LA TEORIA DELLA COMPLESSITÀ COMPUTAZIONALE

INTRODUZIONE

OBIETTIVO: classificare gli algoritmi a seconda delle risorse utilizzate

- risorse necessarie (lower bound)
- risorse sufficienti (upper bound)

Aspetti considerati:

- tempo di esecuzione
- occupazione di memoria

Dato un problema, se $\exists$ un algoritmo efficiente che lo risolve ➔ PROBLEMA TRATTABILE

TESI DI CHURCH

I problemi trattabili sono i problemi risolubili in tempo polinomiale (nella lunghezza dell'input) da una MTD.

I problemi risolvibili mediante un algoritmo con tempo polinomiale appartengono alla classe P

TRATTABILITÀ DELLA CLASSE P

Motivi:

- Esistono pochi problemi pratici che richiedono un tempo polinomiale $O(n^k)$ con grado k molto alto

- Un problema risolto in tempo polinomiale in un modello computazionale può essere risolto ancora in tempo polinomiale con un altro modello

- La classe P ha delle interessanti proprietà di chiusura, rispetto alle operazioni +, x e o

PROBLEMA

Un problema astratto è una relazione Q:I→S

I = insieme delle istanze

S = insieme delle soluzioni

Esempio

Problema: dato un numero x calcolare $f(x)=x^2+1$

Istanza: dato x=2 calcolare f(x) f(2)=5

TIPI DI PROBLEMI

- **problemi di decisione**, in cui data un'istanza x $\in$ I restituisce un valore del tipo vero/falso
- **problemi di ottimizzazione**, in cui data un'istanza x restituisce y $\in$ sol(x) che assume il valore massimo (o minimo) fra tutte le soluzioni possibili
- **problemi di ricerca**, in cui data un'istanza x restituisce una soluzione y $\in$ sol(x)
- **problemi di enumerazione**, in cui data un'istanza restituisce |sol(x)|

CONSIDERAZIONI

Per l'analisi della complessità computazionale ci si può concentrare sui problemi di decisione in quanto:

- Richiedono l'introduzione di un minor numero di concetti
- I problemi di decisione non sono più difficili degli altri
- Altri tipi di problemi possono essere ricondotti a problemi di decisione

CODIFICA

Un problema è risolto tramite un algoritmo

→ le sue istanze devono essere rappresentate in modo accettabile per l'algoritmo

Una codifica è una corrispondenza e:S→C

S = insieme di oggetti astratti

C = insieme di stringhe su un certo alfabeto

esempio (problema di decisione) e:S→{0,1}*

PROBLEMA CONCRETO

codifica

Pb astratto ==========> Pb concreto

Un problema concreto è risolvibile in tempo polinomiale se

$\exists$ un algoritmo che lo risolve in tempo $O(n^k)$ per

qualche costante k

CODIFICHE CORRELATE

Dato un insieme I di istanze di un problema, si dice che due codifiche e_1 ed e_2 sono correlate polinomialmente se esistono due funzioni calcolabili in tempo polinomiale tale che

$f_{12}(e_1(i))=e_2(i)$, $f_{21}(e_2(i))=e_1(i)$ $\forall$ elemento i di I

la codifica $e_2(i)$ può essere calcolata dalla codifica $e_1(i)$ attraverso un algoritmo di tempo polinomiale

INDIPENDENZA DALLA CODIFICA

Teorema

Sia Q un problema di decisione astratto su un insieme I di istanze, e siano e_1 ed e_2 due codifiche di I correlate polinomialmente. Allora

$$e_1(Q) \in P \Longleftrightarrow e_2(Q) \in P$$

DIMOSTRAZIONE

Ipotesi: $e_1(Q)$ risolto in tempo $O(n^k)$ e la codifica $e_1(i)$ è calcolata da $e_2(i)$ in tempo $O(n^c)$, $n=|e_1(i)|$

Per il calcolo di $e_2(Q)$, dato i si calcola prima $e_1(i)$ e quindi si esegue l'algoritmo per $e_1(Q)$ su $e_1(i)$.
La conversione delle codifiche impiega tempo $O(n^c)$, $|e_1(i)| = O(n^c)$, poiché l'output di un esecutore non può essere più lungo del suo tempo di esecuzione.
Per risolvere il problema con $e_1(i)$ si impiega un tempo $O(|e_1(i)|^k) = O(n^{ck})$ (tempo polinomiale)

DA CONCRETO AD ASTRATTO

Cerchiamo di estendere la definizione di problema risolvibile in tempo polinomiale dal pb concreto al pb astratto usando la codifica.

L'efficienza nella risoluzione di un problema non dovrebbe dipendere da come il problema è codificato.

In realtà, l'efficienza dipende strettamente dalla codifica.

ESEMPIO

PROCEDURE(K)
0 while (k>0)
1 do k←k-1 } O(k)

Considero k=8

k	$e_1(Q)$ codifica unaria	$e_2(Q)$ codifica binaria
	1 1 1 1 1 1 1 1	1 0 0 0
	1 1 1 1 1 1 1	0 1 1 1
	1 1 1 1 1 1	0 1 1 0
	1 1 1 1 1	0 0 1 1
	. . . .	. . . .
	1	0 0 0 1

Dimensione input $\|e_{1/2}(i)\|$	k	tetto(lg k)
Tempo di esecuzione	$O(\|e_1(i)\|)$	$O(2^{\wedge}(\|e_2(i)\|))$

ESEMPIO (1)

input	codifica unaria	codifica decimale
k=1	1	1
5	1 1 1 1 1	5
10	1 1 … … 1 1	10

codifica unaria: la dimensione è polinomiale con l'input

codifica decimale: la dimensione è logaritmica con l'input

CODIFICA RAGIONEVOLE

Una codifica è ragionevole se:

- $|\Sigma| \geq 2$
- non introduce dati irrilevanti, né richiede una generazione elevata di dati per la rappresentazione di un'istanza del problema

Se la codifica è ragionevole, la complessità del problema è indipendente dalla codifica scelta per risolvere il problema.

LINGUAGGI FORMALI

Alcuni richiami:

Alfabeto $\Sigma = \{a_1,\ldots,a_q\}$

Stringa = $<a_1,\ldots,a_p>$ (se vuota, indicata da ε)

Linguaggio L = {stringa} (L vuoto, indicato da $\emptyset$)

Operatori (dati linguaggi A e B)

Unione: $A \cup B = \{x \mid x \in A \text{ o } x \in B\}$

Concatenazione: $AB = \{xy \mid x \in A, y \in B\}$

Potenza: $A^n = A^{n-1}A \quad n>0$ ($n=0$ $A^n = \{\varepsilon\}$)

Chiusura di Kleene: $A^* = A^0 \cup A^1 \cup A^2\ldots$

Complemento: $\overline{A} = \Sigma^* - A$

ACCETTAZIONE/RICONOSCIMENTO

Dato un algoritmo A e una stringa x di {0,1}*

- Se A(x)=1: A accetta x
- Se A(x)=0: A rifiuta x

Il linguaggio accettato da A è L = {x di {0,1}*| A(x)=1}
Anche se un linguaggio L è riconosciuto da A, l'algoritmo non necessariamente rifiuta una stringa che non appartiene a L

DECISIONE

Un linguaggio L è deciso da un algoritmo A se:

- ogni stringa binaria in L è accettata da A
- ogni stringa binaria in $\overline{L}$ è rifiutata da A

DEFINIZIONI

Un linguaggio L è riconosciuto in tempo polinomiale da un algoritmo A se, $\forall$ stringa x di lunghezza n

A accetta x in tempo $O(n^k)$ per qualche costante k

<==> $x \in L$

Un linguaggio L è deciso in tempo polinomiale da un algoritmo A se, $\forall$ stringa $x \in \{0,1\}^*$

A decide correttamente l'appartenenza di x a L in tempo

$O(n^k)$ per qualche costante k

CLASSI DI COMPLESSITÀ

classe di complessità = insieme di linguaggi per i quali l'appartenenza alla classe è determinata in base ad una misura di complessità dell'algoritmo che determina se una data stringa appartenga al linguaggio

CLASSE P

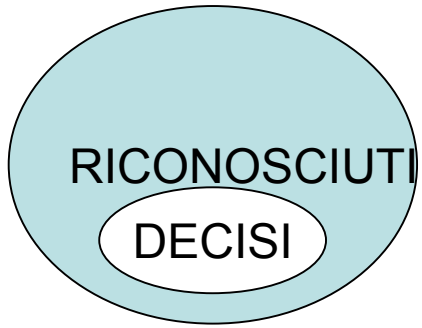
P = {L su alfabeto Σ={0,1}: esiste un algoritmo che decide L
in tempo polinomiale}

Teorema:

P = {L : L è riconosciuto da un algoritmo di tempo
polinomiale}

CLASSE P (1)

Dimostrazione:



Verifichiamo che se L è riconosciuto da un algoritmo polinomiale, allora L è deciso da un algoritmo polinomiale

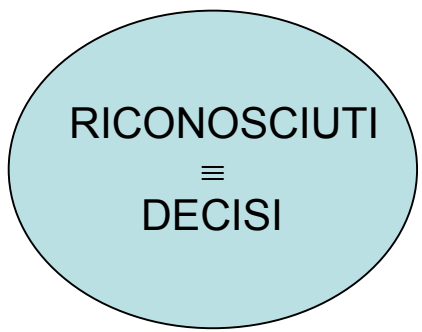
Sia L un linguaggio riconosciuto da un qualche algoritmo polinomiale A.
Poiché A riconosce L in tempo $O(n^k)$ per qualche k, esiste una costante c tale che A riconosce L in al più $T=cn^k$ passi.

CLASSE P (2)

Consideriamo un algoritmo B che simula l'azione di A per ogni stringa di input x.
Alla fine di T, l'algoritmo B verifica il comportamento di A.
Se A ha accettato x, allora B accetta x restituendo 1, se A ha rifiutato x, allora B rifiuta restituendo 0.
Il tempo impiegato da B per simulare A non aumenta il tempo di esecuzione per più di un fattore polinomiale.

Il linguaggio L è deciso da B

in tempo polinomiale



Verifica in tempo polinomiale

- esiste un cammino su un grafo (V,E) che va da un nodo *u* a un nodo *v* che abbia al più *k* archi?
  - PATH(x) = 1 o 0 (in tempo polinomiale)

x  <(V,E), u, v, k >

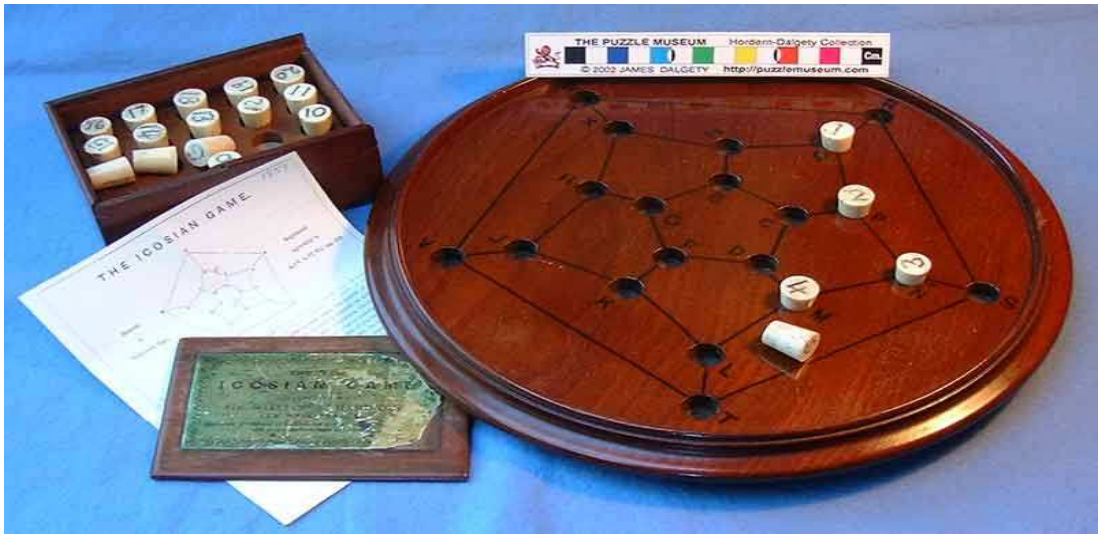
- Dato un cammino con al più *k* archi , si può verificare che sia un cammino buono
  - A (x, y) = 1 o 0 in tempo polinomiale

x  <(V,E), u, v, k >

y  il cammino di prova

Verifica in tempo polinomiale

- Spesso verificare una possibile soluzione è “più facile” che risolvere il problema da zero



Dato un grafo, esiste un ciclo hamiltoniano?

  - Un algoritmo che prova tutti i possibili cammini (permutazioni dei vertici) è superpolinomiale
  - Esiste un algoritmo polinomiale che dato il grafo ed un cammino verifica che questo sia un ciclo hamiltoniano

Verifica in tempo polinomiale

- Definiamo **algoritmo di verifica**, $A(x,y)$ con argomenti
  $x \in \{0,1\}^*$ stringa
  $y \in \{0,1\}^*$ **certificato**
- A **verifica** una stringa *x'* se esiste un certificato *y'* t.c. A(x',y')=1
- Si dice linguaggio verificato da A il linguaggio L

$$L = \{ x \in \{0,1\}^* : \exists\ y \in \{0,1\}^* \text{ tale che } A(x,y)=1 \}$$

- **A verifica il linguaggio L**
  - **se** $\forall\ x \in L,\ \exists\ y$ t.c $A(x,y)=1$
  - **se** $\forall\ x \notin L, \neg\exists\ y$ t.c. $A(x,y)=1$

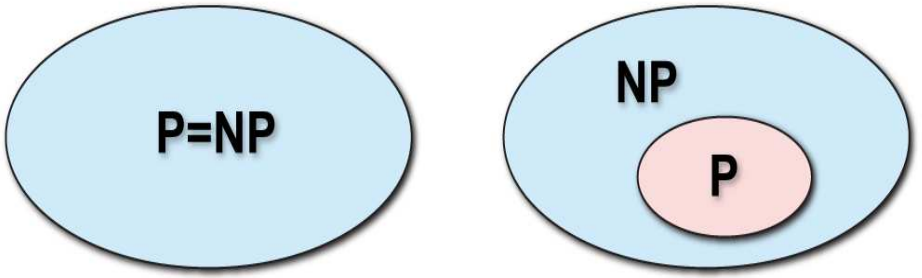
La Classe NP

- E' la classe dei linguaggi che possono essere
  verificati da un algoritmo in tempo polinomiale
  - $L \in NP \Leftrightarrow$
    $\exists$ A(x,y) polin. rispetto a |x| e una costante *c* t.c
    $L = \{x \in \{0,1\}^* : \exists$ certificato y con $|y| = O(|x|^c)$ t.c $A(x,y)=1\}$

- Sia $L \in P$. Allora esiste un algoritmo D che decide L in tempo polinomiale ($D(x)=1 \Leftrightarrow x \in L$)
  - Sia A(x,y) := D(x)
    - A verifica L !
      - quindi $P \subseteq NP$

P, NP, co-P, co-NP

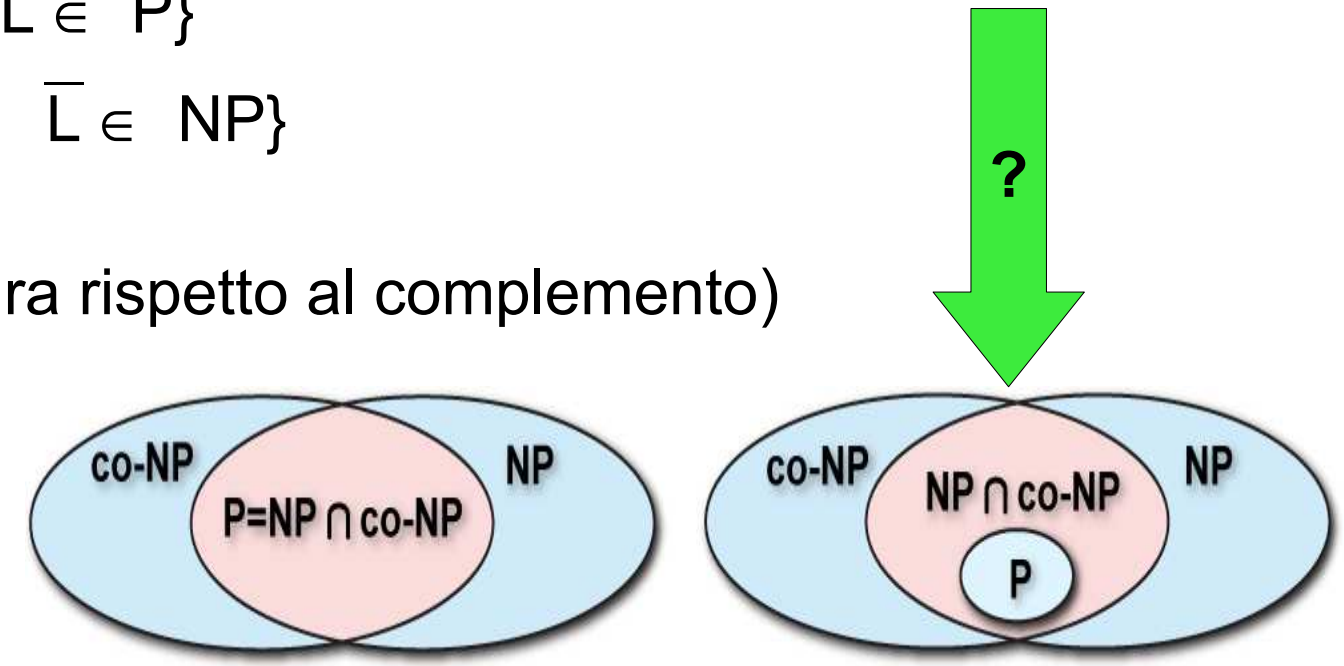
- $P \subseteq NP$
- $P \subset NP$?



  - co-P = $\{L : \overline{L} \in P\}$
  - co-NP = $\{L : \overline{L} \in NP\}$

- P=co-P (chiusura rispetto al complemento)

- NP=co-NP?



Riducibilità

- Problema 1: Risolvere un'equazione del tipo $\alpha x + \beta = 0$
- Problema 2: Risolvere un'equazione del tipo $ax^2 + bx + c = 0$

- Il primo è *difficile al più quanto il secondo*, perché può essere ad esso ricondotto (trasformando $\alpha x + \beta$ in $0x^2 + bx + c$, con $b=\alpha$ e $c=\beta$)

  Nel nostro caso,

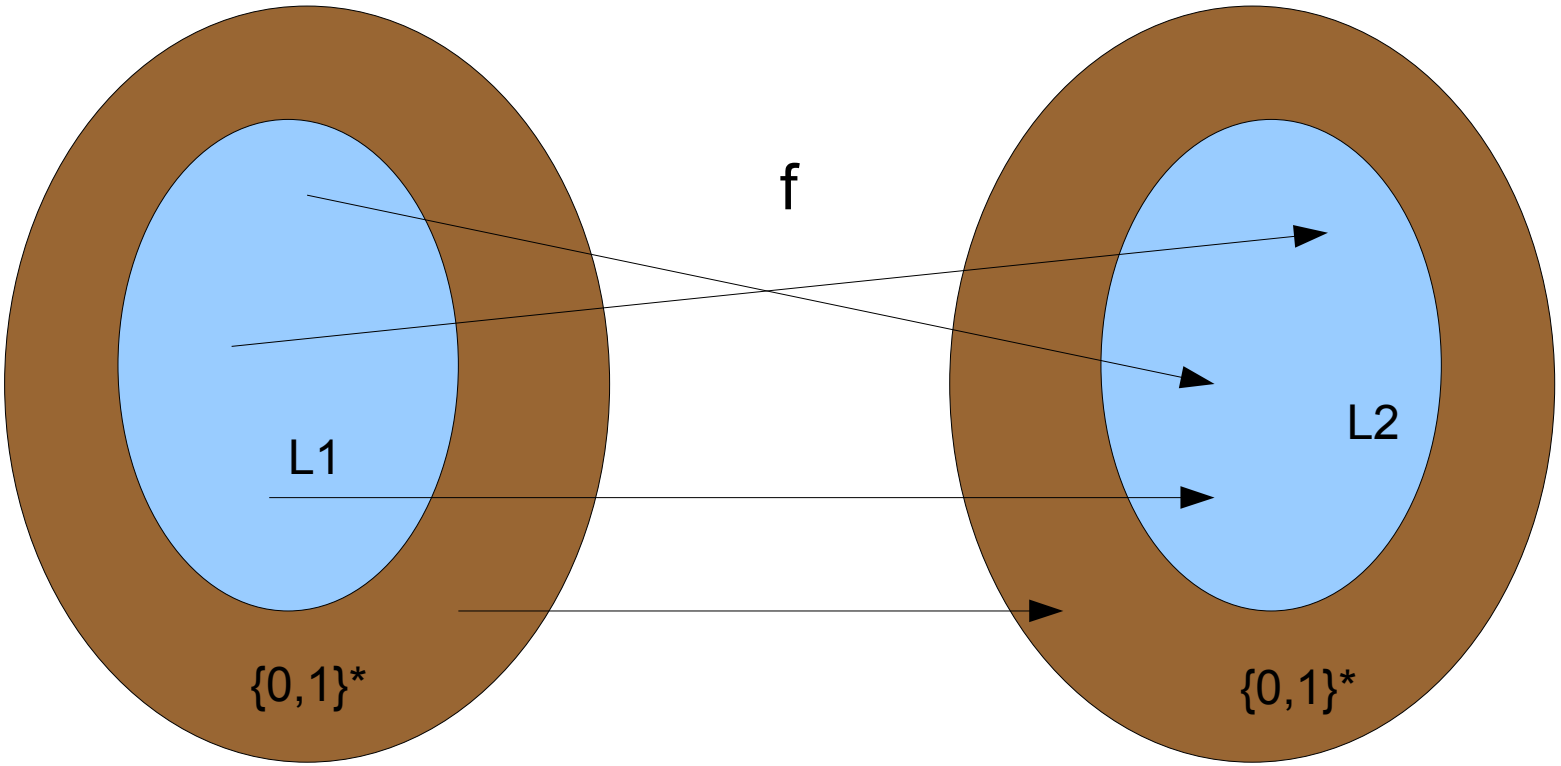
- Un linguaggio L1 **è riducibile in tempo polinomiale** a un linguaggio L2 ($\mathbf{L1 \leq_p L2}$) se

  $\exists$ f: $\{0,1\}^* \rightarrow \{0,1\}^*$ funzione calcolabile in t. polinomiale

  t.c $\forall$ x $\in \{0,1\}^*$, $\quad x \in L1 \Leftrightarrow f(x) \in L2$

Riducibilità

- **f**: funzione di riduzione
- **F** algoritmo con tempo polinomiale che calcola f : algoritmo di riduzione

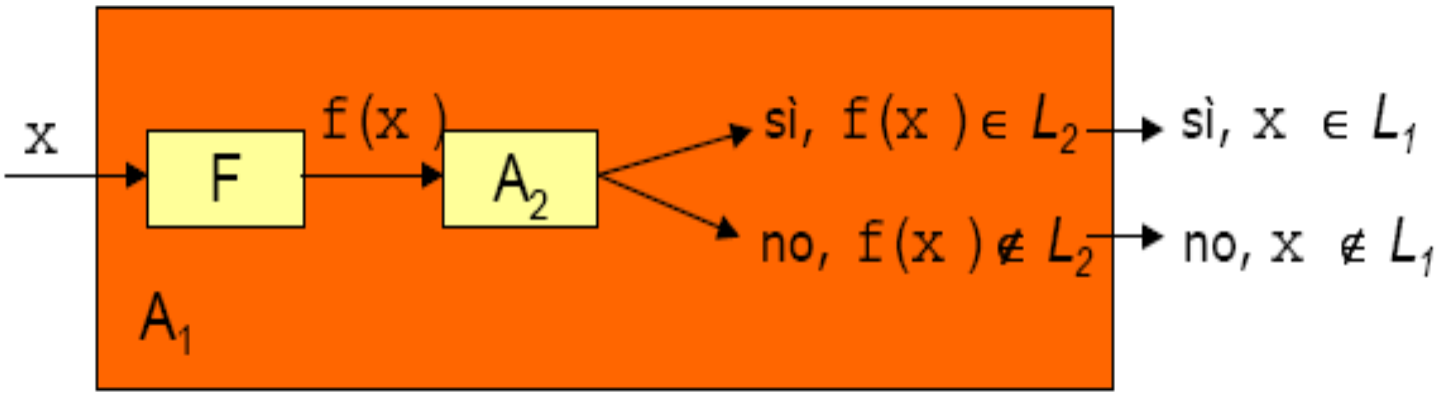


Riducibilità

- L1, L2 $\subseteq \{0,1\}*$ t.c. L1 $\leq_p$ L2, allora

  L2 $\in$ P $\Rightarrow$ L1 $\in$ P

Dim.
Sia A2 un algoritmo che decide L2 in tempo polinomiale, F un algoritmo di riduzione con tempo polinomiale da L1 a L2... [omissis]



NP e intrattabilità

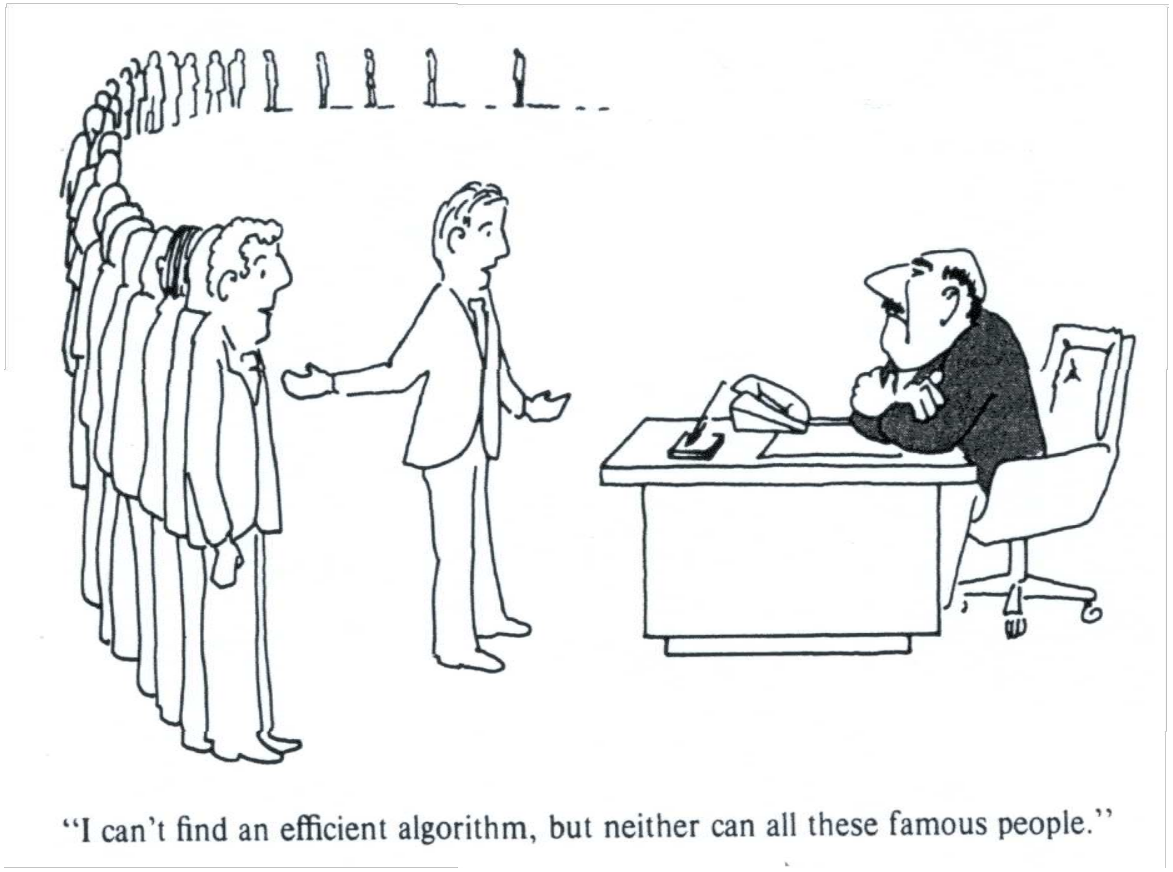
- Sia L un linguaggio tale che
  $L' \leq_p L \quad \forall L' \in NP$
  - » NP-difficile

- Sia L un linguaggio NP-difficile ***e*** di classe NP
  - » NP-completo (classe NPC)

- NP-difficili ~"problemi non necessariamente NP, difficili almeno quanto un qualsiasi $L' \in NP$"
- NP-completi ~"i più difficili tra gli NP"

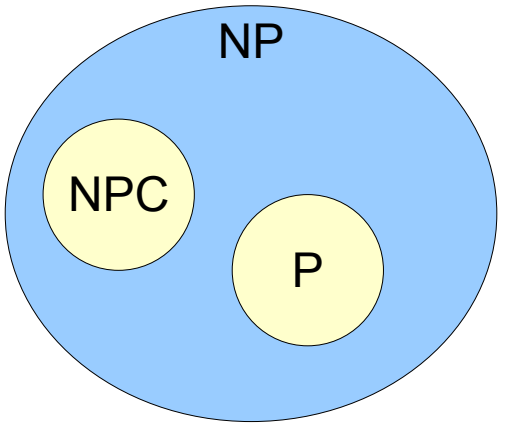
NP e intrattabilità

- Se L' $\in$ NPC e L' $\in$ P , allora P = NP
- Se $\exists$ L $\in$ NP non decidibile in tempo polinomiale, allora nessun linguaggio NP-completo è decidibile in tempo polinomiale (per assurdo...)



"I can't find an efficient algorithm, but neither can all these famous people."

Ad oggi non si è trovato alcun algoritmo con tempo polinomiale per alcun problema NP-completo



NP completezza

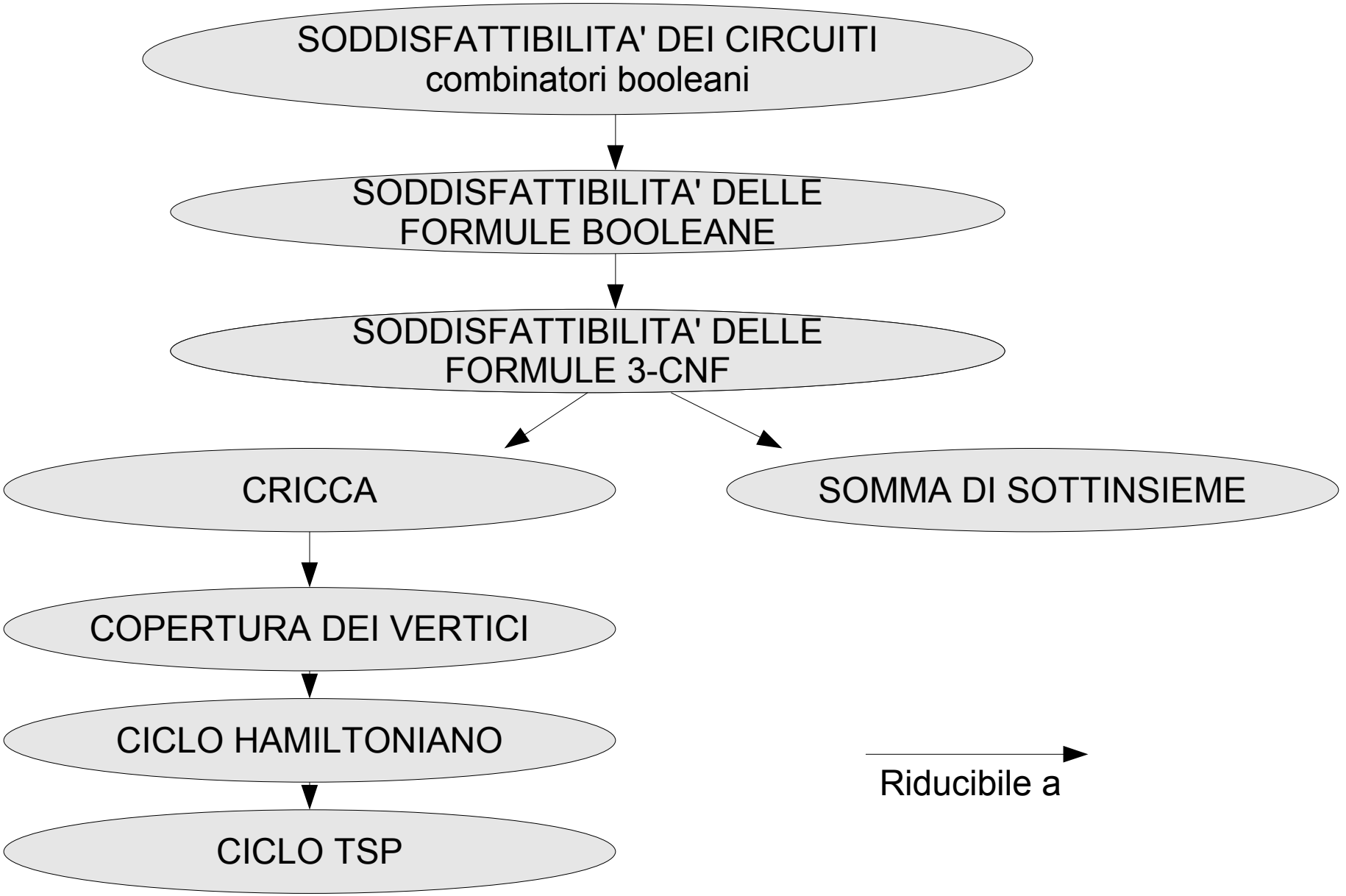
A) Dimostrazione “diretta”

B) Si dimostra $L \in NP$

Si cerca un L' NP-completo noto
Si trova F algoritmo di riduzione che permetta di ridurre L' a L in tempo polinomiale.

- B) è corretto per il seguente risultato
  - Se L è tale che $L' \leq_p L$ per qualche $L' \in NPC$, allora L è NP-difficile.

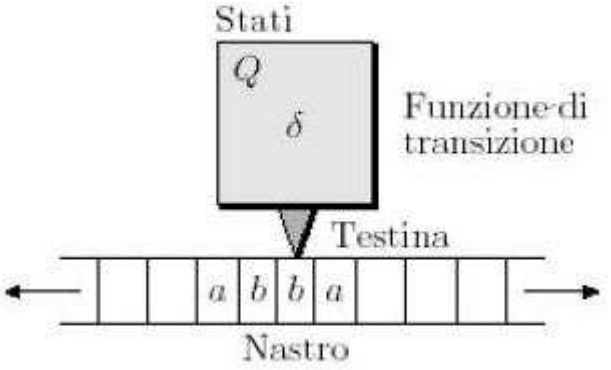
Alcuni problemi NP-completi



Macchina di Turing Deterministica

Una macchina di Turing deterministica (MTD) è una sestupla
$M = (\Gamma, b, Q, q0, F, \delta)$ t.c.:

- Γ è l'alfabeto dei simboli del nastro;
- b è un carattere speciale denominato BLANK;
- Q è un insieme finito e non vuoto di stati;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali;
- δ è la funzione di transizione.



$\delta: (Q - F) \times (\Gamma \cup \{b\}) \rightarrow Q \times (\Gamma \cup \{b\}) \times \{d,s,i\}$, dove
d, s e i indicano rispettivamente lo spostamento a destra, a sinistra e l'assenza di spostamento della testina.

Macchina di Turing Non Deterministica

Una macchina di Turing non deterministica (MTND) è una sestupla
$M=(\Gamma, b, Q, q0, F, \delta_N)$ tale che:

- Γ è l'alfabeto dei simboli del nastro;
- b è un carattere speciale denominato BLANK;
- Q è un insieme finito e non vuoto di stati;
- $q0$ è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali;
- δ_N è la funzione di transizione.
  $\delta_N: Q \times (\Gamma \cup \{b\}) \rightarrow \{P(Q \times (\Gamma \cup \{b\}) \times \{d,s,i\})\}$
  (con P insieme potenza), dove d, s e i indicano rispettivamente lo spostamento a destra, a sinistra e l'assenza di spostamento della testina.

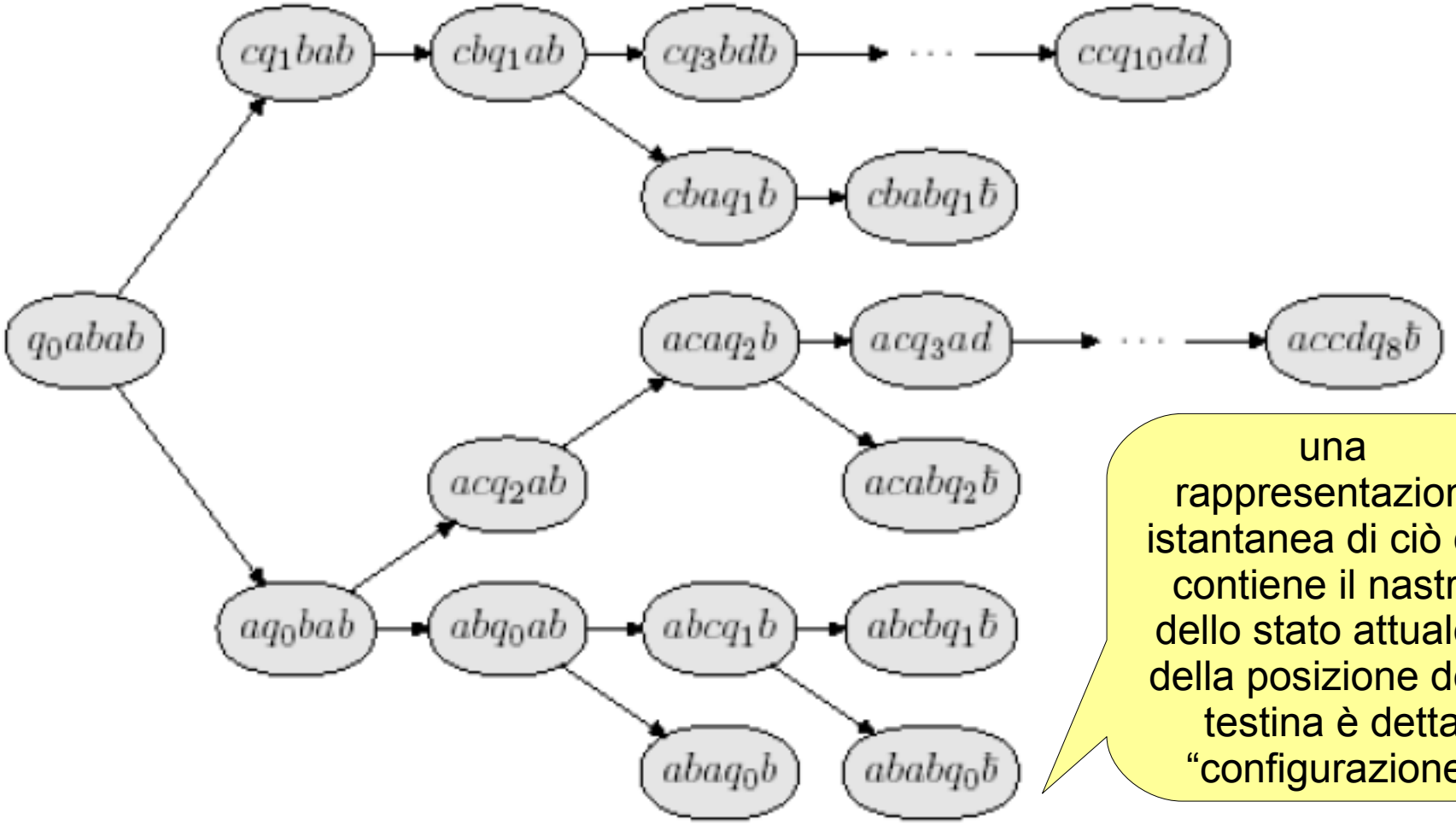
MTND - esempio

Rappresentazione di una possibile funzione di transizione per una MTND

	a	b	c	d	$\flat$
q_0	$\{(q_0, a, d), (q_1, c, d)\}$	$\{(q_0, b, d), (q_2, c, d)\}$	–	–	–
q_1	$\{(q_1, a, d), (q_3, d, s)\}$	$\{(q_1, b, d)\}$	–	–	–
q_2	$\{(q_2, a, d)\}$	$\{(q_2, b, d), (q_3, d, s)\}$	–	–	–
q_3	$\{(q_3, a, s)\}$	$\{(q_3, b, s)\}$	$\{(q_4, c, d)\}$	–	–
q_4	$\{(q_6, c, d)\}$	$\{(q_7, c, d)\}$	–	–	–
q_5	$\{(q_5, a, d)\}$	$\{(q_5, b, d)\}$	–	$\{(q_7, d, d)\}$	–
q_6	$\{(q_6, a, d)\}$	$\{(q_6, b, d)\}$	–	$\{(q_8, d, d)\}$	–
q_7	–	$\{(q_9, d, s)\}$	–	$\{(q_7, d, d)\}$	–
q_8	$\{(q_9, d, s)\}$	–	–	$\{(q_8, d, d)\}$	–
q_9	$\{(q_3, a, s)\}$	$\{(q_3, b, s)\}$	$\{(q_{10}, x, i)\}$	$\{(q_9, d, s)\}$	–
q_{10}	–	–	–	–	–

MTND esempio

- Comportamento della MTND con δ_N appena definita, input “abab”, Γ={a, b, c, d}, Q={q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10}, F={q10}



una rappresentazione istantanea di ciò che contiene il nastro, dello stato attuale e della posizione della testina è detta “configurazione”

(Ri-)definizioni

- Una **configurazione** *c* è **massimale** quando non è possibile applicare la funzione di transizione
  - Stato finale (accettazione)
  - Stato non finale (non accettazione)
- Per MTD, dato un input, si dice che è accettato (rifiutato) se il cammino di computazione termina in una configurazione massimale di accettazione (non accettazione)
- Per MTND, un input è accettato se almeno uno dei cammini di computazione termina in una configurazione massimale di accettazione.

Complessità spaziale e temporale

- Il tempo richiesto alla MTD per una computazione è ben descrivibile facendo riferimento al numero di configurazioni in cui la MTD si trova durante il cammino di computazione

- Lo spazio richiesto alla MTD è ben descrivibile facendo riferimento al numero di caselle accedute

- Il tempo richiesto da una MTND per accettare un input è il minimo tra i tempi delle sue computazioni accettanti.

- Lo spazio richiesto da una MTND per accettare un input è il minimo tra gli spazi delle sue computazioni accettanti.

Accettazione e Rifiuto (Stringhe)

- Dato un alfabeto Σ , una macchina di Turing (MTD) *M=(*Γ*, b, Q, q0, F,* δ*)* ed un intero $\tau > 0$, *una stringa x* $\in \Sigma^*$ *è accettata (rifiutata) da M in tempo* τ se esiste una computazione q0x ├ *c* tale che la sua esecuzione impiega un tempo *r* con $r \leq \tau$ e *c* è una configurazione massimale di accettazione (di non accettazione).
- Dato un alfabeto Σ , una macchina di Turing (MTD) *M= (*Γ*, b, Q, q0, F,* δ*)* ed un intero $\sigma > 0$, *una stringa x* $\in \Sigma^*$ *è accettata (rifiutata) da M in uno spazio* σ se esiste una computazione q0x ├ *c* nel corso della quale si accede a *v* celle di nastro con $v \leq \sigma$ e *c* è una configurazione massimale di accettazione (di non accettazione).

Accettazione di Linguaggi

- Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) *M = (*Γ*, b, Q, q0, F,* δ*)* ed una funzione t:N→N, diciamo che *M accetta L ⊆ Σ* in tempo t(n)* se, per ogni x ∈ Σ*, *M* accetta x in tempo al più t(|x|) se x ∈ L, mentre, se x∉L, allora *M* non lo accetta.
- Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) *M = (*Γ*, b, Q, q0, F,* δ*)* ed una funzione s:N→N, diciamo che *M accetta L ⊆ Σ* in spazio s(n)* se, per ogni x ∈ Σ*, *M* accetta x in uno spazio al più s(|x|) se x ∈ L, mentre, se x∉L, allora *M* non lo accetta.
- Diciamo che *un linguaggio L è accettabile in tempo* deterministico (non deterministico) *t(n)* se esiste una macchina di Turing deterministica (non deterministica) che accetta L in tempo t(n).

Decisione dei Linguaggi

- Dato un alfabeto Σ, una macchina di Turing deterministica *M = (*Γ*, b, Q, q0, F,* δ*)* ed una funzione t:N→N, diciamo che *M decide L ⊆ Σ* in tempo t(n)* se, per ogni x ∈ Σ*, *M* accetta x in tempo al più t(|x|) se x ∈ L, mentre, se x ∉L, allora *M* lo rifiuta sempre in un tempo al più t(|x|).
- Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) *M = (*Γ*, b, Q, q0, F,* δ*)* ed una funzione s:N→N, diciamo che *M accetta L ⊆ Σ* in spazio s(n)* se, per ogni x ∈ Σ*, *M* accetta x in uno spazio al più s(|x|) se x ∈ L, mentre, se x ∉L, allora *M* lo rifiuta sempre in uno spazio al più s(|x|).
- Diciamo che *un linguaggio L è decidibile in tempo t(n)* se esiste una macchina di Turing deterministica che accetta L in tempo t(n).

In breve

- TEMPO
  - Un linguaggio L è *accettabile* in tempo deterministico (non deterministico) t(n) se esiste una macchina di Turing deterministica (non deterministica) che accetta L in tempo t(n).
  - Un linguaggio L è *decidibile* in tempo t(n) se esiste una macchina di Turing deterministica che decide L in tempo t(n).
- SPAZIO
  - Un linguaggio L è *accettabile* in spazio deterministico (non deterministico) se esiste una macchina di Turing deterministica (non deterministica) che accetta L in spazio s(n).
  - Un linguaggio L è *decidibile* in spazio s(n) se esiste una macchina di Turing deterministica che decide L in spazio s(n).

Classi di complessità (cenni)

Sia f una funzione $f:N \rightarrow N$,

- *DTIME(f(n))* è l'insieme dei linguaggi *decisi* da una macchina di Turing deterministica in un tempo al più f(n);
- *DSPACE(f(n))* è l'insieme dei linguaggi *decisi* da una macchina di Turing deterministica in spazio al più f(n);
- *NTIME(f(n))* è l'insieme dei linguaggi *accettati* da una macchina di Turing non deterministica in un tempo al più f(n);
- *NSPACE(f(n))* è l'insieme dei linguaggi *accettati* da una macchina di Turing non deterministica in spazio al più f(n).

Classi di complessità notevoli

$$P = \bigcup_{k=0}^{\infty} DTIME\left(n^k\right)$$

$$NP = \bigcup_{k=0}^{\infty} NTIME\left(n^k\right)$$

$$PSPACE = \bigcup_{k=0}^{\infty} DSPACE\left(n^k\right)$$

$$NPSPACE = \bigcup_{k=0}^{\infty} NSPACE\left(n^k\right)$$

Classi di complessità notevoli

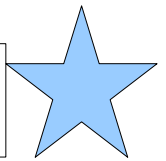
$$EXPTIME = \bigcup_{k=0}^{\infty} DTIME\left(2^{n^k}\right)$$

$$NEXPTIME = \bigcup_{k=0}^{\infty} NTIME\left(2^{n^k}\right)$$

$$LOGSPACE = DSPACE(\log n)$$

Teoremi di gerarchia (cenni)

- Si riportano (senza pretesa alcuna di completezza) alcuni risultati inerenti alle classi viste
  - LOGSPACE $\subset$ PSPACE
  - P $\subset$ EXPTIME
  - NP $\subset$ NEXPTIME
- Teorema :
  - Per ogni funzione f:N$\rightarrow$N valgono le due proprietà:
    - DTIME(f(n)) $\subseteq$ NTIME(f(n));
    - DSPACE(f(n)) $\subseteq$ NSPACE(f(n)).
  - "dim."
    - MTD può essere vista come un caso particolare di MTND
- Da cui P $\subseteq$ NP, PSPACE $\subseteq$ NPSPACE



Teoremi di gerarchia (cenni)

- Poiché in t passi di computazioni è impensabile che una MTD con un singolo nastro visiti più di t celle si ha il seguente terema
  - Per ogni funzione $f:N \rightarrow N$ valgono le due proprietà:
    - $DTIME(f(n)) \subseteq DSPACE(f(n))$;
    - $NTIME(f(n)) \subseteq NSPACE(f(n))$.
- $P \subseteq PSPACE$
- $NP \subseteq NPSPACE$
- Teorema di SAVITCH
  - Per ogni funzione $f(n) \geq \log(n)$
    - $NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$
  - Corollario: **NPSPACE = PSPACE**
    - Dim: per la chiusura dei polinomi rispetto all'elevazione al quadrato e per 

Soddisfattibilità dei circuiti (Bonus slide)

- $L \in NP \Rightarrow \exists A$ che lo verifica in un tempo,nel caso peggiore, T(n) (n= |x|, x stringa e y certificato di lunghezza $O(n^k)$ per $k \geq 1$) polinomiale in n.
- A è eseguito con l'hardware del calcolatore (elementi combinatori + sequenziali)
  - Gli elementi di memoria possono essere rimossi a patto di ripetere T(n) gli elementi combinatori in gioco (da uno stato in memoria in un certo ciclo si passa a segnali che insistono tra un circuito combinatorio e il successivo)
    - Chiamiamo “configurazione” uno stato della memoria del calcolatore
      - esecuzione di un'istruzione: configurazione' →configurazione"
      - sia M il circuito booleano che mappa una configurazione in un'altra
- F riceve x in ingresso: realizza un circuito combinatorio formato da T(|x|) copie di M in cascata, al cui ingresso vi sono i segnali corrispondenti a y. Chiamiamo $\mathbf{C_x(y)}$ questo circuito combinatorio, il cui output dovrà essere l'output del calcolo di **A(x,y)**.

Soddisfattibilità dei circuiti (Bonus slide II)

- E' facile convincersi che F può creare C_x in tempo polinomiale.
  - A eseguito al più in $O(n^k)$ passi, quindi saranno necessarie $O(n^k)$ ripetizioni di M
  - Un configurazione è rappresentabile con un numero di bit polinomiale in n.
  - M deve trattare i bit che sarebbero in memoria, quindi dipende dalla lunghezza della configurazione, che è polinomiale...
- $x \in L \Rightarrow \exists$ y t.c. A(x,y)=1. Ma allora esiste un ingresso y che applicato al circuito C_x è tale che $C_x(y)=1$ e quindi $f(x)=C_x$ è soddifattibile.
- $C_x \in$ SAT (insieme dei circuiti booleani soddisfattibili) $\Rightarrow \exists$ y t.c $C_x(y)=1$, ma C_x calcola appunto A(x,y), che allora per y vale 1, e dunque verifica l'appartenenza di x ad L.
  - è dimostrato che

$$x \in L \Leftrightarrow f(x) \in SAT$$

- SAT è NP-difficile (ed è anche NP, quindi è NP-completo).