del Papadimitriou]

Se $f(n) \geq n$ è una funzione di complessità propria, allora la classe $\text{TIME}(f(n))$ è strettamente contenuta in $\text{TIME}(f(2n+1)^3)$

Prova:

- Poichè f è propria e $f(n) \geq n$, abbiamo

$$f(n) \leq f(2n+1) \leq f(2n+1)^3$$

  pertanto $\text{TIME}(f(n)) \subseteq \text{TIME}(f(2n+1)^3)$

- Per verificare che l'inclusione è stretta, dimostreremo che per una opportuna g, H_g appartiene a $\text{TIME}(f(2n+1)^3)$ ma non a $\text{TIME}(f(n))$

The Time Hierarchy Theorem II

- Scegliamo $g(n) = f(2n+1)$. Notare che

$$2\left\lfloor \frac{n}{2} \right\rfloor + 1 = \left\{ \begin{array}{ll} n+1 & \text{se } n \text{ pari} \\ n & \text{se } n \text{ dispari} \end{array} \right\} \geq n$$

  quindi (poichè f è propria, dunque non decrescente)

$$g(\left\lfloor \frac{n}{2} \right\rfloor) = f(2\left\lfloor \frac{n}{2} \right\rfloor + 1) \geq f(n)$$

  di conseguenza

$$\mathrm{TIME}(g(\left\lfloor \frac{n}{2} \right\rfloor)) \supseteq \mathrm{TIME}(f(n)) . \qquad (1)$$

- Per il Lemma 2, $H_g \notin \mathrm{TIME}(g(\left\lfloor \frac{n}{2} \right\rfloor))$, quindi per (1) $H_g \notin \mathrm{TIME}(f(n))$
- Inoltre per il Lemma 1, $H_g \in \mathrm{TIME}(g(n)^3) = \mathrm{TIME}(f(2n+1)^3)$ QED

Raffinamento del Time Hierarchy Theorem

- Valgono separazioni anche più forti
- È stato dimostrato che

$$\mathrm{TIME}(f(n)) \subset \mathrm{TIME}(f(n)\log^2 f(n))$$

  [vedere problema 7.4.8 nel Papadimitriou]
- Il nostro teorema però è sufficiente per un importante risultato (prossima slide)

Separazione di **P** e **EXP**

- Ricordate che EXP $= \mathrm{TIME}(2^{n^k})$

Teorema

$\mathrm{P} \subset \mathrm{EXP}$

Prova.

- Ogni polinomio in n diventa alla fine più piccolo di 2^n, quindi

$$\mathrm{P} \subseteq \mathrm{TIME}(2^n) \subseteq \mathrm{EXP}$$

  Resta da dimostrare che l'inclusione è stretta.

- Per il Time Hierarchy Theorem,

$$\mathrm{TIME}(2^n) \subset \mathrm{TIME}((2^{2n+1})^3) = \mathrm{TIME}(2^{6n+3}) \subseteq \mathrm{TIME}(2^{n^2}) \subseteq \mathrm{EXP}$$

QED

The Space Hierarchy Theorem

Con due analoghi dei lemmi 1 e 2 si riesce a dimostrare che

Teorema

Se f è propria, allora $\text{SPACE}(f(n)) \subset \text{SPACE}(f(n) \log f(n))$

[Vedere Teorema 7.2 e Problema 7.4.9 nel Papadimitriou]

Gerarchie nondeterministiche

- Alcuni degli hierarchy theorems per il modello nondeterministico:
  - $\mathrm{NTIME}(n^c) \subset \mathrm{NTIME}(n^d)$ per tutte le costanti $1 \leq c < d$
  - se $f(n+1) = O(g(n))$ e $f(n) = o(g(n))$,[2]

$$\mathrm{NSPACE}(f(n)) \subset \mathrm{NSPACE}(g(n))$$

    ad es. per *ogni h crescente* (non importa quanto lentamente):
    - $\mathrm{NSPACE}(\log n) \subset \mathrm{NSPACE}(\log(n) \cdot h(n))$
    - $\mathrm{NSPACE}(n^k) \subset \mathrm{NSPACE}(n^k \cdot h(n))$
    - $\mathrm{NSPACE}(2^n) \subset \mathrm{NSPACE}(2^n \cdot h(n))$
    - $\mathrm{NSPACE}(2^{2^n}) \subset \mathrm{NSPACE}(2^{2^{n+1}})$

Notare come le gerarchie siano molto più fitte che nel modello deterministico

[Dagli articoli citati nella nota 7.4.10 del Papadimitriou]

[2] Per ogni $\epsilon > 0$ esiste $\bar{n}$ such that $\forall n > \bar{n}$, $f(n) \leq \epsilon\, g(n)$

Gerarchie nondeterministiche

- Alcuni degli hierarchy theorems per il modello nondeterministico:
  - $\mathrm{NTIME}(n^c) \subset \mathrm{NTIME}(n^d)$ per tutte le costanti $1 \leq c < d$
  - se $f(n+1) = O(g(n))$ e $f(n) = o(g(n))$,[2]

$$\mathrm{NSPACE}(f(n)) \subset \mathrm{NSPACE}(g(n))$$

ad es. per *ogni h crescente* (non importa quanto lentamente):

- $\mathrm{NSPACE}(\log n) \subset \mathrm{NSPACE}(\log(n) \cdot h(n))$
- $\mathrm{NSPACE}(n^k) \subset \mathrm{NSPACE}(n^k \cdot h(n))$
- $\mathrm{NSPACE}(2^n) \subset \mathrm{NSPACE}(2^n \cdot h(n))$
- $\mathrm{NSPACE}(2^{2^n}) \subset \mathrm{NSPACE}(2^{2^{n+1}})$

Notare come le gerarchie siano molto più fitte che nel modello deterministico

[Dagli articoli citati nella nota 7.4.10 del Papadimitriou]

[2] Per ogni $\epsilon > 0$ esiste $\bar{n}$ such that $\forall n > \bar{n}$, $f(n) \leq \epsilon\, g(n)$

Gerarchie nondeterministiche

- Alcuni degli hierarchy theorems per il modello nondeterministico:
  - $\mathrm{NTIME}(n^c) \subset \mathrm{NTIME}(n^d)$ per tutte le costanti $1 \leq c < d$
  - se $f(n+1) = O(g(n))$ e $f(n) = o(g(n))$,[2]

    $$\mathrm{NSPACE}(f(n)) \subset \mathrm{NSPACE}(g(n))$$

    ad es. per *ogni h crescente* (non importa quanto lentamente):
    - $\mathrm{NSPACE}(\log n) \subset \mathrm{NSPACE}(\log(n) \cdot h(n))$
    - $\mathrm{NSPACE}(n^k) \subset \mathrm{NSPACE}(n^k \cdot h(n))$
    - $\mathrm{NSPACE}(2^n) \subset \mathrm{NSPACE}(2^n \cdot h(n))$
    - $\mathrm{NSPACE}(2^{2^n}) \subset \mathrm{NSPACE}(2^{2^{n+1}})$

Notare come le gerarchie siano molto più fitte che nel modello deterministico

[Dagli articoli citati nella nota 7.4.10 del Papadimitriou]

[2] Per ogni $\epsilon > 0$ esiste $\bar{n}$ such that $\forall n > \bar{n}$, $f(n) \leq \epsilon\, g(n)$

Gerarchie nondeterministiche

- Alcuni degli hierarchy theorems per il modello nondeterministico:
  - $\text{NTIME}(n^c) \subset \text{NTIME}(n^d)$ per tutte le costanti $1 \le c < d$
  - se $f(n+1) = O(g(n))$ e $f(n) = o(g(n))$,[2]

    $$\text{NSPACE}(f(n)) \subset \text{NSPACE}(g(n))$$

    ad es. per *ogni h crescente* (non importa quanto lentamente):
    - $\text{NSPACE}(\log n) \subset \text{NSPACE}(\log(n) \cdot h(n))$
    - $\text{NSPACE}(n^k) \subset \text{NSPACE}(n^k \cdot h(n))$
    - $\text{NSPACE}(2^n) \subset \text{NSPACE}(2^n \cdot h(n))$
    - $\text{NSPACE}(2^{2^n}) \subset \text{NSPACE}(2^{2^{n+1}})$

  Notare come le gerarchie siano molto più fitte che nel modello deterministico
  [Dagli articoli citati nella nota 7.4.10 del Papadimitriou]

[2] Per ogni $\epsilon > 0$ esiste $\bar{n}$ such that $\forall n > \bar{n}$, $f(n) \le \epsilon\, g(n)$

Relazioni tra Spazio e Tempo

Obbiettivi

- Date informazioni sullo spazio (rispettivamente il tempo) necessario per risolvere un problema...
- ...determinare quanto tempo (rispettivamente spazio) sarà necessario
- Tra le applicazioni:
    - riconoscere soluzioni (o stime di complessità) errate, dove spazio e tempo "non corrispondono"
    - guidare lo sviluppo di algoritmi, evitando di cercare soluzioni impossibili

Obbiettivi

- Date informazioni sullo spazio (rispettivamente il tempo) necessario per risolvere un problema...
- ...determinare quanto tempo (rispettivamente spazio) sarà necessario
- Tra le applicazioni:
  - riconoscere soluzioni (o stime di complessità) errate, dove spazio e tempo "non corrispondono"
  - guidare lo sviluppo di algoritmi, evitando di cercare soluzioni impossibili

Obbiettivi

- Date informazioni sullo spazio (rispettivamente il tempo) necessario per risolvere un problema...
- ...determinare quanto tempo (rispettivamente spazio) sarà necessario
- Tra le applicazioni:
  - riconoscere soluzioni (o stime di complessità) errate, dove spazio e tempo "non corrispondono"
  - guidare lo sviluppo di algoritmi, evitando di cercare soluzioni impossibili

Limiti di tempo dato lo spazio

- Ricordate che per ogni cella di nastro utilizzata nella computazione c'è voluto un passo di computazione per "prenderla"
  - muovendo la testina oltre il limite del nastro
  - l'equivalente di "new" o "malloc" nelle MdT
- Ciò significa che lo spazio utilizzato è sempre minore o uguale al tempo di computazione
- Quindi ogni problema risolubile in tempo $f(n)$ è anche risolubile in spazio $O(f(n))$, da cui

$$\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

- Ma esistono limiti più stretti...

Limiti di tempo dato lo spazio

- Ricordate che per ogni cella di nastro utilizzata nella computazione c'è voluto un passo di computazione per "prenderla"
  - muovendo la testina oltre il limite del nastro
  - l'equivalente di "new" o "malloc" nelle MdT
- Ciò significa che lo spazio utilizzato è sempre minore o uguale al tempo di computazione
- Quindi ogni problema risolubile in tempo $f(n)$ è anche risolubile in spazio $O(f(n))$, da cui

$$\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

- Ma esistono limiti più stretti...

Limiti di tempo dato lo spazio

- Ricordate che per ogni cella di nastro utilizzata nella computazione c'è voluto un passo di computazione per "prenderla"
  - muovendo la testina oltre il limite del nastro
  - l'equivalente di "new" o "malloc" nelle MdT
- Ciò significa che lo spazio utilizzato è sempre minore o uguale al tempo di computazione
- Quindi ogni problema risolubile in tempo $f(n)$ è anche risolubile in spazio $O(f(n))$, da cui

$$\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$$

- Ma esistono limiti più stretti...

Limiti di tempo dato lo spazio

- Nel 1975 Hopcroft, Paul e Valiant hanno dimostrato che

Teorema

$$\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}\left(\frac{f(n)}{\log f(n)}\right)$$

[Vedere Problema 7.4.17 nel Papadimitriou]

Esempi

- Se un vettore può essere ordinato in tempo $O(n \log n)$, allora lo spazio necessario per l'algoritmo è al massimo $O(n)$

$$\frac{n \log n}{\log(n \log n)} = \frac{n \log n}{\log n + \log\log n} = \frac{n}{1 + \log\log n / \log n} \leq n$$

  - altrimenti l'algoritmo sicuramente non è ottimale: spreca memoria

- Se un problema non può essere risolto in spazio polinomiale, allora ci si può scordare di risolverlo con un algoritmo che gira in tempo polinomiale
  - ad es. verificare l'equivalenza di due espressioni regolari

Esempi

- Se un vettore può essere ordinato in tempo $O(n \log n)$, allora lo spazio necessario per l'algoritmo è al massimo $O(n)$

$$\frac{n \log n}{\log(n \log n)} = \frac{n \log n}{\log n + \log \log n} = \frac{n}{1 + \log \log n / \log n} \leq n$$

    - altrimenti l'algoritmo sicuramente non è ottimale: spreca memoria

- Se un problema non può essere risolto in spazio polinomiale, allora ci si può scordare di risolverlo con un algoritmo che gira in tempo polinomiale
    - ad es. verificare l'equivalenza di due espressioni regolari

Limiti di spazio dato il tempo

- Ricordate le MdT space bounded: se ne può forzare la terminazione calcolando quante configurazioni diverse possono avere
  - se M passa una seconda volta per la stessa configurazione va in ciclo
  - la variante M' che termina va sempre in "no" dopo un numero di passi pari al numero di configurazioni
- Il numero di configurazioni è esponenziale rispetto allo spazio massimo: $O((2|\Sigma|)^{f(n)})$
- Quindi, nell'ipotesi peggiore, il tempo necessario per risolvere un problema space-bounded è esponenziale nello spazio utilizzato

Teorema

$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Limiti di spazio dato il tempo

- Ricordate le MdT space bounded: se ne può forzare la terminazione calcolando quante configurazioni diverse possono avere
  - se M passa una seconda volta per la stessa configurazione va in ciclo
  - la variante M' che termina va sempre in "no" dopo un numero di passi pari al numero di configurazioni
- Il numero di configurazioni è esponenziale rispetto allo spazio massimo: $O((2|\Sigma|)^{f(n)})$
- Quindi, nell'ipotesi peggiore, il tempo necessario per risolvere un problema space-bounded è esponenziale nello spazio utilizzato

Teorema

$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Limiti di spazio dato il tempo

- Ricordate le MdT space bounded: se ne può forzare la terminazione calcolando quante configurazioni diverse possono avere
  - se M passa una seconda volta per la stessa configurazione va in ciclo
  - la variante M' che termina va sempre in "no" dopo un numero di passi pari al numero di configurazioni
- Il numero di configurazioni è esponenziale rispetto allo spazio massimo: $O((2|\Sigma|)^{f(n)})$
- Quindi, nell'ipotesi peggiore, il tempo necessario per risolvere un problema space-bounded è esponenziale nello spazio utilizzato

Teorema

$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Limiti di spazio dato il tempo

- Ricordate le MdT space bounded: se ne può forzare la terminazione calcolando quante configurazioni diverse possono avere
  - se M passa una seconda volta per la stessa configurazione va in ciclo
  - la variante M' che termina va sempre in "no" dopo un numero di passi pari al numero di configurazioni
- Il numero di configurazioni è esponenziale rispetto allo spazio massimo: $O((2|\Sigma|)^{f(n)})$
- Quindi, nell'ipotesi peggiore, il tempo necessario per risolvere un problema space-bounded è esponenziale nello spazio utilizzato

Teorema

$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Limiti di spazio dato il tempo

- Ricordate le MdT space bounded: se ne può forzare la terminazione calcolando quante configurazioni diverse possono avere
  - se M passa una seconda volta per la stessa configurazione va in ciclo
  - la variante M' che termina va sempre in "no" dopo un numero di passi pari al numero di configurazioni
- Il numero di configurazioni è esponenziale rispetto allo spazio massimo: $O((2|\Sigma|)^{f(n)})$
- Quindi, nell'ipotesi peggiore, il tempo necessario per risolvere un problema space-bounded è esponenziale nello spazio utilizzato

Teorema

$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Discussione

- In particolare, quando $f(n)$ è un polinomio, questo teorema implica

$$\text{PSPACE} \subseteq \text{EXP}$$

- Non si sa se PSPACE $\subset$ EXP
- Molti pensano di si perchè
  - PSPACE può usare solo spazio polinomiale
  - EXP può anche usare spazio esponenziale, grazie alla relazione

$$\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n))$$

Relazioni tra classi di complessità deterministiche e nondeterministiche

Applicazioni

- Con queste analisi si può valutare, ad esempio
  - quanto ci costa oggi rimpiazzare una soluzione nondeterministica con una deterministica
  - che succederebbe se fosse **P=NP**...
  - ...e più in generale cosa succederebbe se certe classi collassassero
  - con impatto - ad esempio - sulla crittografia
  - ...

Le proprietà più semplici

- Notate che le MdT deterministiche sono casi particolari di MdT nondeterministiche
  - dove capita che Δ associ un'unica azione ad ogni coppia (stato,simbolo)

Ne segue immediatamente che

Proposizione [Teorema 7.4 del Papadimitriou, punto (a)]

$$\begin{aligned} \mathrm{TIME}(f(n)) &\subseteq \mathrm{NTIME}(f(n)) \\ \mathrm{SPACE}(f(n)) &\subseteq \mathrm{NSPACE}(f(n)) \end{aligned}$$

- Come ci si aspetta, il nondeterminismo non riduce la classe di problemi risolubili, può solo (eventualmente) estenderla

Da nondeterministico a deterministico

- Abbiamo già dimostrato che $\mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$
- Dimostreremo i seguenti risultati

Teorema [7.4 del Papadimitriou, punti (b) e (c)]

1. $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$
2. $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log(n)+f(n)})$

- Confrontare 1 con l'inclusione in alto e 2 con

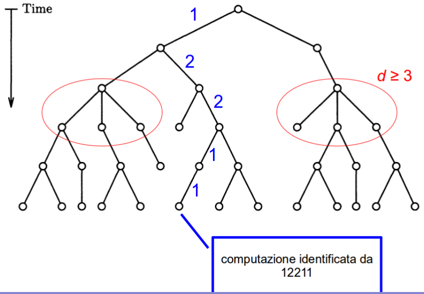
$$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(k^{f(n)})$$

Ciò <u>sembra</u> indicare che il nondeterminismo influenza il tempo, ma poco o nulla lo spazio

Teorema 7.4 – Prova che NTIME($f(n)$) $\subseteq$ SPACE($f(n)$)

Cenni (assomiglia alla prova del Teorema 2.6)

- Stessa simulazione delle MdT nondeterministiche in tempo esponenziale
- I possibili run di qualunque MdT nondeterministica M vengono rappresentati con sequenze di interi compresi tra 1 e d (d grado di nondeterminismo di M)



Teorema 7.4 – Prova che $\mathsf{NTIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$

Cenni (assomiglia alla prova del Teorema 2.6)

- Se M gira in tempo $f(n)$, allora ogni ramo è lungo al più $f(n)$ passi
- Quindi anche lo spazio utilizzato in ogni run è limitato da $f(n)$
  - serve almeno un passo per ogni nuova cella occupata
- Nella simulazione deterministica bisogna provare tutti i run (rami) per vedere se almeno uno è di successo (come nel Teorema 2.6)
- Basta eseguirli uno alla volta, riutilizzando alla fine lo spazio per i tentativi successivi
  - come in una esplorazione depth-first

Teorema 7.4 – Prova che $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$

Cenni (assomiglia alla prova del Teorema 2.6)

- Lo spazio massimo utilizzato è la somma di quello utilizzato da M più quello ausiliario richiesto per enumerare i run
- Lo spazio usato da M in ogni run è $O(f(n))$ (vedi slide precedente)
- Lo spazio aggiuntivo che serve per memorizzare le sequenze di interi corrispondenti ai run nondeterministici è $O(f(n) \log d) = O(f(n))$
  - per via dei limiti sulla lunghezza dei run e il fatto che d è costante
- Quindi lo spazio totale utilizzato nella simulazione deterministica è $O(f(n))$

QED

Teorema 7.4 – Prova che $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log(n)+f(n)})$

The *reachability method*

- Questa prova utilizza una strategia di dimostrazione chiamata *reachability method* applicata in diversi teoremi
  - sfrutta l'algoritmo polinomiale per REACHABILITY
  - applicandola al grafo delle configurazioni della MdT M data
  - concettualmente simile a vedere le computazioni di M come un albero (vedi prova precedente)
  - ma questa volta ogni configurazione corrisponde a 1 solo nodo
- Sia data una MdT nondeterministica M a k nastri che riconosce un linguaggio L in spazio $f(n)$. Dobbiamo mostrare come simularla deterministicamente in tempo $k^{\log(n)+f(n)}$

Teorema 7.4 – Prova che NSPACE$(f(n)) \subseteq$ TIME$(k^{\log(n)+f(n)})$

The *reachability method*

- Prima ricordiamo come sono fatte le configurazioni di M: uno "snapshot" dello stato della MdT in un singolo passo della computazione

$$(q, w_1, u_1, \ldots, w_k, u_k)$$

  dove q è lo stato corrente, e ogni $w_i u_i$ rappresenta l'i-esimo nastro (w_i è la parte sinistra, fino alla testina, u_i quello che segue a destra)

- M è una MdT con *input e output* che deve solo riconoscere L, quindi
  - $w_1 u_1 = \triangleright x$ (dove x è l'input dato)
  - $w_k u_k$ è inutilizzato (basta terminare in "yes" o "no")

Teorema 7.4 – Prova che NSPACE$(f(n)) \subseteq$ TIME$(k^{\log(n)+f(n)})$

The *reachability method*

- Queste osservazioni ci permettono di ottimizzare la rappresentazione delle configurazioni:

$$(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$$

  dove $i \leq |x| = n$ è la posizione della testina sul nastro di input
- Inoltre i nastri $2, \ldots, k-1$ sono lunghi al più $f(n)$
  - data l'ipotesi che il tempo utilizzato è $f(n)$
- Quindi il numero di configurazioni diverse è

$$|K| \cdot (n+1) \cdot |\Sigma|^{(2k-2)f(n)} = O(nc_0^{f(n)}) = O(c_1^{\log n + f(n)})$$

  dove c_1 dipende solo da M (non da x)

Teorema 7.4 – Prova che $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log(n)+f(n)})$

The *reachability method*

- Adesso definiamo il grafo delle configurazioni, con $c_1^{\log n + f(n)}$ nodi
- Denotato con $G(M, x)$
  - nodi = configurazioni ottimizzate
  - archi: (C_1, C_2) tali che $C_1 \stackrel{M}{\to} C_2$
- Chiaramente: $x \in L$
  sse $M(x) = \mathit{yes}$
  sse c'è una computazione $C_0 \stackrel{M}{\to} C_1 \stackrel{M}{\to} \cdots \stackrel{M}{\to} C_m$ che accetta
  sse c'è un cammino in $G(M, x)$ tra la configurazione iniziale C_0 e una configurazione finale che accetta
  cioè REACHABILITY su di un grafo di $c_1^{\log n + f(n)}$ nodi

Teorema 7.4 – Prova che NSPACE($f(n)$) $\subseteq$ TIME($k^{\log(n)+f(n)}$)

The *reachability method*

- Ricordate che REACHABILITY si risolve in tempo polinomiale
  - per eccesso: $O(n^2)$
- Quindi (per qualche c_2) possiamo decidere se $x \in L$ in tempo

$$c_2(c_1^{\log n + f(n)})^2 = c_2 c_1^{2(\log n + f(n))} = c_2(c_1^2)^{\log n + f(n)} = O(c^{\log n + f(n)})$$

Teorema 7.4 – Prova che $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log(n)+f(n)})$

The *reachability method*

- Mancano solo pochi dettagli: come generare $G(M, x)$?
- **Metodo 1**: Costruire esplicitamente la matrice di adiacenza di $G(M, x)$ su un apposito nastro, poi lanciare l'algoritmo per REACHABILITY su quel nastro
  - spazio richiesto esponenziale – non necessario
- **Metodo 2**: Evitare la memorizzazione esplicita di $G(M, x)$:
  - quando serve sapere se (C_1, C_2) è un arco
  - simuliamo un passo di M per vedere se $C_1 \xrightarrow{M} C_2$
  - C_1 determina completamente i possibili C_2

QED

Esercizio: Relazioni derivate dal Teorema 7.4

Usando le proprietà appena dimostrate, provate che

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$$

Influenza del nondeterminismo sullo spazio: il teorema di Savitch

Motivazioni

- Abbiamo già avuto l'impressione che il nondeterminismo influenzi il consumo di memoria poco o nulla
- Tuttavia le relazioni tra SPACE e NSPACE non sono ancora complete
- Sappiamo che SPACE$(f(n)) \subseteq$ NSPACE$(f(n))$ ma ci mancano inclusioni inverse
  - quanto spazio in più ci vuole per simulare un algoritmo nondeterministico?
- Ricordate che passando da soluzioni nondeterministiche a deterministiche il tempo può aumentare esponenzialmente (per quanto ne sappiamo oggi)
- Invece possiamo dimostrare che lo *spazio* richiesto aumenta di poco (quadraticamente)

Strategia

- Usare il *reachability method*
- insieme a una soluzione ottimizzata di REACHABILITY
- che fa uso di spazio $\log^2 n$
- A questo ci serve il Teorema di Savitch:

Teorema di Savitch [7.5 nel Papadimitriou]

REACHABILITY $\in \mathsf{SPACE}(\log^2 n)$

Prova del teorema di Savitch – I

- Dati:
  - un grafo G con n nodi
  - due nodi x e y
  - un intero $i \geq 0$

  scriviamo $PATH(x, y, i)$ sse G contiene un cammino da x a y di lunghezza $\leq 2^i$

- Quindi, visto che i nodi sono n, risolvere REACHABILITY equivale a calcolare se $PATH(x, y, \lceil \log n \rceil)$ vale, per qualunque coppia di nodi x e y

Prova del teorema di Savitch – II

- A questo scopo definiamo una MdT M con un nastro di input e due ausiliari che calcola $PATH(x, y, i)$
  - vista come funzione booleana (predicato)
- Sul nastro di input (read only) scriviamo la matrice di adiacenza di G
- Il 2^o nastro contiene (x, y, i) (interi codificati in binario)

Prova del teorema di Savitch – III

- L'algoritmo è ricorsivo:
  1. se $i = 0$ si verifica se esiste un cammino lungo $2^0 = 1$ scandendo la matrice di G alla ricerca di un arco (x, y)
  2. se $i > 0$ si verifica ricorsivamente per ogni nodo z se valgono sia $PATH(x, z, i - 1)$ sia $PATH(z, y, i - 1)$
     - concatenandoli si ottiene un cammino da x a y di lunghezza massima $2 \cdot 2^{i-1} = 2^i$

- La ricorsione è implementata "a mano", usando il 2^o nastro come stack di attivazione

Prova del teorema di Savitch – IV

- Per mantenere i limiti di spazio desiderati, consideriamo un z alla volta e riutilizziamo lo spazio per i successivi
- Per ogni z appendiamo $(x, z, i-1)$ al 2^o nastro (che funge da stack di attivazione)
- Risolviamo ricorsivamente $PATH(x, z, i-1)$
- se la risposta è negativa, togliamo $(x, z, i-1)$ dal 2^o nastro e passiamo al z successivo
- altrimenti sovrascriviamo $(x, z, i-1)$ con $(z, y, i-1)$ e risolviamo ricorsivamente $PATH(z, y, i-1)$
- se la risposta è negativa, eliminiamo $(z, y, i-1)$ dallo "stack" e passiamo al z successivo
- altrimenti eliminiamo $(z, y, i-1)$ dallo "stack" e restituiamo una risposta positiva a $PATH(x, y, i)$

Prova del teorema di Savitch – V

- Stima dello spazio richiesto:
  - il 2^o nastro contiene al massimo $\lceil \log n \rceil$ triple,
    - perchè a ogni livello di ricorsione si dimezza la lunghezza massima dei cammini cercati, e la lunghezza massima totale è proprio n

    ciascuna lunga $3 \log n$ simboli
  - il 3^o nastro è un contatore da 1 a n che serve per enumerare i nodi z – lo spazio richiesto è $\log n$
- Spazio totale:

$$\lceil \log n \rceil \cdot 3 \log n + \log n = O(\log^2 n)$$

QED

Spazio min. richiesto per simulare una MdT nondeterministica

Corollario del teorema di Savitch

Per ogni funzione di complessità propria $f(n) \geq \log n$,

$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$$

Prova del corollario

- Stessa costruzione del Teorema 7.5, basata su reachability
  - si risolve reachability sul grafo delle configurazioni
  - usando l'algoritmo di Savitch
  - e generando implicitamente il grafo delle configurazioni per risparimiare spazio

- Spazio richiesto:

$$\log^2(c^{f(n)}) = (\log(c^{f(n)}))^2 = O(f(n)^2) = O(f^2(n))$$

QED

Altro corollario fondamentale

- Da SPACE($f(n)$) $\subseteq$ NSPACE($f(n)$) e NSPACE($f(n)$) $\subseteq$ SPACE($f^2(n)$), quando f è un polinomio, segue immediatamente che:

Corollario

PSPACE = NPSPACE

- Ovvero il nondeterminismo non estende la classe di problemi che si possono risolvere in spazio polinomiale
- Questo rafforza l'impressione che il nondeterminismo influenzi poco o nulla lo spazio

Capitolo di riferimento

Papadimitriou

- Parte 3, Capitolo 7, paragrafo 2