Sommario

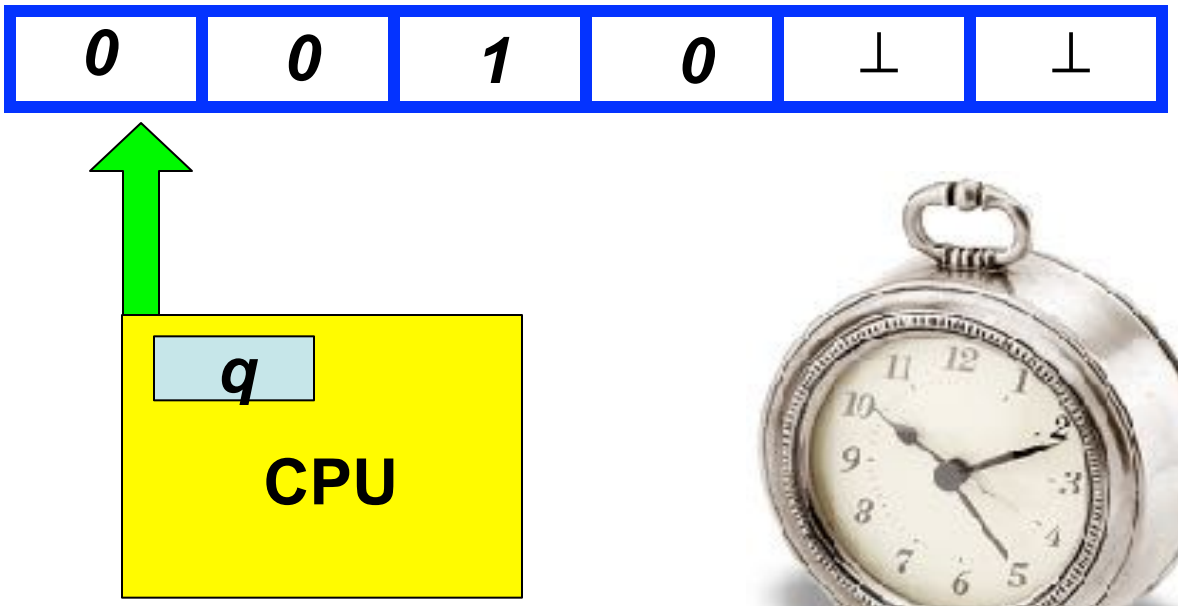
- **Complessità di tempo e spazio di una TM che si ferma sempre.**

- **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 $Passi_M : \{0,1\}^* \rightarrow N$ tale che

$Passi_M(x)$ =n° passi su input x

Definiamo la funzione $t_M : N \rightarrow N$ tale che

$t_M(n) = \max\{Passi_M(x)$ per x di lunghezza n$\}$

Il caso peggiore!

Esempio: la TM M che riconosce $\{0^n1^n \mid n \geq 0\}$

su input del tipo 10^{n-1} T termina in un passo rifiutando
Quindi $Passi_M(10^{n-1}) = 1$

su input del tipo 0^m1^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:
$Passi_M(0^m1^m) = O(m^2)$. 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.

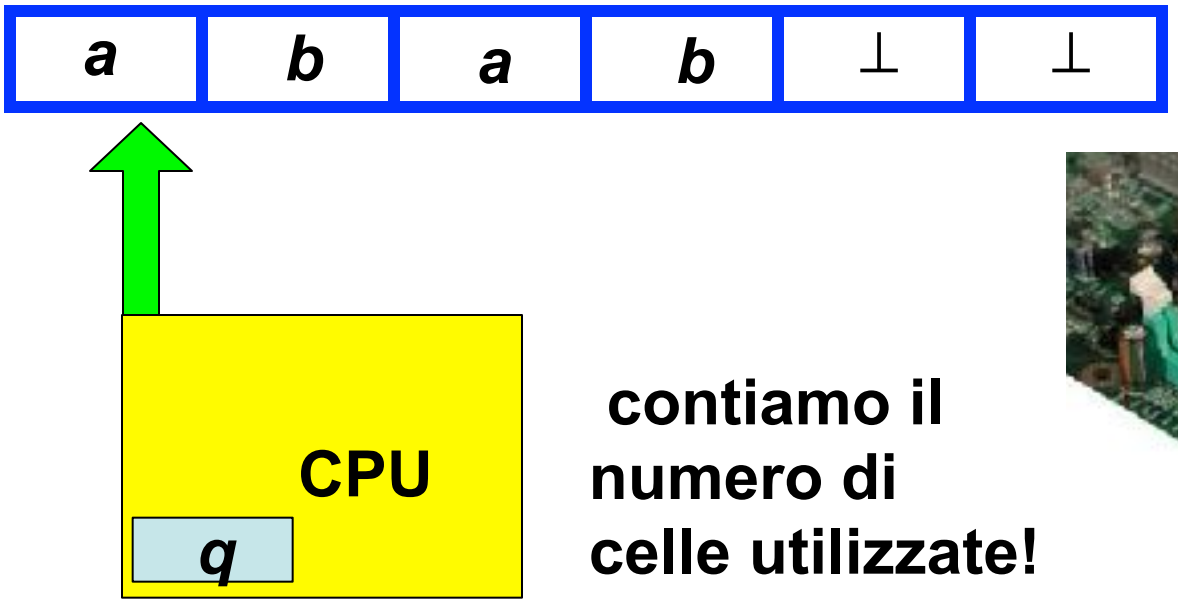
Per un numero
per es 17

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 $T_M : \{a,b\}^* \rightarrow N$ tale che

$Celle_M(x)$ = n° celle utilizzate su input x

Definiamo la funzione

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

Quindi $s_M(n) \geq n$

Il caso peggiore!

Relazione tra spazio e tempo: conoscendo il tempo limito lo spazio

Conoscendo la complessità di tempo, $t_T(n)$, possiamo limitare superiormente quella di spazio, $s_T(n)$,?

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 possiamo limitare superiormente quella di tempo?

Relazione tra spazio e tempo: conoscendo lo spazio limito il tempo

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 $s_T(n)$.

Con ogni mossa la TM cambia la configurazione in cui si trova. Il numero di tutte le possibili configurazioni, con contenuto del nastro di lunghezza al più $s_T(n)$, fornisce quindi un limite superiore alla complessità di tempo di una TM.

Relazione tra spazio e tempo: dallo spazio al tempo

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

$$q * s_T(n) * d^{s_T(n)} \leq 2^{\lg q} * 2^{\lg(s_T(n))} * 2^{\lg d * s_T(n)}$$

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{la somma dei n° delle celle utilizzate su input x di lunghezza n, 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 $O((t_T(n))^2)$.

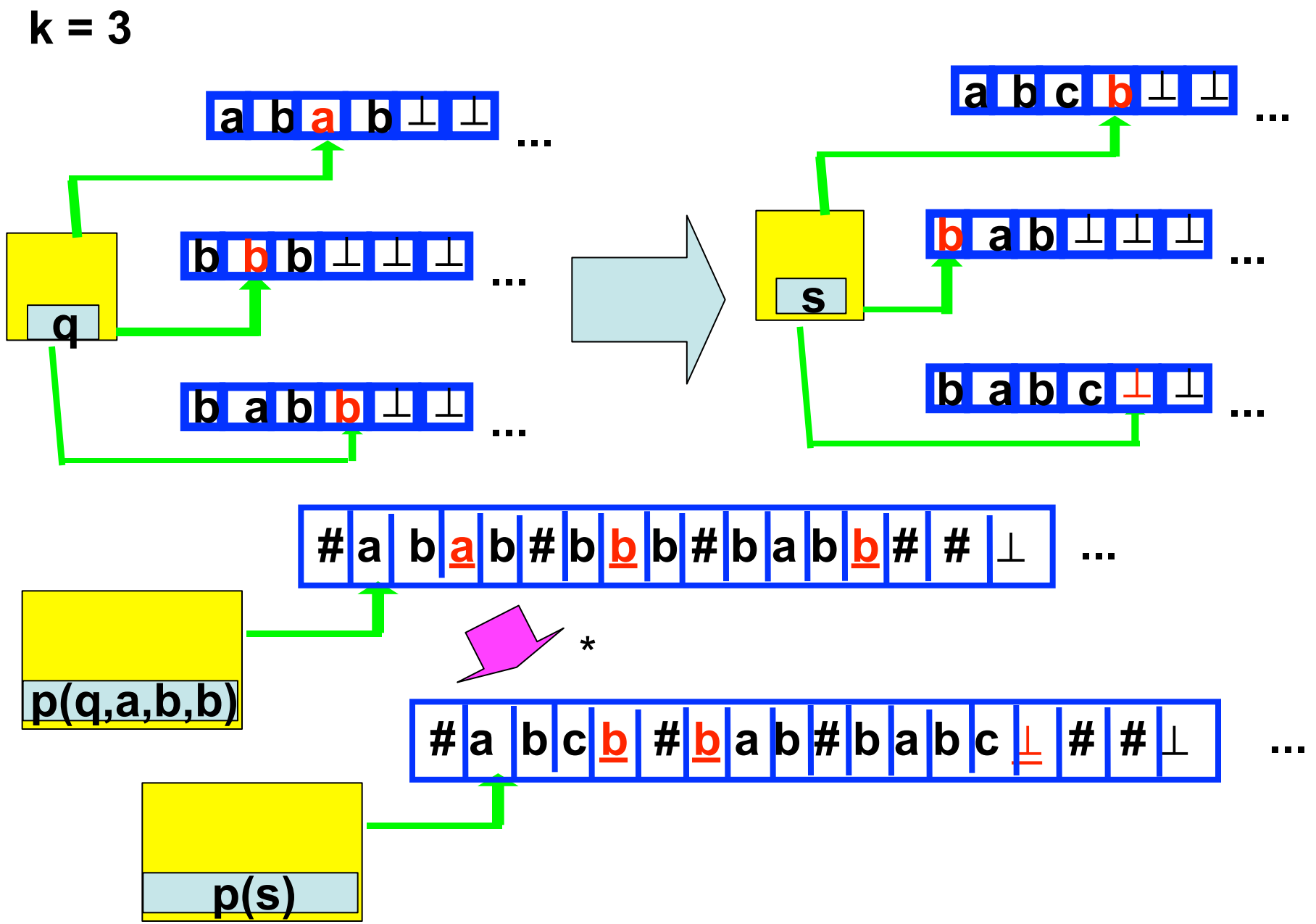
Prova. Ricordiamo che la TM a un solo nastro inizializza il proprio nastro per predisporre la simulazione della macchina a k nastri.

Alla fine di questa fase avremo questo contenuto sul nastro:

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

dove $a_1...a_n$ è la stringa input. Questo costa O(n).

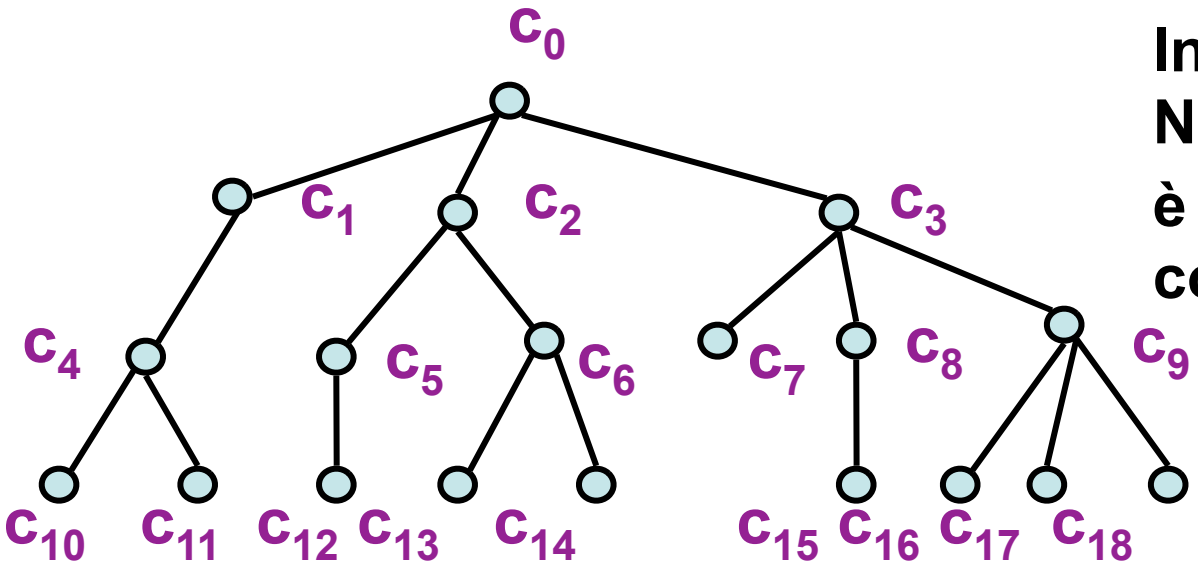
Simulazione di una mossa



Complessità di tempo di una NTM M

Definiamo la funzione $NPassi_M : \{a,b\}^* \rightarrow N$ tale che

$NPassi_M(x)$ = max{n° passi su input x}



In altre parole $NPassi_M(x) : \{a,b\}^* \rightarrow N$ è l'altezza dell'albero di computazione di x.

Definiamo la funzione $t_M : N \rightarrow N$ tale che

$t_M(n)$ = max{$NPassi_M(x)$ per x di lunghezza n}

In altre parole $t_M(n)$ è l'altezza del più alto albero di computazione su un input di dimensione n.

Il caso peggiore!

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

Complessità di spazio di una NTM M

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

Definiamo la funzione $s_M : N \rightarrow N$ tale che:

$s_M(n) = \max\{NCelle_M(x)$ su input x 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!

La TM M' equivalente a una NTM M

M': input: $a_1 \ldots a_n$

la configurazione iniziale è: $q_0 a_1 \ldots a_n q_0 \perp q_0 \perp$, input sul primo nastro e gli altri due vuoti

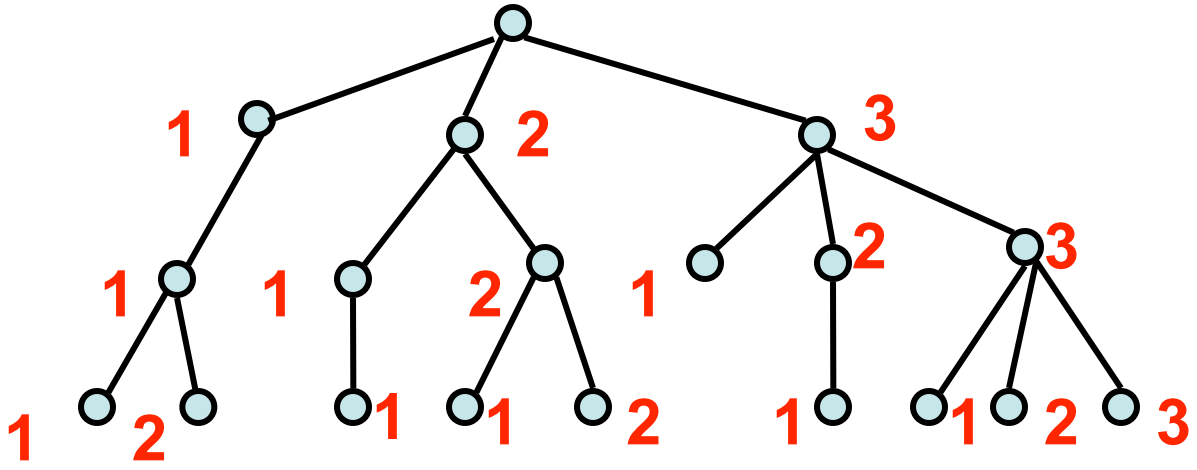
1. M' copia l'input sul secondo nastro e inizializza il terzo nastro alla stringa 1, configurazione: $q_0 a_1 \ldots a_n q_0 a_1 \ldots a_n q_0 1$

2. M' esegue una computazione di M sul nastro 2 facendo la mossa determinata dal simbolo in lettura sul terzo nastro. Se la mossa porta a una configurazione di accettazione di M, anche M' accetta, se porta a una di rifiuto M' va alla fase 3. Se il simbolo letto sul terzo nastro non corrisponde a una scelta o è il simbolo di cella vuota M' va alla fase 3.

3.esegui la TM M_0 per generare sul terzo nastro la successiva sequenza guida e torna al passo 2

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.

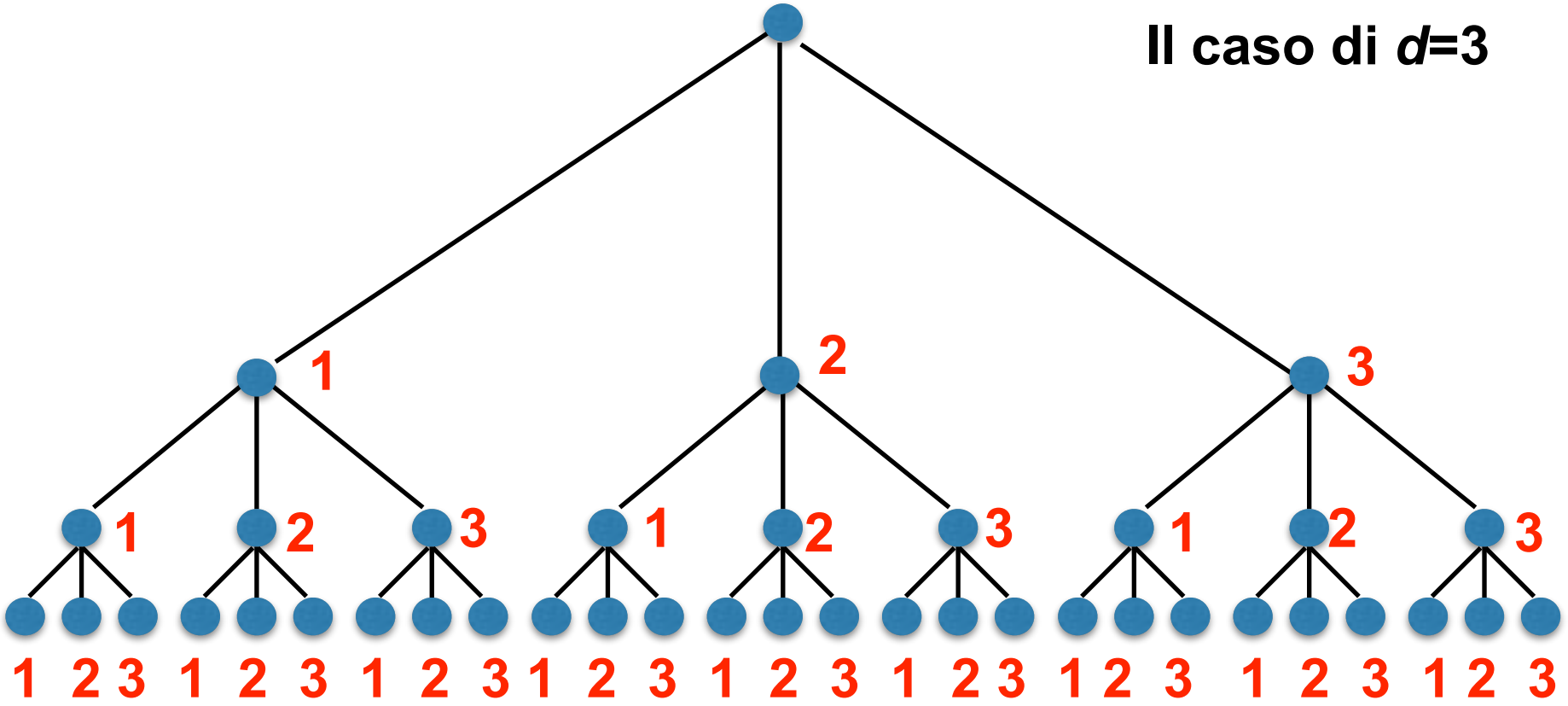
Il contatore verrà incrementato di 1 ogni volta che la computazione guidata dalla sequenza sul terzo nastro produce uno stato di rifiuto o fallisce. Il contatore viene confrontato con d^n, quando si genera sul IV nastro la sequenza d^n (cioè si simula l'ultimo cammino di computazione di lunghezza n)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

tempo e nontederminismo

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

tempo e nontederminismo

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.



tempo e nontederminismo

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

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

Quindi nel passaggio da una NTM T alla TM T1 equivalente la complessità di tempo diventa

$$t_{T1}(n) = O(d^{t_T(n)}) = 2^{O(t_T(n))}.$$

Teorema. Se L è riconosciuto da una NTM T con complessità di tempo $t_T(n)$, allora L è riconosciuto da una TM T1 con complessità di tempo

$$t_{T1}(n) = 2^{O(t_T(n))}$$

Classi di complessità di tempo

Chiamiamo

TIME(f(n)

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

Chiamiamo

NTIME(f(n)

la classe dei linguaggi decisi da una NTM in tempo O(f(n))

Classi di complessità di spazio

Chiamiamo

SPACE(f(n))

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

Chiamiamo

NSPACE(f(n))

la classe dei linguaggi decisi da una NTM in spazio O(f(n))

Classi notevoli

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

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

La classe dei linguaggi decisi in tempo polinomiale da una NTM

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

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 cumuni 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 matching massimo può essere risolto con 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

La tesi di Cobhan-Edmonds si rafforza con 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 un tempo polinomialmente correlato.

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

O in termini più informali:
Church-Turing estesa: ogni modello di calcolo generale e fisicamente realizzabile definisce lo stesso insieme di problemi efficientemente risolvibili.

La classe P

Ancora 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!) e il problema del vuoto.**
- **Ben noti problemi sui grafi: connessione di un grafo non diretto, esistenza di un cammino tra due nodi,etc.**

PRIMES

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

Istanza 1. z = 23.536.481.273.
risposta NO: z = 104.729 x 224.737.

Istanza 2. z = 23.536.481.277.
risposta SI

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

Primes è in P

FACTOR

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 x 224.737.

Istanza 2. z = 23.536.481.273 e U = 100.000.
risposta NO : 104.729 x 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 deterministico esponenziale per FACTOR

```
CercaFattori(x,U)
 prec: (U≤√x)
      k = 2
      while k ≤ U
           if x mod k = 0 then esci e  "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.

Algoritmo non deterministico e polinomiale per FACTOR

CercaFattoriNonDet(x,U)
prec: (U≤√x)
"non deterministicamente prendi un numero k tra 2 e U"
if x mod 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.

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 inutilizzabili**

Se $P \subset NP$, allora

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

Una vignetta di Pavel Pudlák