

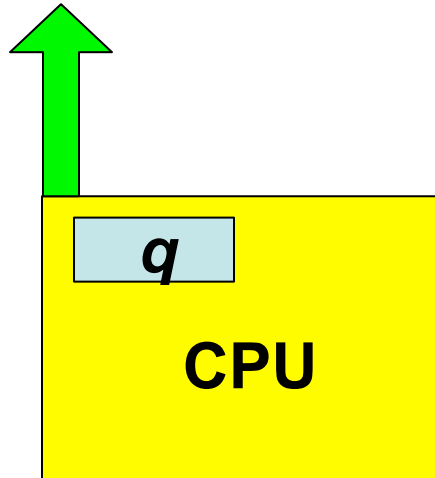
Sommario

- Complessità di **tempo** e **spazio** di una TM che si ferma **sempre**.
- Relazioni tra le due misure
- Analisi complessità delle TM costruite per dimostrare che più nastri o il non determinismo non aumentano il potere computazionale del modello
- Definizione **classi di complessità**

Complessità di tempo di una TM M



... Da ora in poi consideriamo **solo** TM che si **fermano sempre**



Contiamo i passi!

Definiamo la funzione $\text{Passi}_M : \{a,b\}^* \rightarrow \mathbb{N}$ tale che $\text{Passi}_M(x) = n^\circ$ passi su input x

Definiamo la funzione $t_M : \mathbb{N} \rightarrow \mathbb{N}$ tale che $t_M(n) = \max\{\text{Passi}_M(x) \text{ per } x \text{ di lunghezza } n\}$

Il caso peggiore!

Esempio: la TM M che riconosce $\{0^n 1^n \mid n \geq 0\}$

su input del tipo 10^{n-1} T termina in **un passo** rifiutando

Quindi $\text{Passi}_M(10^{n-1}) = 1$

su input del tipo $0^m 1^m$ tale che $2m = n$

T compie $m + 1$ passi per la prima coppia 01 per ottenere sul nastro $X0^{m-1}Y1^{m-1}$

poi la TM torna indietro di $m + 1$ posizioni e compie altri $m + 1$ passi per la seconda coppia...

Ogni volta la distanza tra lo 0 e l'1 corrispondente è m quindi complessivamente si compiono $O(m^2)$ passi:
 $\text{Passi}_M(0^m 1^m) = O(m^2)$. Nel caso peggiore quindi

$$t_M(n) = O(n^2)$$

Dimensione input

La complessità di tempo o di spazio è sempre misurata in funzione della dimensione dell'input, quindi la rappresentazione dei dati può fare differenza.

In questo contesto ogni rappresentazione **ragionevole** va bene.

Per un numero

per es **17**

vanno bene

17

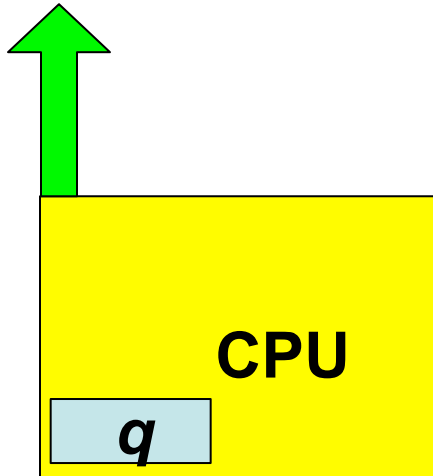
10001

non va bene

$1^{17} = 11111111111111111$

N.B. Per ogni T in TM , assumiamo che un input di lunghezza n dovrà essere letto interamente almeno in un caso, e quindi che vale sempre $t_T(n) \geq n$.

Complessità di spazio di una TM M



contiamo il
numero di
celle utilizzate!



Definiamo la funzione $T_M : \{a,b\}^* \rightarrow \mathbb{N}$ tale che
 $Celle_M(x) = n^\circ$ celle utilizzate su input x

Definiamo la funzione

$$s_M(n) = \max\{Celle_M(x) \text{ per } x \text{ di lunghezza } n\}$$

Quindi, nella nostra ipotesi sull'input, $s_M(n) \geq n$

Il caso peggiore!

Relazione tra spazio e tempo

Conoscendo la complessità di tempo possiamo limitare superiormente quella di spazio?

Poichè ogni cella visitata comporta un passo di calcolo è evidente che per ogni TM T:

$$s_T(n) \leq t_T(n)$$

E se si conosce la complessità di spazio $s_T(n)$ possiamo limitare superiormente quella di tempo?

Relazione tra spazio e tempo - 2

Se la complessità di spazio è $s_T(n)$ allora il contenuto del nastro in ogni passo di calcolo è una parola di lunghezza al più $s_T(n)$.

Possiamo allora calcolare il numero di tutte le configurazioni della TM in questione con contenuto del nastro di lunghezza al più $s_T(n)$, ottenendo così un limite superiore banale alla complessità di tempo della TM.

Relazione tra spazio e tempo - 2

Se d è il numero di simboli dell'alfabeto di nastro, il numero delle parole di lunghezza $s_T(n)$ è $d^{s_T(n)}$

Tenendo conto che in ogni stato si può avere un qualsiasi contenuto di nastro, detto q il numero degli stati, abbiamo

$$q * d^{s_T(n)}$$

e infine tenendo conto che la testina di lettura può trovarsi su una cella qualunque, il numero delle configurazioni è

$$q * s_T(n) * d^{s_T(n)}$$

In conclusione $t_T(n) \leq 2^{O(s_T(n))}$

Spazio, tempo e numero di nastri

La complessità di spazio di una TM T a k nastri, $s_T(n)$, è $s_T(n) = \max\{\text{la somma dei n° delle celle utilizzate su input } x \text{ di lunghezza } n, \text{ su ogni nastro}\}$

Teorema. Se L è riconosciuto da una TM con k nastri con complessità di spazio $s_T(n)$, allora L è riconosciuto da una TM a un solo nastro con complessità di spazio $O(s_T(n))$.

E per il tempo?

Tempo e numero di nastri

Teorema. Se L è riconosciuto da una TM con k nastri con complessità di tempo $t_T(n)$, allora L è riconosciuto da una TM a un solo nastro con complessità di tempo $O((t_T(n))^2)$.

Prova. Ricordiamo che la TM a un solo nastro inizializza il proprio nastro per predisporre la simulazione della macchina a k nastri.

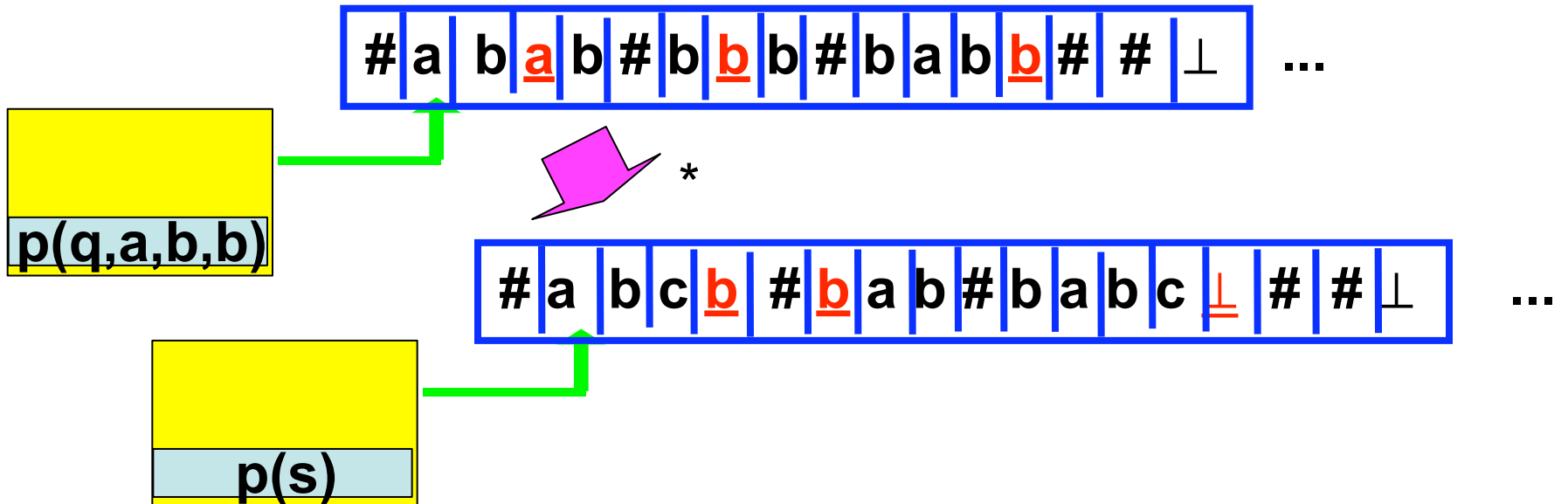
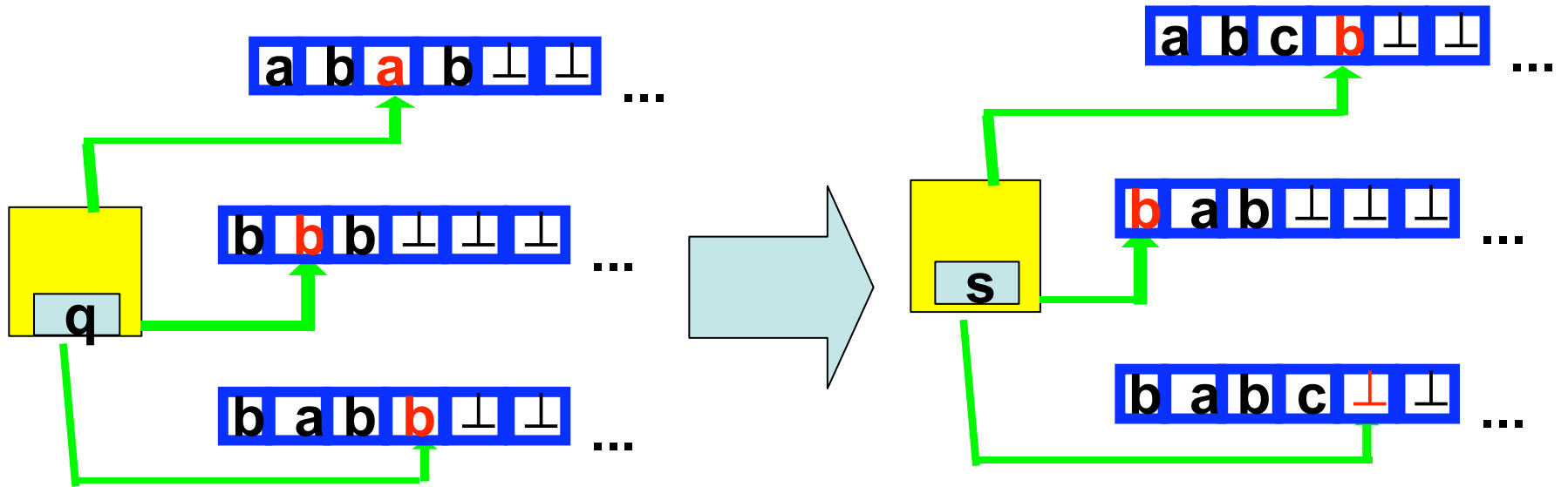
Alla fine di questa fase avremo questo contenuto sul nastro:

$$\$p(q_0)\underline{a_1}\dots\underline{a_n}\underbrace{\# \perp \# \dots \perp \# \perp \#\#}_{k-1 \text{ volte}}$$

dove $\underline{a_1}\dots\underline{a_n}$ è la stringa input. Questo costa $O(n)$.

Simulazione di una mossa

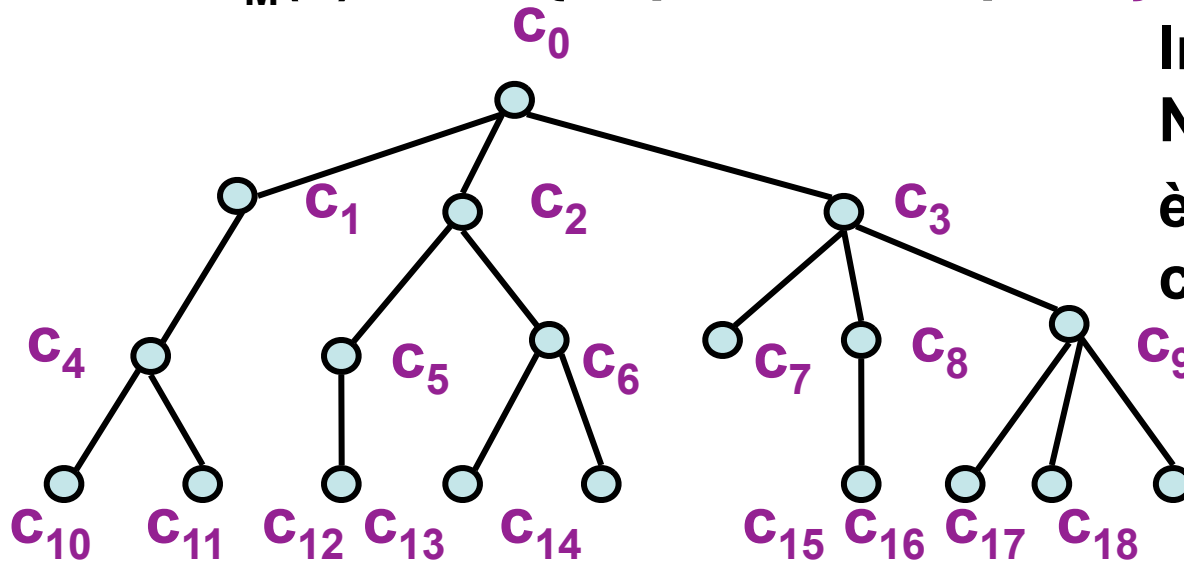
$k = 3$



Complessità di tempo di una NTM M

Definiamo la funzione $\text{NPassi}_M : \{a,b\}^* \rightarrow \mathbb{N}$ tale che

$$\text{NPassi}_M(x) = \max\{\text{n}^\circ \text{ passi su input } x\}$$



In altre parole

$$\text{NPassi}_M(x) : \{a,b\}^* \rightarrow \mathbb{N}$$

è l'altezza dell'albero di computazione di x .

Anche qui $t_T(n) \geq n$.

Definiamo la funzione $t_M : \mathbb{N} \rightarrow \mathbb{N}$ tale che

$$t_M(n) = \max\{\text{NPassi}_M(x) \text{ per } x \text{ di lunghezza } n\}$$

Il caso peggiore!

In altre parole $t_M(n) : \mathbb{N} \rightarrow \mathbb{N}$ è l'altezza dell'albero di computazione più alto tra tutti quelli determinati da input di lunghezza n .

Complessità di spazio di una NTM M

Definiamo la funzione $S_M : \{a,b\}^* \rightarrow \mathbb{N}$ tale che
 $NCelle_M(x) = \max\{n^\circ \text{ celle utilizzate in ogni computazione su input } x\}$

Definiamo la funzione $s_M : \mathbb{N} \rightarrow \mathbb{N}$ tale che:
 $s_M(n) = \max\{NCelle_M(x) \text{ su input } x \text{ di lunghezza } n\}$

Anche qui $s_M(n) \geq n$.

Spazio, tempo e nontederminismo

Sappiamo costruire una TM equivalente a una NTM data.

Possiamo mettere in relazione le complessità di tempo e spazio delle macchine?

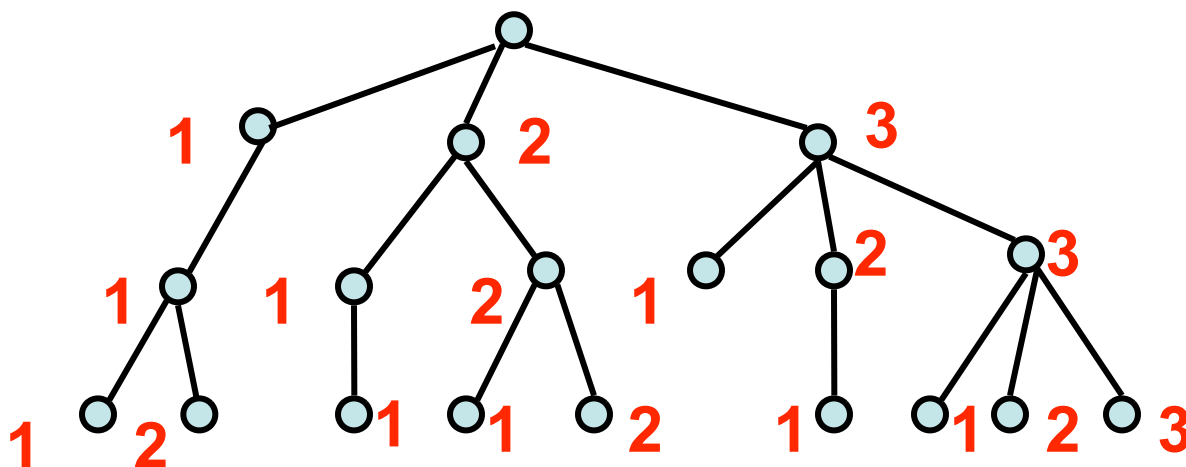
Dobbiamo prima **modificare la costruzione della TM in modo che si fermi quando la NTM data si ferma!**

Una TM equivalente a una NTM che non si ferma sempre

- Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ la NTM da simulare.
- Sia $d = \max \{ |\delta(q, a)| \mid q \text{ in } Q \text{ e } a \text{ in } \Gamma \}$
- Sia $D = \{1, 2, \dots, d\}$ l'alfabeto delle sequenze guida
- Sia M_0 la TM che genera e scrive sul nastro la sequenza successiva a quella data in input.
- La TM M' equivalente a M è una 3-TM che esegue tutte le possibili esecuzioni della NTM nell'ordine dettato dalle sequenze guida sul terzo nastro. Queste ultime vengono generate e aggiornate dalla copia della TM M_0 incorporata in M' . Come visto, la TM M' usa il secondo nastro per ogni esecuzione, e conserva l'input sul primo per poterlo ricopiare e avviare la successiva esecuzione. M' si ferma e accetta quando M si ferma e accetta.
- In tutti gli altri casi **non** si ferma, perchè genera la successiva sequenza sul terzo nastro e ricomincia la simulazione.

Una TM equivalente a una NTM che si ferma sempre: il caso del rifiuto

Una parola è rifiutata se ogni cammino di computazione su di essa è di rifiuto :



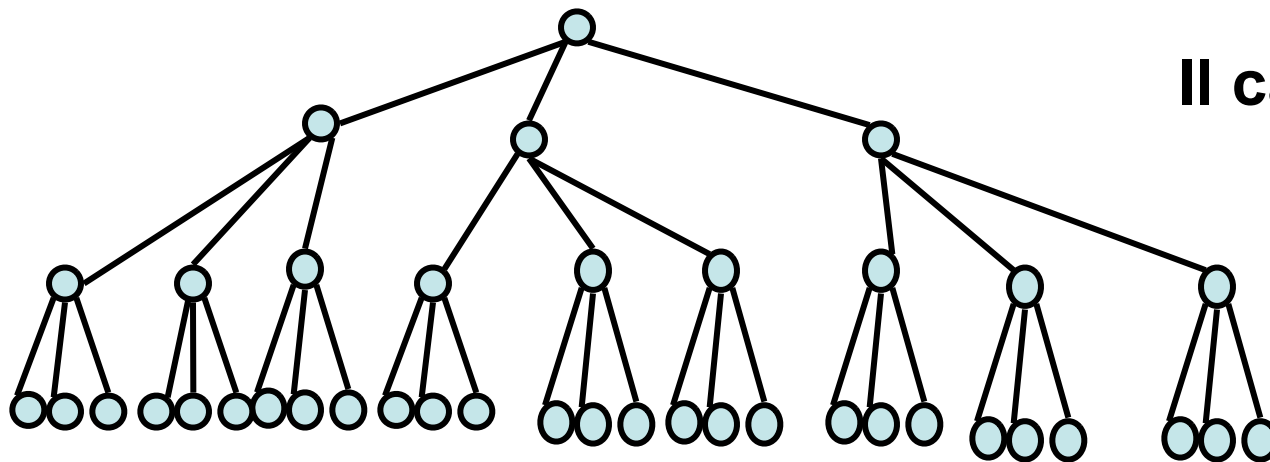
Aggiungiamo un contatore, il IV nastro, che sarà inizializzato a 1 ogni volta che si genera sul III nastro una sequenza del tipo 1^n , cioè quando si simula il cammino più a sinistra.

Il contatore verrà incrementato di 1 ogni volta che la computazione guidata dalla sequenza sul terzo nastro produce uno stato di rifiuto o fallisce. Il contatore viene confrontato con d^n , quando si genera sul IV nastro la sequenza d^n (cioè si simula l'ultimo cammino di computazione di lunghezza n) e se i due valori coincidono si ferma e rifiuta.

Spazio, tempo e nondeterminismo -2

Otteniamo un limite superiore alla complessità di tempo della TM deterministica equivalente alla NTM data, considerando il più grande albero di computazione possibile.

Si tratta dell'albero d -ario completo di altezza la complessità di tempo $t_T(n)$, dove d è il massimo grado di non determinismo.



Il caso di $d=3$

Spazio, tempo e nondeterminismo - 3

Il numero dei nodi di un albero d -ario completo di altezza $t_T(n)$ è

$$(d^{t_T(n)+1} - 1)/(d - 1) = O(d^{t_T(n)})$$

Quindi nel passaggio da una NTM alla TM equivalente la complessità di tempo diventa $O(t_T(n)d^{t_T(n)}) = 2^{O(t_T(n))}$.

Teorema. Se L è riconosciuto da una NTM con complessità di tempo $t_T(n)$, allora L è riconosciuto da una TM con complessità di tempo

$$2^{O(t_T(n))}$$

Classi di complessità

Chiamiamo

(N)TIME(f(n))

la classe dei linguaggi decisi da una (N)TM in tempo $O(f(n))$

Chiamiamo

(N)SPACE(f(n))

la classe dei linguaggi decisi da una (N)TM in spazio $O(f(n))$

Classi notevoli

La classe dei linguaggi decisi in tempo polinomiale da una **TM**:

$$P = \bigcup_{k \geq 0} \text{TIME}(n^k)$$

La classe dei linguaggi decisi in tempo polinomiale da una **NTM**

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

Evidentemente

$$P \subseteq NP$$

La classe P

In virtù della tesi di Church-Turing possiamo parlare di **P** come della classe dei problemi risolvibili da un algoritmo in tempo polinomiale.

La tesi di Cobhan-Edmonds afferma che possiamo parlare di **P** come della classe dei problemi **trattabili**.

La definizione di Jack Edmonds (1965)

Efficiente: tempo polinomiale per tutti gli inputs.

Inefficiente: tempo esponenziale per qualche inputs.

La classe P

In un articolo apparso negli atti dell' International Congress for Logic, Methodology, and Philosophy of Science del 1964, Alan Cobham ha formalmente caratterizzato l'insieme dei problemi risolvibili in tempo polinomiale dimostrandone l' **indipendenza** da un particolare modello di macchina deterministica. Nello stesso articolo ha mostrato anche che molte comuni funzioni matematiche sono calcolabili in tempo polinomiale.

In un articolo apparso sul Canadian Journal of Mathematics nel 1965, Jack Edmonds dimostra che il problema del massimo matching può essere risolto da un algoritmo polinomiale. Un matching in un grafo è un sottoinsieme di archi con la proprietà che due archi comunque presi nell'insieme non hanno un vertice in comune. La sezione 2 dell'articolo contiene una 'digressione filosofica sul concetto di "algoritmo efficiente"', come egli stesso afferma nell'introduzione.

Egli propone di utilizzare il tempo polinomiale per identificare gli algoritmi "buoni" o "efficienti".

La classe P

A sostegno della tesi di Cobhan-Edmonds troviamo la tesi di Church-Turing **estesa** che afferma che tutto ciò che possiamo calcolare in tempo $t(n)$ su un modello di calcolo universale e ragionevole (=fisicamente realizzabile) può essere calcolato con una TM in tempo $t(n)^{O(1)}$, cioè in tempo polinomialmente correlato.

Quindi **P** è **invariante** rispetto al modello di calcolo, come già osservato da Cobham.

O in termini informali:

Church-Turing **estesa**: ogni modello di calcolo generale e **fisicamente realizzabile** definisce lo stesso insieme di problemi **efficientemente** risolvibili.

La classe P

A sostegno della tesi di Cobham-Edmonds notiamo anche che **P** è chiusa rispetto alle principali operazioni sugli algoritmi:
composizione in sequenza,
utilizzo di un algoritmo all'interno di un ciclo in un altro algoritmo, oppure
utilizzo di un algoritmo, tra le varie istruzioni del nuovo

Infatti possiamo
sommare,
moltiplicare o
comporre funzionalmente tra loro polinomi ottenendo ancora polinomi.

Per contro se il grado del polinomio è “grande” la trattabilità non è più così realistica, ma considerazioni di tipo empirico portano a pensare che ogni “naturale” problema risolvibile in tempo polinomiale, lo è con un algoritmo polinomiale di grado piccolo (≤ 4)

Esempi di problemi in P

- **Tutti i problemi di decisione per FA: il problema del vuoto, il problema dell'infinito, il problema dell'appartenenza, il problema dell'equivalenza...**
- **Alcuni problemi di decisione per CFG: il problema dell'appartenenza per CFG (l'algoritmo CYK!), il problema del vuoto, del finito.**
- **Ben noti problemi sui grafi: connessione di un grafo non diretto, esistenza di un cammino tra due nodi, etc.**

Esempio di problema in P

PRIMES: Dato un intero positivo x , x è primo?

Istanza 1. $z = 23.536.481.273$.

risposta NO: $z = 104.729 \times 224.737$.

Istanza 2. $z = 23.536.481.277$.

risposta SI

PRIMES = $\{z \mid z \text{ è un numero primo}\}$ si risolve con un algoritmo polinomiale, come Agrawal-Kayal-Saxena hanno dimostrato nel 2002.

Esempio di problema in NP

FACTOR: Dati due interi positivi z e U , c'è un divisore non banale di z minore di U ?

Istanza 1. $z = 23.536.481.273$ e $U = 110.000$.

risposta SI: $z = 104.729 \times 224.737$.

Istanza 2. $z = 23.536.481.273$ e $U = 100.000$.

risposta NO : $104.729 \times 224,737$ è la fattorizzazione in fattori primi di z .

Istanza 3. $z = 23.536.481.277$ e $U = 23.536.481.277$

risposta NO : z è un numero primo.

Algoritmo non deterministico e polinomiale per FACTOR

CercaFattoriNonDet(x,U)

prec: ($U \leq \sqrt{x}$)

“non deterministicamente prendi un numero k tra 2 e U”

if $x \bmod k = 0$ **then** “SI” **else** “NO”

Analisi complessità:

L'algoritmo CercaFattoriNondet lavora in tempo polinomiale

Nel 1994 Peter Shor (MIT) ha sviluppato un algoritmo probabilistico in grado di risolvere FACTOR in tempo polinomiale con un calcolatore quantistico.

Algoritmo deterministico esponenziale per FACTOR

CercaFattori(x,U)

prec: ($U \leq \sqrt{x}$)

$k = 2$

while $k \leq U$

if $x \bmod k = 0$ **then** “SI”

else $k = k + 1$

“NO”

Analisi complessità:

L'algoritmo CercaFattori nel caso peggiore esegue \sqrt{x} volte il ciclo. Quindi se n è la lunghezza della rappresentazione binaria di x (quindi x è minore di 2^{n+1}) allora l'algoritmo ha una complessità in $O(2^{n/2})$. Infatti considera tutti i numeri binari tra 2 e $2^{n/2}$.

Per es. se $x = 2^{1024}$, cioè circa 10^{308} , ci sono circa 10^{154} numeri da verificare.

Esempi di problemi in NP

- **Il problema della soddisfacibilità per formule booleane, per formule booleane in CNF, in 3CNF,...**
- **molti problemi sui grafi: l'esistenza di una k-colorazione, per $k \geq 3$, di un k-clique con k in input, di un ciclo hamiltoniano in un grafo diretto o no,...**

P vs NP

Uno dei problemi fondamentali dell'Informatica teorica è capire se

$$P \subset NP \text{ o } P = NP$$

Se $P = NP$, allora

- avremmo algoritmi efficienti per TSP, FACTOR.
- schemi crittografici come RSA sarebbero inutili, con conseguenze drammatiche per il sistema bancario

Se $P \subset NP$, allora

- non avremmo algoritmi efficienti per TSP, FACTOR.
- le banche starebbero tranquille

P vs NP

P \subset NP

è sostenuta dall'intuizione che trovare una soluzione per un problema è più difficile che verificarne una soluzione.

Una vignetta di Pavel Pudlák

PF'03



* SORRY, THIS CARTOON IS TOO SMALL TO CONTAIN THE PROOF