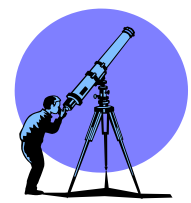
teoria della complessità: la risorsa



spazio

teoria della complessità

- obiettivo: classificare i problemi dal punto di vista delle risorse di calcolo che richiedono
- identificazione di classi di complessità
- risorsa: spazio (memoria)
- modello di calcolo: macchina di Turing; ipotesi di notevole generalità

$s_A(n)$ spazio (numero di celle) usato per l'esecuzione dell'algoritmo A sull'input x con $|x|=n$ nel caso peggiore

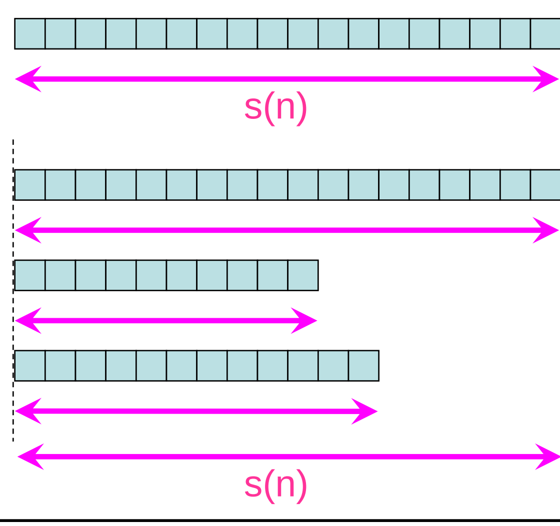
complessità e problemi di decisione su linguaggi

data una funzione $f:N \rightarrow N$ possiamo definire le seguenti classi di linguaggi con riferimento a MT ad un nastro

DSPACE(f(n)) insieme dei linguaggi decisi da una MT deterministica in spazio al più O(f(n))

NSPACE(f(n)) insieme dei linguaggi decisi da una MT non deterministica in spazio al più O(f(n))

spazio deterministico e spazio non deterministico



SAT è risolvibile in spazio lineare deterministico

- teorema: SAT$\in$DSPACE(n)
- dimostrazione:
  - sia n la lunghezza dell'input; le variabili sono quindi al più n
  - generiamo sul nastro, in iterazioni successive, tutte le possibili assegnazioni vero/falso delle variabili
  - in ogni iterazione valutiamo se la formula è vera e cancelliamo il nastro, per riusarlo nell'iterazione successiva

relazioni elementari tra classi di complessità

teorema: per ogni f: N$\rightarrow$N con f(n)$\geq$n

DTIME(f(n))$\subseteq$NTIME(f(n)) e

DSPACE(f(n))$\subseteq$NSPACE(f(n))

dimostrazione: una MT è una MTND

teorema: per ogni f: N$\rightarrow$N con f(n)$\geq$n

DTIME(f(n))$\subseteq$DSPACE(f(n)) e

NTIME(f(n))$\subseteq$NSPACE(f(n))

dimostrazione: in t passi di computazione una MT può usare al più t celle di nastro

relazioni elementari tra classi di complessità

teorema: per ogni f: $N \rightarrow N$ con $f(n) \geq n$, $NTIME(f(n)) \subseteq DSPACE(f(n))$

dimostrazione:

data una MTND M che decide il linguaggio L in tempo f(n) costruiamo una MT deterministica M' che decide L usando spazio O(f(n))

M' simula M esplorando l'albero di computazione di M' ed eseguendo ad uno ad uno tutti i cammini radice-foglia dell'albero (ognuno di al più f(n) passi)

per ogni cammino, M' usa al più O(f(n)) celle di nastro

tali celle vengono cancellate alla fine di ogni cammino e riutilizzate per il cammino successivo

teorema di SAVITCH

teorema: per ogni f: $N \rightarrow N$ con $f(n) \geq n$, $NSPACE(f(n)) \subseteq DSPACE(f^2(n))$

dimostrazione:

- simuliamo deterministicamente la MTND che usa spazio f(n)
- una strategia troppo semplice non funziona
  - infatti potremmo provare ad una ad una tutte le computazioni non deterministiche
  - però in questo caso in ogni computazione dobbiamo ricordare quale essa sia, per poter passare alla successiva

teorema di SAVITCH

dimostrazione:

- ma una computazione che usa spazio O(f(n)) può attraversare $2^{O(f(n))}$ diverse configurazioni, facendo una scelta non deterministica per ciascuna di esse
- questo potrebbe portarci a dover usare spazio $2^{O(f(n))}$; molto di più di quanto dica il teorema

- quindi cambiamo strategia, studiando un problema correlato

lo yieldability problem

- sono date due configurazioni c_1 e c_2 di una MTND N ed un numero T
- vogliamo verificare se N può raggiungere c_2 da c_1 in al più T passi
- lo yieldability problem può essere risolto da una MT deterministica
  - si prova con ogni configurazione intermedia c_m e si verifica ricorsivamente se da c_1 si può raggiungere c_m e da c_m si può raggiungere c_2, in entrambi i casi con al più T/2 passi
  - il secondo test può riusare lo spazio del primo

lo yieldability problem

- attenzione: c'è bisogno di spazio ulteriore per memorizzare lo stack della ricorsione
- ogni livello della ricorsione richiede spazio O(f(n)) per memorizzare la configurazione
- la profondità della ricorsione è log(T), dove T è il tempo impiegato dalla computazione più profonda della MTND
- abbiamo che $T=2^{O(f(n))}$, quindi $\log(T)=O(f(n))$
- quindi la computazione determinsitica usa spazio $O(f^2(n))$

teorema di SAVITCH

dimostrazione:

- sia N una MTND che decide il linguaggio A con spazio O(f(n))
- costruiamo una MT M che decide A
- M usa una MT "canyield" che verifica se da una configurazione se ne può raggiungere un'altra in un numero di passi prefissato

teorema di SAVITCH

dimostrazione:

- data la stringa w di input, con |w|=n, le configurazioni c_1 e c_2 di N su w e T, la MT canyield(c_1,c_2,T) accetta se, quando N parte dalla configurazione c_1, la stessa N ha una qualche computazione che la porta in c_2 in al più T passi; altrimenti rifiuta

teorema di SAVITCH

dimostrazione:

- canyield(c_1,c_2,T)
  - se T=1 verifica se c_1=c_2 o se c_2 è raggiungibile in un passo; se si accetta, altrimenti rifiuta
  - se T>1, per ogni configurazione c_m di N su w, che usa spazio f(n)
    - esegui canyield(c_1,c_m,T/2), esegui canyield(c_m,c_2,T/2)
    - se entrambe accettano allora accetta, altrimenti rifiuta

teorema di SAVITCH

dimostrazione:

- costruiamo M come segue
- modifichiamo N in modo che abbia una sola configurazione di accettazione c_F
- sia c_0 la configurazione iniziale di N su w
- scegliamo una costante d tale che N non abbia più di $2^{df(n)}$ configurazioni usando nastro f(n)
- osserva: $2^{df(n)}$ limita superiormente il tempo impiegato da ogni branch di N

teorema di SAVITCH

dimostrazione:

- M restituisce il risultato di canyield(c_0,c_F,$2^{df(n)}$)
- la simulazione è chiaramente corretta
- ogni volta che canyield viene invocata memorizza c_1 e c_2 e T sul nastro in uno stack
- ogni livello di ricorsione usa spazio addizionale O(f(n))

teorema di SAVITCH

dimostrazione:

- inizialmente $T=2^{df(n)}$
- ogni livello di ricorsione divide T per 2
- la profondità della ricorsione è $O(\log(2^{df(n)}))=O(f(n))$
- lo spazio usato è effettivamente $(f^2(n))$

teorema di SAVITCH

dimostrazione:

- una questione sottile
- quando M esegue canyield deve conoscere f(n)
- per risolvere il problema possiamo modificare M in modo che tenti i valori di f(n) ad uno ad uno: 1,2,3,4,5,7,8,....
- la macchina smette di tentare quando non raggiunge nessuna configurazione di una certa taglia

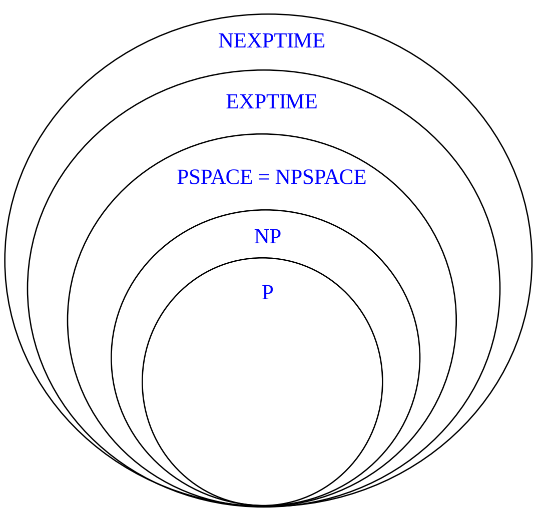
la classe PSPACE

$$\mathbf{PSPACE} = \bigcup_{k=0}^{\infty} \mathrm{DSPACE}(n^k)$$

il teorema di Savitch implica che:

PSPACE=NPSPACE

relazioni tra classi di complessità



inclusioni strette accertate:

$P \subset$ EXPTIME
$NP \subset$ NEXPTIME

un problema ancora aperto:

P = NP ?

ci si convince sempre di più che non sia così, ma nessuno lo ha mai dimostrato

PSPACE-completezza

- anche per la classe PSPACE è possibile definire una completezza, con l'obiettivo di identificare i problemi più difficili della classe
- anche in questo caso le riduzioni usate sono riduzioni che impiegano tempo polinomiale
- perché non usare riduzioni che usano spazio polinomiale?

PSPACE-completezza

- regola generale per definire la completezza rispetto ad una classe:
  - quando si definiscono i problemi completi rispetto ad una classe il modello di calcolo usato per la riduzione deve essere meno potente di quello usato per definire la stessa classe

PSPACE-completezza

- un esempio di problema PSPACE-completo:
  - TQBF = {<Φ>| Φ è una formula booleana pienamente quantificata}
  - formula boolenana con quantificatori (esistenziali o universali) per ogni variabile